Artificial Neural Networks with Java

Tools for Building Neural Network Applications

Igor Livshin

Apress®

Artificial Neural Networks with Java

Tools for Building Neural Network Applications

Igor Livshin

Apress®

Artificial Neural Networks with Java: Tools for Building Neural Network Applications

Igor Livshin
Chicago, IL, USA

ISBN-13 (pbk): 978-1-4842-4420-3 ISBN-13 (electronic): 978-1-4842-4421-0
https://doi.org/10.1007/978-1-4842-4421-0

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Celestin Sureh John
Development Editor: Matthew Moodie
Coordinating Editor: Aditee Mirashi

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-4420-3. For more detailed information, please visit www.apress.com/source-code.

Printed on acid-free paper

For Asa and Sid Ostrow.

Table of Contents

About the Author

Igor Livshin worked for many years as senior J2EE architect at two large insurance companies, Continental Insurance and Blue Cross & Blue Shield of Illinois, developing large-scale enterprise applications. He published his first book, *Web Studio Application Developer 5.0*, in 2003. He currently works as a senior architect at Dev Technologies Corp, specializing in developing neural network applications. Igor has a master's degree in computer science from the Institute of Technology in Odessa, Russia/Ukraine.

About the Technical Reviewer

Jeff Heaton, PhD, is a vice president and data scientist at Reinsurance Group of America (RGA) and an adjunct faculty member at Washington University in St. Louis. With 20 years of life insurance industry experience, Jeff has crafted a variety of InsurTech solutions using machine learning technologies such as deep learning, random forests, and gradient boosting, among others. Utilizing electronic health record (EHR) data such as FIHR, ICD10, SNOMED, and RxNorm, he works closely with all stages of model development from inception to ultimate deployment. Jeff is the author of several books and has published peer-reviewed research through IEEE and the ACM. He is a senior member of the Institute of Electrical and Electronics Engineers (IEEE) and Fellow of the Life Management Institute (FLMI). Jeff holds a PhD in computer science from Nova Southeastern University in Ft. Lauderdale, Florida.

Acknowledgments

I would like to thank Apress and its wonderful team of editors and tech reviewers for their dedication and hard work in publishing this book.

Introduction

Artificial intelligence is a rapidly advancing area of computer science. Since the invention of computers, we have observed an intriguing phenomenon. Tasks that are difficult for a human (such as heavy computations, searching, memorizing large volumes of data, and so on) are easily done by computers, while tasks that humans are naturally able to do quickly (such as recognizing partially covered objects, intelligence, reasoning, creativity, invention, understanding speech, scientific research, and so on) are difficult for computers.

Artificial intelligence as a discipline was invented in 1950s. Initially it failed because of a lack of backpropagation and an automated means of training. In the 1980s, it failed because of an inability to form its own internal representations, solved later by deep learning and the availability of more powerful computers.

A new nonlinear network architecture was developed after that second failure, and the tremendous increase in machines' computing power finally contributed to the phenomenal success of AI in 1990s. Gradually, AI became capable of solving many industrial-grade problems such as image recognition, speech recognition, natural language processing, pattern recognition, prediction, classification, self-driving cars, robotic automation, and so on.

The tremendous success of AI has recently triggered all types of unwarranted speculations. You will find discussions about robots of the near future matching and exceeding the intelligence of humans. However, currently, AI is a set of clever mathematical and processing methods that let computers learn from the data they process and apply this knowledge to solve many important tasks. A lot of things that belong to humans such as intelligence, emotion, creativity, feeling, reasoning, and so on, are still outside the possibility of AI.

Still, AI is rapidly changing. In recent years, computers have become so good at playing chess that they reliably beat their human counterparts. That is not surprising, because their creators taught the programs centuries of accumulated human experience in chess. Now, machines compete against each other in the world computer chess championship. One of the best chess-playing programs, called Stockfish 8, won the world computer chess championship in 2016.

In 2017 Google developed a chess-playing program called AlphaZero, which defeated the Stockfish 8 program in the 2017 world computer chess championship. The amazing part of this is that no one taught AlphaZero the chess strategies, like had been done during the development of other chess-playing programs. Instead, it used the latest machine learning principles to teach itself chess by playing against itself. It took the program four hours of learning chess strategies (while playing against itself) to beat Stockfish 8. Self-teaching is the new milestone achievement of artificial intelligence.

AI has many branches. This book is dedicated to one of them: neural networks. Neural networks enable computers to learn from observational data and make predictions based on that knowledge. Specifically, this book is about neural network training and using it for function approximation, prediction, and classification.

What This Book Covers

This practical how-to book covers many aspects of developing neural network applications. It starts from scratch explaining how neural networks work and proceeds with an example of training a small neural network, making all the calculations manually. This book covers the internals of front and backward propagation and facilitates understanding of the main principles of neural network processing. It quickly familiarizes you with all the basic principles of the front and backward propagation techniques. The book also teaches you how to prepare data to be used in neural network development and suggests various data preparation methods for many unconventional neural network processing tasks.

The next big topic discussed in the book is using Java for neural network processing. Most books that teach AI use Python as the developing language; however, Java is the most widely used programming language. It is faster than Python, and it allows you to develop projects on one computer and run them on many different machines supporting the Java environment. In addition, when artificial intelligence processing is part of an enterprise application, no other languages can compete with Java.

The book uses a Java framework called Encog and shows all the details of how to use it for developing large-scale neural network applications. The book also discusses the difficulties of approximating complex noncontinuous functions as well as continuous functions with complex topologies, and it introduces my micro-batch method that solves this issue.

The step-by-step approach includes plenty of examples, diagrams, and screenshots to facilitate your learning experience. Each topic discussed in the book comes with corresponding examples of practical development, with many tips for each topic.

All the examples in this book were developed for the Windows 7/10 platform, although being developed in Java, they can run on any Java-supporting environment. All the Java tools described in this book are free to use and can be downloaded from the Internet.

Who This Book Is For?

The book is for professional developers who are interested in learning neural network programming in a Java environment. The book should also be of interest to more experienced AI developers who will find here more advanced topics and tips related to the development of more complex neural network applications. The book also includes my latest research of neural network approximation of noncontinuous functions and continuous functions with complex topologies.

CHAPTER 1

Learning About Neural Networks

The artificial intelligence neural network architecture schematically mimics a human brain network. It consists of layers of neurons directionally connected to each other. Figure 1-1 shows a schematic image of a human neuron.

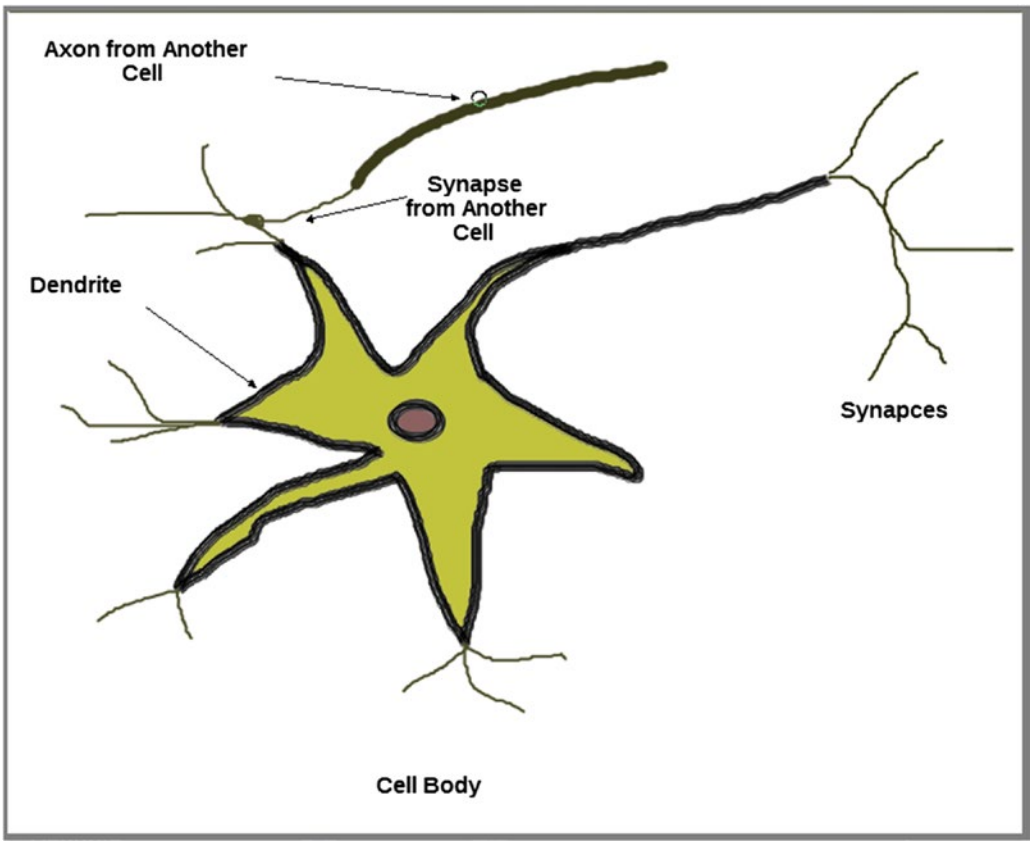


Figure 1-1. *Schematic image of a human neuron*

I. Livshin, *Artificial Neural Networks with Java*, https://doi.org/10.1007/978-1-4842-4421-0_1

Biological and Artificial Neurons

A biological neuron (on a simplified level) consists of a cell body with a nucleus, axon, and synapses. Synapses receive impulses, which are processed by the cell body. The cell body sends a response over an axon to its synapses, which are connected to other neurons. Mimicking the biological neuron, an artificial neuron consists of a neuron body and connections to other neurons (see Figure 1-2).

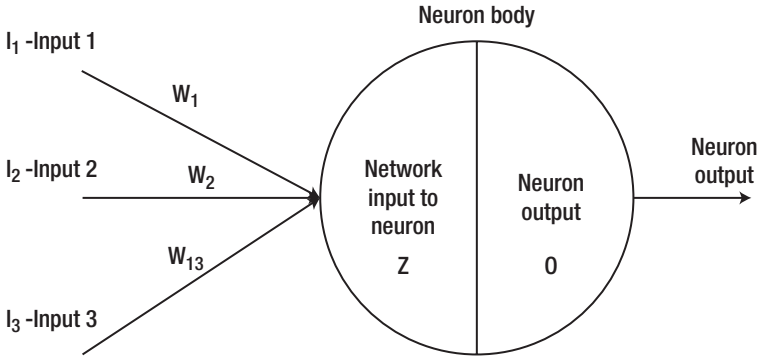


Figure 1-2. *Single artificial neuron*

Each input to a neuron is assigned a weight, W. The weight assigned to a neuron indicates the impact this input makes in the calculation of the network output. If the weight assigned to neuron1 (W_1) is greater than the weight assigned to neuron2 (W_2), then the impact of the input from neuron1 on the network output is more significant than from neuron2.

The body of a neuron is depicted as a circle divided into two parts by a vertical line. The left part is called the *network input* to the neuron, and it shows the part of the calculation that the neuron body performs. This part is typically marked on network diagrams as Z. For example, the value of Z for the neuron shown in Figure 1-2 is calculated as a sum of each input to the neuron multiplied by the corresponding weight and finally adding the bias. That is the linear part of calculation (Equation 1-1).

$$Z = W_1^* I_1 + W_2^* I_2 + W_3^* I_3 + B_1 \quad (1\text{-}1)$$

Activation Functions

To calculate output O from the same neuron (Figure 1-2), you can apply a special nonlinear function (called the *activation function*, σ) to the linear part of calculation Z (Equation 1-2).

$$O = \sigma(Z) \tag{1-2}$$

There are many activation functions that are used for networks. Their usage depends on various factors, such as the interval where they are well-behaved (not saturated), how fast the function changes when its argument changes, and simply your preferences. Let's look at the one of the most frequently used activation functions; it's called *sigmoid*. The function has the formula shown in Equation 1-3.

$$\sigma(Z) = \frac{1}{1+e^{-z}} \tag{1-3}$$

Figure 1-3 shows the graph of a sigmoid activation function.

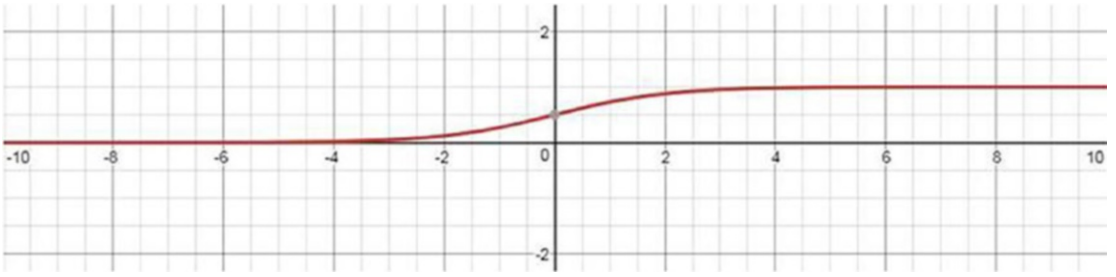


Figure 1-3. *Graph of a sigmoid function*

As shown on the graph in Figure 1-3, the sigmoid function (sometimes also called the *logistic* function) best behaves on the interval [-1, 1]. Outside of this interval, it quickly saturates, meaning that its value practically does not change with the change of its argument. That is why (as you will see in all this book's examples) the network's input data is typically normalized on the interval [-1, 1].

Some activation functions are well-behaved on the interval [0, 1], so the input data is correspondingly normalized on the interval [0, 1]. Figure 1-4 lists the most frequently used activation functions. It includes the function name, plot, equation, and derivative. This information will be useful when you start calculating various parts within a network.

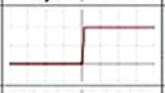
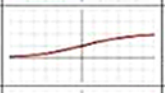
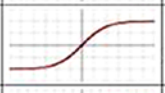
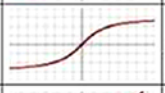
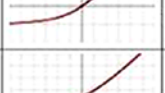
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Figure 1-4. *Activation functions*

Using a specific activation function depends on the function being approximated and on many other conditions. Many recent publications suggest using tanh as the activation function for hidden layers, using the linear activation function for regression, and using the softmax function for classification. Again, these are just general recommendations. I recommend you experiment with various activation functions for your project and select those that produce the best result.

For this book's examples, I found experimentally that the tanh activation function works best. It is also well-behaved (as the sigmoid activation function) on the interval [-1,1], but its rate of change on this interval is faster than for the sigmoid function. It also saturates slower. I use tanh in almost all the examples of this book.

Summary

The chapter introduced you to the form of artificial intelligence called *neural networks*. It explained all the important concepts of neural networks such as layers, neurons, connections, weights, and activation functions. The chapter also explained the conventions used for drawing a neural network diagram. The following chapter shows all the details of neuron network processing by explaining how to manually calculate all the network results. For simplicity, two terms—*neural network* and *network*—will be used interchangeably for the rest of this book.

CHAPTER 2

Internal Mechanics of Neural Network Processing

This chapter discusses the inner workings of neural network processing. It shows how a network is built, trained, and tested.

Function to Be Approximated

Let's consider the function $y(x) = x^2$, as shown in Figure 2-1; however, we'll pretend that the function formula is not known and that the function was given to us by its values at four points.

I. Livshin, *Artificial Neural Networks with Java*, https://doi.org/10.1007/978-1-4842-4421-0_2

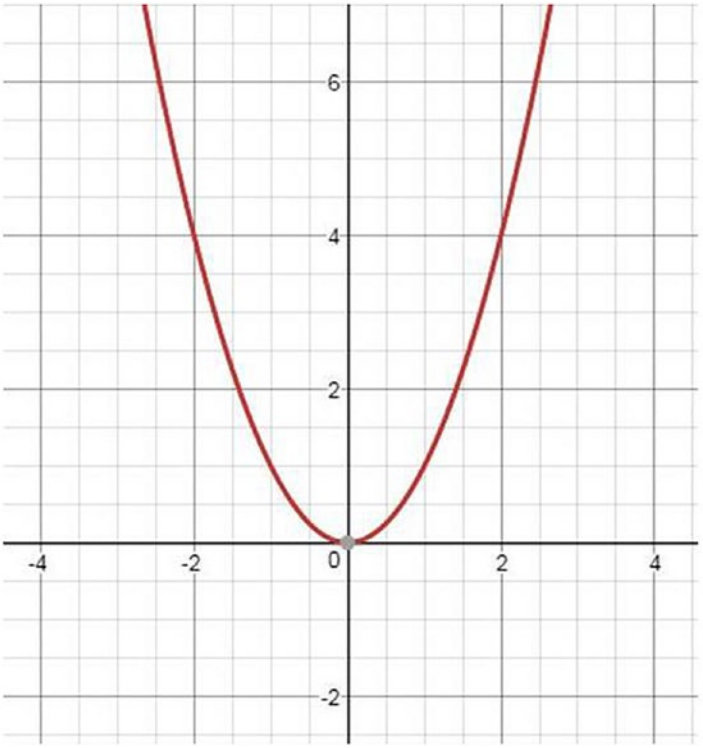


Figure 2-1. *Graph of the function*

Table 2-1 lists the function values at the four points.

Table 2-1. *Function Values Given at Four Points*

x	f(x)
1	1
3	9
5	25
7	49

In this chapter, you will build and train a network that can be used for predicting the values of the function at some arguments (x) that are not being used for training. To be able to get the value of the function at nontrained points (but within the training range), you first need to approximate this function. When the approximation is done, you can then find the values of the function at any points of interest. That's what the network is used for, because the network is the universal approximation mechanism.

Network Architecture

How is a network built? A network is built by including layers of neurons. The first layer on the left is the *input layer*, and it contains the neurons that receive input from the outside. The last layer on the right is the *output layer*, and it contains the neurons that carry the output of the network. One or more *hidden layers* are located between the input and output layers. Hidden-layer neurons are used for performing most of the calculations during the approximation of the function. Figure 2-2 shows the network diagram.

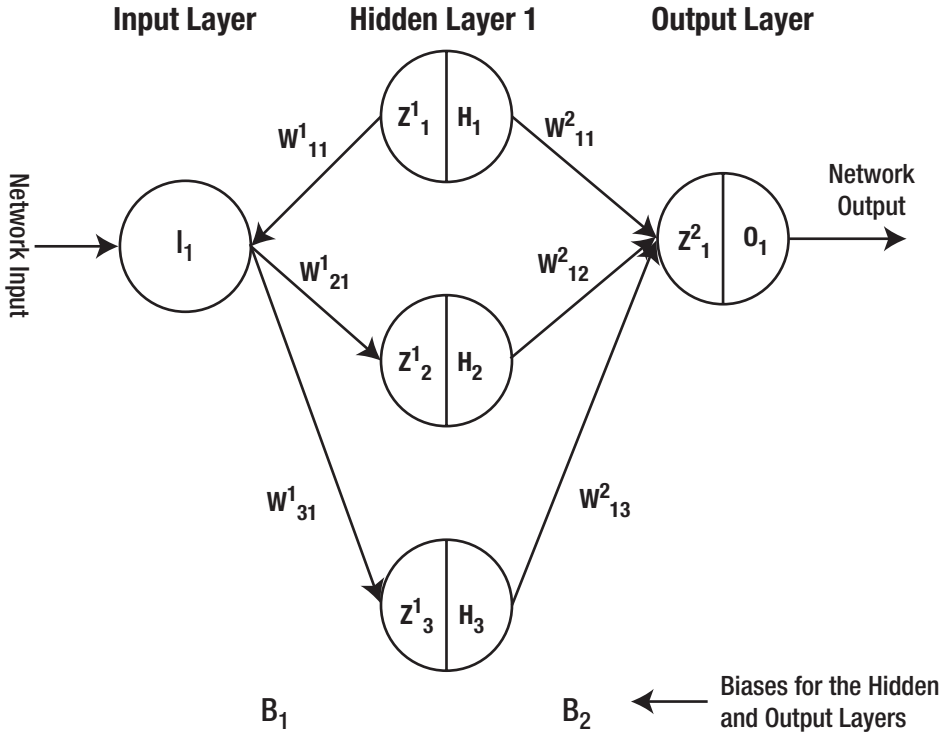


Figure 2-2. *Neural network architecture*

Connections are drawn from the neurons in the previous layer to the neurons of the next layer. Each neuron in the previous layer is connected to all the neurons of the next layer. Each connection carries a weight, and each weight is numbered with two indexes. The first index is the receiving neuron number, and the second index is the sending neuron number. For example, the connection between the second neuron in the hidden layer (H_2) and the only neuron in the input layer (I_1) is assigned the weight W^1_{21}. The superscript 1 indicates the layer number of the sending neuron. Each layer is assigned

a *bias*, which is similar to the weight assigned to a neuron but applied to a layer level. Biases make the linear part of each neuron output calculation more flexible in matching the approximated function topology.

When the network processing starts, the initial values for weights and biases are usually randomly set. Typically, to determine the number of neurons in a hidden layer, you double the number of neurons in the input layer and add the number of neurons in the output layer. In our case, it is (1*2+1 = 3), or three neurons. The number of hidden layers to be used in the network depends on the complexity of the function to be approximated. Typically, one hidden layer is sufficient for a smooth continuous function, and more hidden layers are needed for more complex function topology. In practice, the number of layers and neurons in the hidden layers that leads to the best approximation results is determined experimentally.

The network processing consists of two passes: a forward pass and a backward pass. In the forward pass, the calculation moves from left to right. For each neuron, the network gets the input to the neuron and calculates the output from the neuron.

Forward-Pass Calculation

The following calculations give the output from neurons H1, H2, and H3:

Neuron H_1

$$Z_1^1 = W_{11}^{1} * I_1 + B_1 * 1$$
$$H_1 = \sigma\left(Z_1^1\right)$$

Neuron H_2

$$Z_2^1 = W_{21}^{1} * I_1 + B_1 * 1$$
$$H_2 = \sigma\left(Z_2^1\right)$$

Neuron H_3

$$Z_3^1 = W_{31}^{1} * I_1 + B_1 * 1$$
$$H_3 = \sigma\left(Z_3^1\right)$$

These values are used when processing neurons in the next layer (in this case, the output layer):

Neuron O_1

$$Z_1^2 = W_{11}^{2*} H_1 + W_{12}^{2*} H_2 + W_{13}^{2*} H_3 + B_2^* 1$$
$$O_1 = \sigma\left(Z_1^2\right)$$

The calculation made in the first pass gives the output of the network (called the *network predicted value*). When training a network, you use the known output for the training points called the *actual* or *target* values. By knowing the output value that the network should produce for the given input, you can calculate the network error, which is the difference between the target value and the network calculated error (predicted value). For the function you want to approximate on this example, the actual (target) values are shown in Table 2-2, column 2.

Table 2-2. *Input Data Set for the Example*

x	f(x)
1	**1**
3	**9**
5	**25**
7	**49**

A calculation is done for each record in the input data set. For example, processing the first record of the input data set is done by using the following formulas.

Input Record 1

Here are the formulas for the first input record:

Neuron H_1

$$Z_1^1 = W_{11}^{1*} I_1 + B_1^* 1.00 = W_{11}^{1*} 1.00 + B_1^* 1.00$$
$$H_1 = \sigma\left(Z_1^1\right)$$

Neuron H_2

$$Z_2^1 = W_{21}^{1*} I_1 + B_1^* 1.00 = W_{21}^{1*} 1.00 + B_1^* 1.00$$
$$H_2 = \sigma\left(Z_2^1\right)$$

Neuron H_3

$$Z_3^1 = W_{31}^{1*} I_1 + B_1^* 1.00 = W_{31}^{1*} 1.00 + B_1^* 1.00$$
$$H_3 = \sigma\left(Z_3^1\right)$$

Neuron O_1

$$Z_1^2 = W_{11}^{2*} H_1 + W_{12}^{2*} H_2 + W_{13}^{2*} H_3 + B_2^* 1.00$$
$$O_1 = \sigma\left(Z_1^2\right)$$

Here is the error for record 1:

$$E_1 = \sigma\left(Z_1^2\right) - \text{Target value for Record } 1 = \sigma\left(Z_1^2\right) - 1.00$$

Input Record 2

Here are the formulas for the second input record:

Neuron H_1

$$Z_1^1 = W_{11}^{1*} I_1 + B_1^* 1.00 = W_{11}^{1*} 3.00 + B_1^* 1.00$$
$$H_1 = \sigma\left(Z_1^1\right)$$

Neuron H_2

$$Z_2^1 = W_{21}^{1*} I_1 + B_1^* 1.00 = W_{21}^{1*} 3.00 + B_1^* 1.00$$
$$H_2 = \sigma\left(Z_2^1\right)$$

Neuron H_3

$$Z_3^1 = W_{31}^{1*} I_1 + B_1^* 1.00 = W_{31}^{1*} 3.00 + B_1^* 1.00$$
$$H_3 = \sigma\left(Z_3^1\right)$$

Neuron O_1

$$Z_1^2 = W_{11}^{2*} H_1 + W_{12}^{2*} H_2 + W_{13}^{2*} H_3 + B_2^* 1.00$$
$$O_1 = \sigma\left(Z_1^2\right)$$

Here is the error for record 2:

$$E_1 = \sigma\left(Z_1^2\right) - \text{Target value for Record } 2 = \sigma\left(Z_1^2\right) - 9.00$$

Input Record 3

Here are the formulas for the third input record:

Neuron H_1

$$Z_1^1 = W_{11}^{1*} I_1 + B_1^* 1.00 = W_{11}^{1*} 5.00 + B_1^* 1.00$$
$$H_1 = \sigma\left(Z_1^1\right)$$

Neuron H_2

$$Z_2^1 = W_{21}^{1*} I_1 + B_1^* 1.00 = W_{21}^{1*} 5.00 + B_1^* 1.00$$
$$H_2 = \sigma\left(Z_2^1\right)$$

Neuron H_3

$$Z_3^1 = W_{31}^{1*} I_1 + B_1^* 1.00 = W_{31}^{1*} 5.00 + B_1^* 1.00$$
$$H_3 = \sigma\left(Z_3^1\right)$$

Neuron O_1

$$Z_1^2 = W_{11}^{2*} H_1 + W_{12}^{2*} H_2 + W_{13}^{2*} H_3 + B_2^* 1.00$$
$$O_1 = \sigma\left(Z_1^2\right)$$

Here is the error for record 3:

$$E_1 = \sigma\left(Z_1^2\right) - \text{Target value for Record } 3 = \sigma\left(Z_1^2\right) - 25.00$$

Input Record 4

Here are the formulas for the fourth input record:

Neuron H_1

$$Z_1^1 = W_{11}^{1*} I_1 + B_1^* 1.00 = W_{11}^{1*} 7.00 + B_1^* 1.00$$
$$H_1 = \sigma\left(Z_1^1\right)$$

Neuron H_2

$$Z_2^1 = W_{21}^{1*} I_1 + B_1^* 1.00 = W_{21}^{1*} 7.00 + B_1^* 1.00$$
$$H_2 = \sigma\left(Z_2^1\right)$$

Neuron H_3

$$Z_3^1 = W_{31}^{1*} I_1 + B_1^* 1.00 = W_{31}^{1*} 7.00 + B_1^* 1.00$$
$$H_3 = \sigma\left(Z_3^1\right)$$

Neuron O_1

$$Z_1^2 = W_{11}^{2*} H_1 + W_{12}^{2*} H_2 + W_{13}^{2*} H_3 + B_2^* 1.00$$
$$O_1 = \sigma\left(Z_1^2\right)$$

Here is the error for input record 4:

$$E_1 = \sigma\left(Z_1^2\right) - \text{Target value for Record } 4 = \sigma\left(Z_1^2\right) - 49.00$$

When all records have been processed for this batch (and the batch here is the entire training set), that point in the processing is called the *epoch*. At that point, you can take the average of the network errors for all records—as in E =(E1 + E2 + E3 + E4)/4—and that is the error at the current epoch. The average error includes each error sign. Obviously, the error at the first epoch (with randomly selected weights/biases) will be too large for a good function approximation; therefore, you need to reduce this error to the acceptable (desired) value called the *error limit*, which you set at the beginning of processing. Reducing the network error is done in the backward pass (also called *backpropagation*). The error limit is determined experimentally. The error limit is set to the minimum error that the network can reach, but not easily. The network should work hard to reach such an error limit. The error limit is explained in more detail in many code examples throughout this book.

Backpropagation-Pass Calculations

How can the network error be reduced? Obviously, the initial weight and bias values are randomly set, and they are not good. They lead to a significant error for epoch 1. You need to adjust them in a way that their new values will lead to a smaller network-calculated error. Backpropagation does this by redistributing the error between all network neurons in the output and hidden layers and by adjusting their initial weight values. Adjustment is also done for each layer bias.

To adjust the weight of each neuron, you calculate the partial derivative of the error function with respect to the neuron's output. For example, the calculated partial derivative for neuron O_1 is $\frac{\partial E}{\partial O_1}$. Because the partial derivative points to the direction of the increased function value (but you need to decrease the value of the error function), so the weight adjustment should be done in the opposite direction.

$$Adjusted\ value\ of\ weight = original\ value\ of\ weight - \eta^* \frac{\partial E}{\partial O_1}$$

Here η is the learning rate for the network, and it controls how fast the network learns. Its value is typically set to be between 0.1 and 1.0.

A similar calculation is done for the bias of each layer. For bias B_1, if the calculated partial derivative is $\frac{\partial E}{\partial \boldsymbol{B}_1}$, then the adjusted bias is calculated as follows:

$$Adjusted\ value\ of\ bias\ \mathrm{B}_1 = Original\ value\ of\ bias\ \mathrm{B}_1 - \eta^* \frac{\partial E}{\partial B_1}$$

By repeating this calculation for each network neuron and each layer bias, you obtain a new set of adjusted weight/bias values. Having a new set of weight/bias values, you return to the forward pass and calculate the new network output using the adjusted weights/biases. You also recalculate the network output error.

Because you adjusted the weights/biases in the direction opposite to the gradient (partial derivatives), the new network-calculated error should decrease. You repeat both the forward and backward passes in a loop until the error becomes less than the error limit. At that point, the network is trained, and you save the trained network on disk. The trained network includes all weight and bias parameters that approximate the predicted function value with the needed degree of precision. In the next chapter, you will learn how to manually process some examples and see all the detailed calculations. However, before doing that, you need to refresh your knowledge of the function derivative and gradient.

Function Derivative and Function Divergent

A derivative of a function is defined as follows:

$$\frac{\partial f}{\partial x} = \lim_{n \to 0} \frac{f(x+dx) - f(x)}{dx}$$

where:

∂x is a small change in a function argument.

f(x) is the value of a function before changing the argument.

f(x + ∂x) is the value of a function after changing the argument.

The function derivative shows the rate of change of a single-variable function f(x) at point x. The gradient is the derivative (rate of change) of a multivariable function f(x, y, z) at the point (x, y, z). The gradient of a multivariable function f(x, y, z) is a product of components calculated for each direction ($\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$). Each component

is called the *partial derivative* of function f(x, y, z) with respect to the specific variable (direction) x, y, z.

The gradient at any function point always points to the direction of the greatest increase of a function. At the local maximum or local minimum, the gradient is zero because there is no single direction of increase at such locations. When you are searching for a function minimum (for example, for an error function) that you want to minimize, you move in the direction opposite to the gradient.

There are several rules for calculating derivatives.

This is the power rule: $\frac{\partial}{\partial x}\left(u^{a}\right)=a*u^{a-1*}\frac{\partial u}{\partial x}$

This is the product rule: $\frac{\partial\left(u^{*}v\right)}{\partial x}=u^{*}\frac{\partial v}{\partial x}+v^{*}\frac{\partial u}{\partial x}$

This is the quotient rule: $\frac{\partial f}{\partial x}\left(\frac{u}{v}\right)=\frac{v^{*}\frac{\partial u}{\partial x}-u^{*}\frac{\partial v}{\partial x}}{v^{2}}$

The chain rule tells you how to differentiate a composite function.

It states that $\frac{\partial y}{\partial x}=\frac{\partial y}{\partial u}^{*}\frac{\partial u}{\partial x}$, where u = f(x).

Here's an example: $y=u^{8}$. $u=x^{2}+5$.

According to the chain rule,

$$\frac{\partial y}{\partial x}=\frac{\partial y}{\partial u}^{*}\frac{\partial u}{\partial x}=8u^{7*}2x=16x^{*}\left(x^{2}+5\right)^{7}$$

Most Commonly Used Function Derivatives

Figure 2-3 lists the most commonly used function derivatives.

$$\frac{d}{dx}(a) = 0$$

$$\frac{d}{dx}(x) = 1$$

$$\frac{d}{dx}(au) = a\frac{du}{dx}$$

$$\frac{d}{dx}(u + v - w) = \frac{du}{dx} + \frac{dv}{dx} - \frac{dw}{dx}$$

$$\frac{d}{dx}(uv) = u\frac{dv}{dx} + v\frac{du}{dx}$$

$$\frac{d}{dx}\left(\frac{u}{v}\right) = \frac{1}{v}\frac{du}{dx} - \frac{u}{v^2}\frac{dv}{dx}$$

$$\frac{d}{dx}(u^n) = nu^{n-1}\frac{du}{dx}$$

$$\frac{d}{dx}(\sqrt{u}) = \frac{1}{2\sqrt{u}}\frac{du}{dx}$$

$$\frac{d}{dx}\left(\frac{1}{u}\right) = -\frac{1}{u^2}\frac{du}{dx}$$

$$\frac{d}{dx}\left(\frac{1}{u^n}\right) = -\frac{n}{u^{n+1}}\frac{du}{dx}$$

$$\frac{d}{dx}[f(u)] = \frac{d}{du}[f(u)]\frac{du}{dx}$$

$$\frac{d}{dx}[\ln u] = \frac{d}{dx}[\log_e u] = \frac{1}{u}\frac{du}{dx}$$

$$\frac{d}{dx}[\log_a u] = \log_a e\frac{1}{u}\frac{du}{dx}$$

$$\frac{d}{dx}e^u = e^u\frac{du}{dx}$$

$$\frac{d}{dx}a^u = a^u \ln a\frac{du}{dx}$$

$$\frac{d}{dx}\left(u^v\right) = vu^{v-1}\frac{du}{dx} + \ln u \;\; u^v\frac{dv}{dx}$$

$$\frac{d}{dx}\sin u = \cos u\frac{du}{dx}$$

$$\frac{d}{dx}\cos u = -\sin u\frac{du}{dx}$$

$$\frac{d}{dx}\tan u = \sec^2 u\frac{du}{dx}$$

$$\frac{d}{dx}\cot u = -\csc^2 u\frac{du}{dx}$$

$$\frac{d}{dx}\sec u = \sec u\tan u\frac{du}{dx}$$

$$\frac{d}{dx}\csc u = -\csc u\cot u\frac{du}{dx}$$

Figure 2-3. *Commonly used derivatives*

It is also helpful to know the derivative of the sigmoid activation function because it is frequently used in the backpropagation step of network processing.

$$\sigma(Z) = 1/(1 + \exp(-Z))$$

$$\frac{\partial 6(Z)}{\partial Z} = Z^{*}(1 - Z)$$

The derivative of the sigmoid activation function gives the rate of change of the activation function at any neuron.

Summary

This chapter explored the inner machinery of neural network processing by explaining how all processing results are calculated. It introduced you to derivatives and gradients and described how these concepts are used in finding one of the error function minimums. The next chapter shows a simple example where each result is manually calculated. Simply describing the rules of calculation is not enough for understanding the subject because applying the rules to a particular network architecture is really tricky.

CHAPTER 3

Manual Neural Network Processing

In this chapter, you'll learn about the internals of neural network processing by seeing a simple example. I'll provide a detailed step-by-step explanation of the calculations involved in processing the forward and backward propagation passes.

Note All calculations in this chapter are based on the information in Chapter 2. If you have any issues reading Chapter 3, consult Chapter 2 for an explanation.

Example 1: Manual Approximation of a Function at a Single Point

Figure 3-1 shows the vector in the 3-D space.

I. Livshin, *Artificial Neural Networks with Java*, https://doi.org/10.1007/978-1-4842-4421-0_3

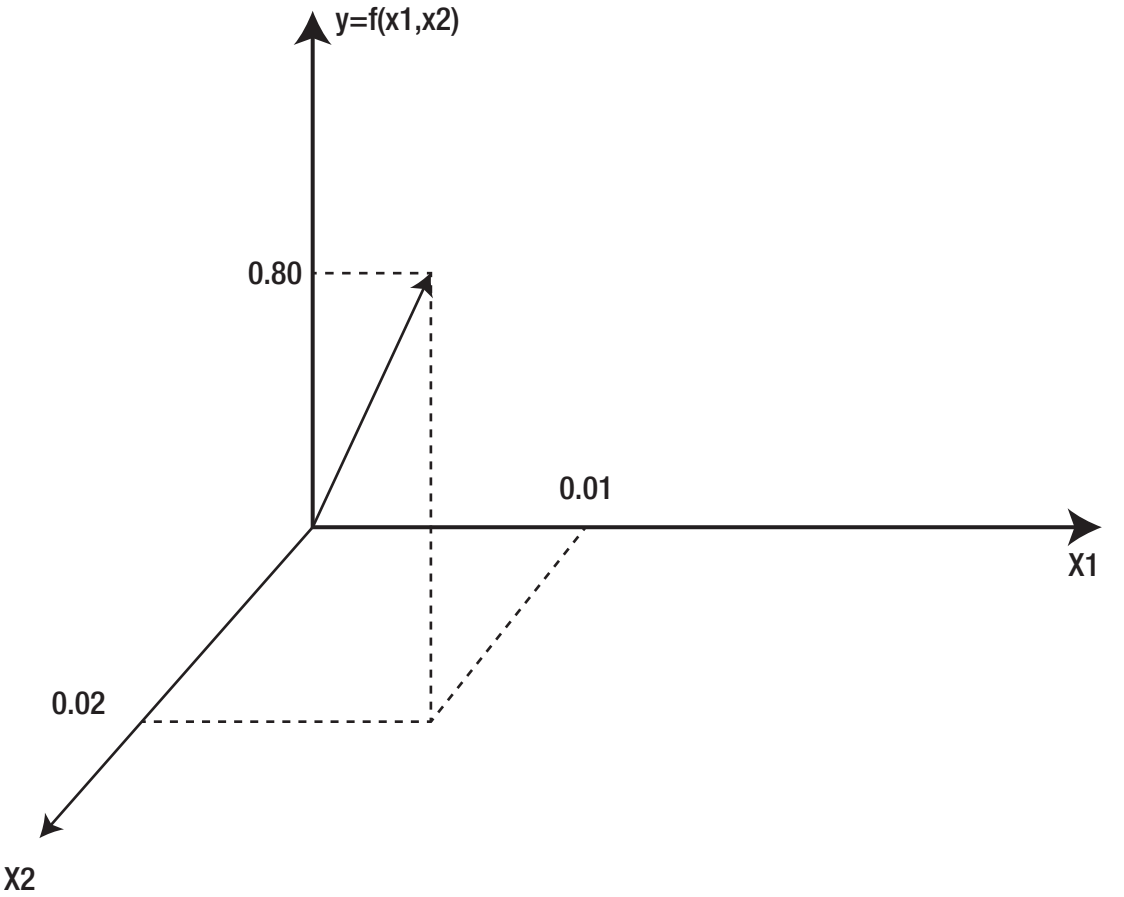


Figure 3-1. *Vector in 3-D space*

The vector represents the value of the function y=f(x1,x2), where x1 = 0.01 and x2 = 0.02.

$$y(0.01,0.02) = 0.80$$

Building the Neural Network

For this example, say you want to build and train a network that for a given input (x1=0.01, x2=0.02) calculates the output result y = 0.80 (the target value for the network).

Figure 3-2 shows the network diagram for the example.

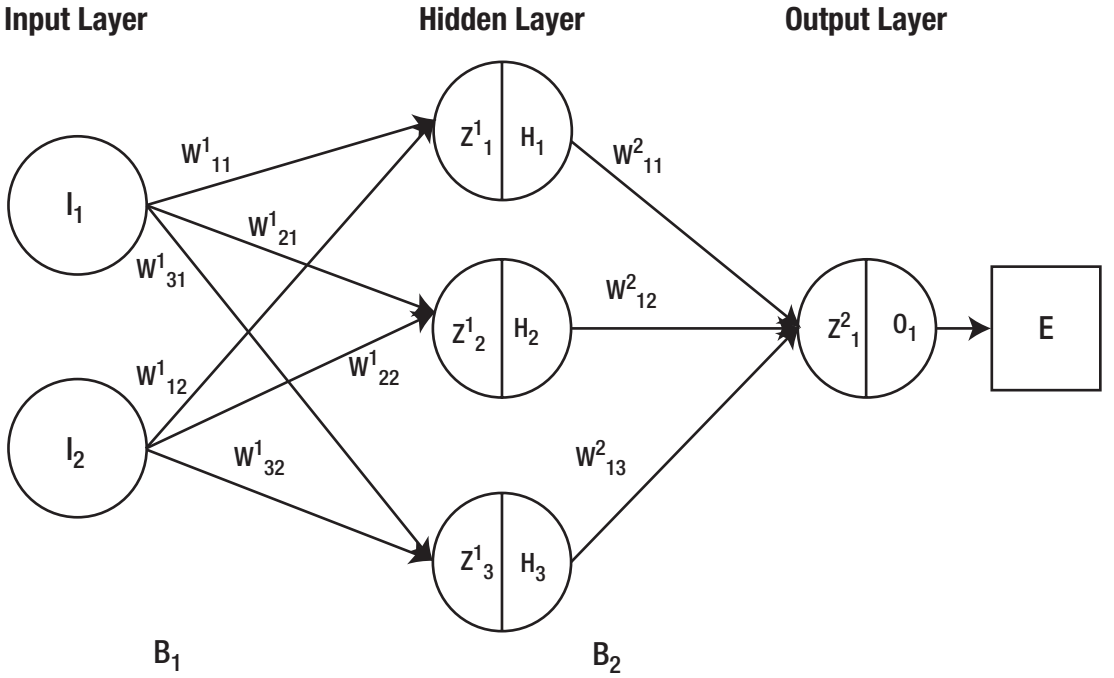


Figure 3-2. *Network diagram*

The network has three layers of neurons (input, hidden, and output). There are two neurons (I_1 and I_2) in the input layer, three neurons (H_1, H_2, H_3) in the hidden layer, and one neuron (O_1) in the output layer. The weights are depicted near the arrows that show the links (connections) between neurons (for example, neurons I_1 and I_2 provide the input for neuron H_1 with the corresponding weights W_{11}^1 and W_{12}^1).

The bodies of neurons in the hidden and output layers (H_1, H_2, H_3, and O_1) are shown as circles divided into two parts (see Figure 3-3). The left part of the neuron body shows the value of the calculated network input for the neuron $\left(Z_1^1 = W_{11}^{1*} I_1 + W_{12}^{1*} I_2 + B_1^* 1\right)$. The initial values for biases are typically set to 1.00. The neuron's output is calculated by applying the sigmoid activation function to the network input into the neuron.

$$H_1 = \sigma\left(Z_1^1\right) = 1/\left(1+\exp\left(-Z_1^1\right)\right)$$

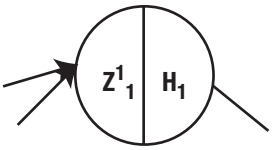


Figure 3-3. *Neuron presentation in the hidden and output layers*

The error function calculation is shown as a quadrant because it is not a neuron (Figure 3-4).

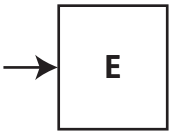


Figure 3-4. *Error function representation*

B1 and B2 are biases for the corresponding network layers.
Here is a summary of the initial network settings:

- Input to neuron $I_1 = 0.01$.
- Input to neuron $I_2 = 0.02$.
- T_1 - (The target output from neuron O_1) = 0.80

You also need to assign initial values to the weights and biases parameters. The values of the initial parameters are typically set randomly, but for this example (where all calculations are done manually) you are assigning them the following values:

$$W_{11}^1 = 0.05 \;\; W_{12}^1 = 0.06 \;\; W_{21}^1 = 0.07 \;\; W_{22}^1 = 0.08 \;\; W_{31}^1 = 0.09 \;\; W_{32}^1 = 0.10$$
$$W_{11}^2 = 0.11 \;\; W_{12}^2 = 0.12 \;\; W_{13}^2 = 0.13$$
$$B_1 = 0.20$$
$$B_2 = 0.25$$

The error limit for this example is set to 0.01.

Forward-Pass Calculation

The forward-pass calculation starts from the hidden layers.

Hidden Layers

For neuron H_1, here are the steps:

1. Calculate the total net input for neuron H_1 (Equation 3-1).

$$Z_1^1 = W_{11}^1 * I_1 + W_{12}^1 * I_2 + B_1 * 1.00 = 0.05 * 0.01 + 0.06 * 0.02 + 0.20 * 1.00 = \mathbf{0}.2017000000000000. \quad (3\text{-}1)$$

2. Use the logistic function to get the output of H_1 (Equation 3-2).

$$H_1 = \delta\left(Z_1^1\right) = 1/\left(1+\exp\left(-Z_1^1\right)\right) = 1/\left(1+\exp\left(-0.2017000000000000\right)\right) = 0.5502547397403884 \quad (3\text{-}2)$$

See Equation 3-3 for neuron H_2.

$$Z_2^1 = W_{21}^1 * I_1 + W_{22}^1 * I_2 + B_1 * 1.00 = 0.07 * 0.01 + 0.08 * 0.02 + 0.20 * 1.00 = \mathbf{0}.2023$$
$$H2 = 1/\left(1+\exp\left(-0.2023\right)\right) = 0.5504032199355139 \quad (3\text{-}3)$$

See Equation 3-4 for neuron H_3.

$$Z_2^1 = W_{31}^1 * I_1 + W_{32}^1 * I_2 + B_1 * 1.00 = 0.09 * 0.01 + 0.10 * 0.02 + 0.20 * 1.00 = \mathbf{0}.20290000000000002$$
$$H3 = 1/\left(1+\exp\left(-0.20290000000000002\right)\right) = 0.5505516911502556 \quad (3\text{-}4)$$

Output Layer

The output layer's neuron O_1 calculation is similar to the hidden-layer neurons calculation but with one difference: the input for output neuron O_1 is the output from the corresponding hidden-layer neurons. Also, notice that there are three hidden-layer neurons contributing to the output layer's neuron O_1.

Here are the steps for neuron O_1:

1. Calculate the total net input for neuron O (Equation 3-5).

$$\begin{aligned} Z_1^2 &= W_{11}^{2*}H_1 + W_{12}^{2*}H_2 + W_{13}^{2*}H_3 + B_2{}^{*}1.00 = 0.11^{*}0.5502547397403884 \\ &\quad + 0.12^{*}0.5504032199355139 + 0.13^{*}0.5505516911502556 + 0.25^{*}1.00 \\ &= 0.44814812761323763 \end{aligned} \tag{3-5}$$

2. Use the logistic function σ to get the output from O_1 (Equation 3-6).

$$\begin{aligned} O_1 &= \sigma\left(Z_1^2\right) = 1/\left(1+\exp\left(-Z_1^2\right)\right) = 1/\left(1+\exp\left(-0.44814812761323763\right)\right) \\ &= 0.6101988445912522 \end{aligned} \tag{3-6}$$

The calculated output from neuron O_1 is 0.6101988445912522, while the target output from neuron O_1 must be = 0.80; therefore, the squared error for the output for neuron O_1 is as shown in Equation 3-7.

$$E = 0.5^{*}\left(T_1 - O_1\right)^2 = 0.5^{*}\left(0.80 - 0.6101988445912522\right) = 0.01801223929724783 \tag{3-7}$$

In this formula, the 0.5 multiplier is used to cancel out the exponent during the derivative calculation. For efficiency reasons, the Encog framework (which you will learn and use later in this book) moves the squaring to later in the calculation.

What is needed here is to minimize the network-calculated error to obtain the good approximation results. This is done by redistributing the network error between the output- and hidden-layer neurons' weights and biases, while taking into consideration that the impact of each neuron on the network output depends on its weight. This calculation is done in the backward-propagation pass.

To redistribute the error toward all the output- and hidden-layer neurons and adjust their weights, you need to understand how much the final error value changes when a weight for each neuron changes. The same is true for the biases for each layer. By redistributing the network error to all output- and hidden-layer neurons, you actually calculate adjustments for each neuron weight and each layer bias.

Backward-Pass Calculation

Calculating weight and bias adjustments for each network neuron/layer is done by moving backward (from the network error to the output layer and then from the output layer to the hidden layers).

Calculating Weight Adjustments for the Output Layer Neurons

Let's calculate the weight adjustment for the neuron W_{11}^2. As you already know, the partial derivative of a function determines the impact of a small change in the error function argument on the corresponding change of the function value. Applying it to the neuron W_{11}^2, here you want to know how a change in W_{11}^2 affects the network error E. To do this, you need to calculate the partial derivative of the error function E with respect to W_{11}^2, which is $\frac{\partial E}{\partial W_{11}^2}$.

Calculating Adjustment for W_{11}^2

Applying the chain rule for derivatives, $\partial E / \partial W_{11}^2$ can be expressed by Equation 3-8.

$$\frac{\partial E}{\partial W_{11}^2} = \frac{\partial E}{\partial O_1} * \frac{\partial O_1}{\partial Z_1^2} * \frac{\partial Z_1^2}{\partial W_{11}^2} \tag{3-8}$$

Let's calculate separately each part of the equation using the derivative calculus (Equation 3-9).

$$\begin{aligned} E &= 0.5 * (T_1 - O_1)^2 \\ \frac{\partial E}{\partial O_1} &= 2 * 0.5 * (T_1 - O_1) * \frac{\partial (0.5(T1 - O1))}{\partial O_1} = (T_1 - O_1) * (-1) = (O_1 - T_1) \\ &= 0.80 - 0.6101988445912522 = -0.18980115540874787 \end{aligned} \tag{3-9}$$

$\frac{\partial O_1}{\partial Z_1^2}$ is the derivative of the sigmoid activation function and is equal to Equation 3-10.

$$\begin{aligned} O_1 * (1 - O_1) &= 0.6101988445912522 * (1 - 0.6101988445912522) \\ &= 0.23785621465075305 \end{aligned} \tag{3-10}$$

See Equation 3-11 and Equation 3-12 to calculate $\frac{\partial Z_1^2}{\partial W_{11}^2}$.

$$Z_1^2 = W_{11}^{2*} H_1 + W_{12}^{2*} H_2 + W_{13}^{2*} H_3 + B_2^{*} 1.00 \tag{3-11}$$

$$\frac{\partial Z_1^2}{\partial W_{11}^2} = H_1 = 0.5502547397403884 \tag{3-12}$$

Note $\frac{\partial\left(W_{12}^{2*} H_2 + W_{13}^{2*} H_3 + B_2^{*} 1.00\right)}{\partial W_{11}^2} = 0$, because this part does not depend on W_{11}^2.

Let's put it all together (Equation 3-13).

$$\frac{\partial E}{\partial W_{11}^2} = -0.18980115540874787^{*} 0.23785621465075305^{*} 0.5502547397403884 = -0.024841461722517316 \tag{3-13}$$

To decrease the error, you need to calculate the new adjusted value for W_{11}^2 by subtracting the value of $\frac{\partial E}{\partial W_{11}^2}$ (optionally multiplied by some learning rate η) from the original value of W_{11}^2 (Equation 3-14).

$$\text{adjusted}W_{11}^2 = W_{11}^2 - \eta^{*} \frac{\partial E}{\partial W_{11}^2}. \text{ For this example, } \eta = 1.$$
$$\text{adjusted}W_{11}^2 = 0.11 + 0.024841461722517316 = 0.13484146172251732 \tag{3-14}$$

Calculating the Adjustment for w_{12}^2

Applying the chain rule for derivatives, $\partial E/\partial W_{12}^2$ can be expressed by Equation 3-15.

$$\frac{\partial E}{\partial W_{11}^2} = \frac{\partial E}{\partial O_1}^{*} \frac{\partial O_1}{\partial Z_1^2}^{*} \frac{\partial Z_1^2}{\partial W_{12}^2} \tag{3-15}$$

Let's calculate separately each part of the equation using derivative calculus (Equation 3-16).

$$\frac{\partial E}{\partial O_1} = -0.18980115540874787 \ (\text{see } 1.13) \tag{3-16}$$

See Equation 3-17 and Equation 3-18 to calculate $\frac{\partial O_1}{\partial Z_1^2}$.

$$\frac{\partial O_1}{\partial Z_1^2} = 0.23785621465075305 \ (\text{see } 1.14) \tag{3-17}$$

See Equation 3-18 and Equation 3-19 to calculate $\frac{\partial Z_1^2}{\partial W_{12}^2}$.

$$Z_1^2 = W_{11}^{2*} H_1 + W_{12}^{2*} H_2 + W_{13}^{2*} H_3 + B_2^* 1.00 \tag{3-18}$$

$$\frac{\partial Z_1^2}{\partial W_{11}^2} = H_2 = 0.5504032199355139 \ (\text{see } 1.3) \tag{3-19}$$

Let's put it all together (Equation 3-20).

$$\begin{aligned} \frac{\partial E}{\partial W_{11}^2} &= -0.18980115540874787^* 0.23785621465075305^* \\ & 0.5502547397403884 = -0.024841461722517316 \end{aligned} \tag{3-20}$$

To decrease the error, you need to calculate the new adjusted value for W_{11}^2 by subtracting the value of $\frac{\partial E}{\partial W_{11}^2}$ (optionally multiplied by some learning rate η) from the original value of W_{12}^2 (Equation 3-21).

$$\begin{aligned} &\text{adjusted}W_{12}^2 = W_{12}^2 - \eta^* \frac{\partial E}{\partial W_{12}^2}. \\ &\text{adjusted}W_{12}^2 = 0.12 + 0.024841461722517316 = 0.1448414617225173. \end{aligned} \tag{3-21}$$

Calculating the Adjustment for W_{13}^2

Applying the chain rule for derivatives, $\partial E / \partial W_{11}^3$ can be expressed by Equation 3-22 and Equation 3-23.

$$\frac{\partial E}{\partial W_{13}^2} = \frac{\partial E}{\partial O_1}^* \frac{\partial O_1}{\partial Z_1^2}^* \frac{\partial Z_1^2}{\partial W_{13}^2}$$

$$\frac{\partial E}{\partial O_1} = -0.18980115540874787 \ (\text{see } 1.13) \tag{3-22}$$

$$\frac{\partial O_1}{\partial Z_1^2} = 0.23785621465075305 \ (\text{see } 1.14) \tag{3-23}$$

See Equation 3-24 and Equation 3-25 to calculate $\frac{\partial Z_1^2}{\partial W_{13}^2}$.

$$Z_1^2 = W_{11}^{2*} H_1 + W_{12}^{2*} H_2 + W_{13}^{2*} H_3 + B_2^* 1.00 \tag{3-24}$$

$$\frac{\partial Z_1^2}{\partial W_{13}^2} = H_3 = 0.5505516911502556 \ (\text{see } 1.4) \tag{3-25}$$

Let's put it all together (Equation 3-26 and Equation 3-27).

$$\frac{\partial E}{\partial W_{13}^2} = -0.18980115540874787^* 0.23785621465075305^* 0.5505516911502556 = -0.024854867708052567 \tag{3-26}$$

$$\text{adjusted}W_{13}^2 = W_{13}^2 - \eta^* \frac{\partial E}{\partial W_{13}^2}. \text{ For this example, } \eta = 1.$$
$$\text{adjusted}W_{12}^2 = 0.13 + 0.024841461722517316 = 0.1548414617225173 \tag{3-27}$$

Therefore, on the second iteration, you will use the following weight-adjusted values:

$$\text{adjusted}W_{11}^{2} = 0.08515853827748268$$

$$\text{adjusted}W_{12}^{2} = 0.09515853827748268$$

$$\text{adjusted}W_{13}^{2} = 0.10515853827748269$$

After adjusting weights for the output neurons, you are ready to calculate weight adjustments for the hidden neurons.

Calculating Weight Adjustments for Hidden-Layer Neurons

Calculating weight adjustments for the neurons in the hidden layer is similar to the corresponding calculations in the output layer but with one important difference. For the neurons in the output layer, the input is now made up of the output results from the corresponding neurons in the hidden layer.

Calculating the Adjustment for W_{11}^{1}

Applying the chain rule for derivatives, $\partial E/\partial W_{11}^{1}$ can be expressed by Equations 3-28, 3-29, 3-30, and 3-31.

$$\frac{\partial E}{\partial W_{11}^{1}} = \frac{\partial E}{\partial H_1}^{*} \frac{\partial H_1}{\partial Z_1^{1}}^{*} \frac{\partial Z_1^{1}}{\partial W_{11}^{1}} \tag{3-28}$$

$$\begin{aligned}\frac{\partial E}{\partial H_1} &= \frac{\partial E}{\partial O_1}^{*} \frac{\partial O_1}{\partial Z_1^{1}} = -0.18980115540874787^{*}0.23785621465075305 \\ &= -0.04514538436186407 \ \ (\text{see } 1.13 \text{ and } 1.14)\end{aligned} \tag{3-29}$$

$$\frac{\partial H_1}{\partial Z_1^1} = \sigma(\mathrm{H}_1) = \mathrm{H}_1^* (1 - \mathrm{H}_1)$$
$$= 0.5502547397403884^* (1 - 0.5502547397403884) \quad (3\text{-}30)$$
$$= 0.24747446113362584$$

$$\frac{\partial Z_1^1}{\partial W_{11}^1} = \frac{\partial \left(W_{11}^{1*} I_1 + W_{12}^{1*} I_2 + B_1^* 1 \right)}{\partial W_{11}^1} = \mathrm{I}_1 = 0.01. \quad (3\text{-}31)$$

Let's put it all together (Equations 3-32, 3-33, and 3-34).

$$\frac{\partial E}{\partial W_{11}^1} = -0.04514538436186407^* 0.24747446113362584^* 0.01$$
$$= -0.0001117232966762273 \quad (3\text{-}32)$$

$$\text{adjusted}\mathrm{W}_{11}^1 = \mathrm{W}_{11}^1 - \eta^* \frac{\partial E}{\partial W_{11}^1} = 0.05 - 0.0001117232966762273$$
$$= 0.049888276703323776 \quad (3\text{-}33)$$

$$\text{adjusted}\mathrm{W}_{11}^1 = \mathrm{W}_{11}^1 - \eta^* \frac{\partial E}{\partial W_{11}^1} = 0.05 + 0.0001117232966762273$$
$$= 0.05011172329667623. \quad (3\text{-}34)$$

Calculating the Adjustment for $\mathbf{W_{12}^1}$

Applying the chain rule for derivatives, $\partial E/\partial \mathrm{W}_{12}^1$ can be expressed by Equations 3-35, 3-36, and 3-37).

$$\frac{\partial E}{\partial W_{12}^1} = \frac{\partial E}{\partial H_1}^* \frac{\partial H_1}{\partial Z_1^1}^* \frac{\partial Z_1^1}{\partial W_{12}^1}$$

$$\frac{\partial E}{\partial H_1} = \frac{\partial E}{\partial O_1}^* \frac{\partial O_1}{\partial Z_1^1} = -0.18980115540874787^* 0.23785621465075305$$
$$= -0.04514538436186407 \quad (\text{see } 1.13 \text{ and } 1.14). \quad (3\text{-}35)$$

$$\frac{\partial H_1}{\partial Z_1^1} = 0.24747446113362584 \quad (\text{see } 1.28) \quad (3\text{-}36)$$

$$\frac{\partial Z_1^1}{\partial W_{12}^1} = \frac{\partial\left(W_{11}^{1*} I_1 + W_{12}^{1*} I_2 + B_1^{*} 1\right)}{\partial W_{12}^1} = \mathrm{I}_2 = 0.02 \tag{3-37}$$

Let's put it all together (Equations 3-38 and 3-39).

$$\begin{aligned}\frac{\partial E}{\partial W_{12}^1} &= -0.04514538436186407^{*}\,0.24747446113362584^{*}\,0.02 \\ &= -0002344659335245464\end{aligned} \tag{3-38}$$

$$\begin{aligned}\text{adjusted}\mathrm{W}_{12}^1 &= \mathrm{W}_{12}^1 - \eta^{*}\frac{\partial E}{\partial W_{12}^1} = 0.06 + 0.0002344659335245464 \\ &= 0.06022344659335245\end{aligned} \tag{3-39}$$

Calculating the Adjustment for $\mathbf{W_{21}^1}$

Applying the chain rule for derivatives, $\partial E/\partial \mathrm{W}_{21}^1$ can be expressed by Equations 3-40, 3-41, 3-42, and 3-43.

$$\frac{\partial E}{\partial W_{21}^1} = \frac{\partial E}{\partial H_2}^{*}\frac{\partial H_2}{\partial Z_2^1}^{*}\frac{\partial Z_2^1}{\partial W_{21}^1} \tag{3-40}$$

$$\begin{aligned}\frac{\partial E}{\partial H_2} &= \frac{\partial E}{\partial O_1}^{*}\frac{\partial O_1}{\partial Z_2^1} = -0.18980115540874787^{*}\,0.23785621465075305 \\ &= -0.04514538436186407 \ \ (\text{see 3.9 and 3.10}).\end{aligned} \tag{3-41}$$

$$\begin{aligned}\frac{\partial H_2}{\partial Z_2^1} &= \mathrm{H}_2^{*}\left(1-\mathrm{H}_2\right) = 0.5504032199355139^{*}\left(1-0.5504032199355139\right) \\ &= 0.059776553406647545\end{aligned} \tag{3-42}$$

$$\frac{\partial Z_2^1}{\partial W_{21}^1} = \frac{\partial\left(W_{21}^{1*} I_1 + W_{22}^{1*} I_2 + B_1^{*} 1\right)}{\partial W_{21}^1} = \mathrm{I}_1 = 0.01 \tag{3-43}$$

Let's put it all together (Equations 3-44 and 3-45).

$$\frac{\partial E}{\partial W_{21}^1} = -0.04514538436186407^* 0.059776553406647545^* 0.01 = -0.00002698635479370598 3 \quad (3\text{-}44)$$

$$\text{adjusted}W_{21}^1 = W_{12}^1 - \eta^* \frac{\partial E}{\partial W_{21}^1} = 0.07 + 0.00002698635479370598 3 = 0.07002698635479371 \quad (3\text{-}45)$$

Calculating the Adjustment for W_{22}^1

Applying the chain rule for derivatives, $\partial E / \partial W_{22}^1$ can be expressed by Equations 3-46, 3-47, 3-48, and 3-49.

$$\frac{\partial E}{\partial W_{22}^1} = \frac{\partial E}{\partial H_2}^* \frac{\partial H_2}{\partial Z_2^1}^* \frac{\partial Z_2^1}{\partial W_{22}^1} \quad (3\text{-}46)$$

$$\frac{\partial E}{\partial H_2} = \frac{\partial E}{\partial O_1}^* \frac{\partial O_1}{\partial Z_2^1} = -0.04514538436186407 \ \ (\text{see } 3.9 \text{ and } 3.10). \quad (3\text{-}47)$$

$$\frac{\partial H_2}{\partial Z_2^1} = H_2^* (1 - H_2) = 0.5504032199355139^* (1 - 0.5504032199355139) = 0.059776553406647545 \quad (3\text{-}48)$$

$$\frac{\partial Z_2^1}{\partial W_{22}^1} = \frac{\partial \left(W_{21}^{1*} I_1 + W_{22}^{1*} I_2 + B_1^* 1 \right)}{\partial W_{22}^1} = I_2 = 0.02 \quad (3\text{-}49)$$

Let's put it all together (Equations 3-50 and 3-51).

$$\frac{\partial E}{\partial W_{22}^1} = -0.04514538436186407^* 0.059776553406647545^* 0.02 = -0.0000539727095874119 66 \quad (3\text{-}50)$$

$$\text{adjusted}W_{22}^1 = W_{22}^1 - \eta^* \frac{\partial E}{\partial W_{22}^1} = 0.08 + 0.0000539727095874119 66 = 0.08005397270958742 \quad (3\text{-}51)$$

Calculating the Adjustment for W_{31}^1

Applying the chain rule for derivatives, $\partial E/\partial W_{31}^1$ can be expressed by Equations 3-52, 3-53, 3-54, and 3-55.

$$\frac{\partial E}{\partial W_{31}^1} = \frac{\partial E}{\partial H_3}^* \frac{\partial H_3}{\partial Z_3^1}^* \frac{\partial Z_3^1}{\partial W_{31}^1} \tag{3-52}$$

$$\frac{\partial E}{\partial H_3} = \frac{\partial E}{\partial O_1}^* \frac{\partial O_1}{\partial Z_3^1} = -0.04514538436186407 \quad (\text{see } 1.13 \text{ and } 1.14). \tag{3-53}$$

$$\begin{aligned} \frac{\partial H_3}{\partial Z_3^1} &= H_3^* (1 - H_3) = 0.5505516911502556^* (1 - 0.5505516911502556) \\ &= 0.24744452652184917 \end{aligned} \tag{3-54}$$

$$\frac{\partial Z_3^1}{\partial W_{31}^1} = \frac{\partial \left(W_{31}^{1*} I_1 + W_{32}^{1*} I_2 + B_1^* 1 \right)}{\partial W_{31}^1} = I_1 = 0.01 \tag{3-55}$$

Let's put it all together (Equations 3-56 and 3-57).

$$\begin{aligned} \frac{\partial E}{\partial W_{22}^1} &= -0.04514538436186407^* 0.24744452652184917^* 0.01 \\ &= -0.0001117097825806835 \end{aligned} \tag{3-56}$$

$$\begin{aligned} \text{adjusted} W_{31}^1 &= W_{31}^1 - \eta^* \frac{\partial E}{\partial W_{31}^1} = 0.09 + 0.0001117097825806835 \\ &= 0.09011170978258068 \end{aligned} \tag{3-57}$$

Calculating the Adjustment for W_{32}^1

Applying the chain rule for derivatives, $\partial E/\partial W_{32}^1$ can be expressed by Equations 3-58, 3-59, 3-60, and 3-61.

$$\frac{\partial E}{\partial W_{32}^1} = \frac{\partial E}{\partial H_3} * \frac{\partial H_3}{\partial Z_3^1} * \frac{\partial Z_3^1}{\partial W_{32}^1} \tag{3-58}$$

$$\frac{\partial E}{\partial H_3} = \frac{\partial E}{\partial O_1} * \frac{\partial O_1}{\partial Z_3^1} = -0.04514538436186407 \quad (\text{see } 1.49). \tag{3-59}$$

$$\begin{aligned} \frac{\partial H_3}{\partial Z_3^1} &= H_3 * (1 - H_3) = 0.5505516911502556 * (1 - 0.5505516911502556) \\ &= 0.24744452652184917 \quad (\text{see } 1.50) \end{aligned} \tag{3-60}$$

$$\frac{\partial Z_3^1}{\partial W_{32}^1} = \frac{\partial \left(W_{31}^1 * I_1 + W_{32}^1 * I_2 + B_1 * 1 \right)}{\partial W_{32}^1} = I_2 = 0.02 \tag{3-61}$$

Let's put it all together (Equations 3-62 and 3-63).

$$\begin{aligned} \frac{\partial E}{\partial W_{32}^1} &= -0.04514538436186407 * 0.24744452652184917 * 0.02 \\ &= -0.000223419565161367 \end{aligned} \tag{3-62}$$

$$\begin{aligned} \text{adjusted} W_{32}^1 &= W_{32}^1 - \eta * \frac{\partial E}{\partial W_{31}^1} = 0.10 + 0.000223419565161367 \\ &= 0.10022341956516137 \end{aligned} \tag{3-63}$$

Updating Network Biases

You need to calculate the error adjustment for the biases B_1 and B_2. Again, using the chain rule, see Equations 3-64 and 3-65.

$$\frac{\partial E}{\partial B_1} = \frac{\partial E}{\partial O_1}^{*} \frac{\partial O_1}{\partial Z_1^1}^{*} \frac{\partial Z_1^1}{\partial B_1} \tag{3-64}$$

$$\frac{\partial E}{\partial B_2} = \frac{\partial E}{\partial O_1}^{*} \frac{\partial O_1}{\partial Z_1^2}^{*} \frac{\partial Z_1^2}{\partial B_2} \tag{3-65}$$

Calculate three parts of the previous formula for both expressions (Equations 3-66, 3-67, 3-68, and 3-69).

$$\frac{\partial Z_1^1}{\partial B_1} = \frac{\partial\left(W_{11}^{1*} I_1 + W_{12}^{1*} I_2 + B_1{}^{*} 1\right)}{\partial B_1} = 1 \tag{3-66}$$

$$\frac{\partial Z_1^2}{\partial B_2} = \frac{\partial\left(W_{11}^{2*} H_1 + W_{12}^{2*} H_2 + W_{13}^{2*} H_3 + B_2{}^{*} 1\right)}{\partial B_2} = 1 \tag{3-67}$$

$$\frac{\partial E}{\partial B_1} = \frac{\partial E}{\partial H_1}^{*} \frac{\partial H_1}{\partial Z_1^1}^{*} 1 = \delta_1^1 \tag{3-68}$$

$$\frac{\partial E}{\partial B_2} = \frac{\partial E}{\partial H_2}^{*} \frac{\partial H_2}{\partial Z_1^2}^{*} 1 = \delta_1^2 \tag{3-69}$$

Because you are using biases B1 and B2 per layer and not per neuron, you can calculate the average δ for the layer (Equations 3-70 through 3-76).

$$\delta^1 = \delta_1^1 + \delta_2^1 + \delta_3^1 \tag{3-70}$$

$$\frac{\partial E}{\partial B_1} = \delta^1 \tag{3-71}$$

$$\frac{\partial E}{\partial B_2} = \delta^2 \tag{3-72}$$

$$\begin{aligned} \delta^2 &= \frac{\partial E}{\partial O_1}^{*} \frac{\partial O_1}{\partial Z_1^2} = -0.18980115540874787^{*} 0.23785621465075305 \\ &= -0.04514538436186407 \end{aligned} \tag{3-73}$$

$$\delta_1^1 = \frac{\partial E}{\partial O_1}^* \frac{\partial O_1}{\partial Z_2^1} = -0.04514538436186407 \quad (3\text{-}74)$$

$$\delta_2^1 = \frac{\partial E}{\partial O_1}^* \frac{\partial O_1}{\partial Z_1^1} = -0.04514538436186407 \quad (3\text{-}75)$$

$$\delta_3^1 = \frac{\partial E}{\partial O_1}^* \frac{\partial O_1}{\partial Z_3^1} = -0.04514538436186407 \quad (3\text{-}76)$$

Because for bias adjustments you calculate per layer, you can take the average of the calculated bias adjustments for each neuron (Equation 3-77).

$$\begin{aligned} \delta^1 &= \left(\delta_1^1 + \delta_2^1 + \delta_3^1\right)/3 = -0.04514538436186407 \\ \delta^2 &= -0.04514538436186407 \end{aligned} \quad (3\text{-}77)$$

With the introduction of variable δ, you get Equations 3-78 and 3-79.

$$\text{adjustedB}_1 = \text{B}_1 - \eta^* \delta_1 = 0.20 + 0.04514538436186407 = 0.2451453843618641 \quad (3\text{-}78)$$

$$\begin{aligned} \text{adjustedB}_2 &= \text{B}_2 - \eta^* \delta_2 = 0.25 + 0.04514538436186407 \\ &= 0.29514538436186405 \end{aligned} \quad (3\text{-}79)$$

Now that you have calculated all the new weight values, you go back to the forward phase and calculate a new error.

Going Back to the Forward Pass

Recalculate the network output for the hidden and output layers using the new adjusted weight/biases.

Hidden Layers

For neuron H_1, here are the steps:

1. Calculate the total net input for neuron H_1 (Equation 3-80).

$$\begin{aligned} Z_1^1 &= W_{11}^1 * I_1 + W_{12}^1 * I_2 + B_1 * 1.00 = 0.05011172329667623 * 0.01 \\ &+ 0.06022344659335245 * 0.02 + 0.2451453843618641 * 1.00 \\ &= 0.2468509705266979 \end{aligned} \tag{3-80}$$

2. Use the logistic function to get the output of H_1 (Equation 3-81).

$$\begin{aligned} H_1 &= \delta\left(Z_1^1\right) = 1/\left(1+\exp\left(-Z_1^1\right)\right) = 1/\left(1+\exp\left(-0.2468509705266979\right)\right) \\ &= 0.561401266257945 \end{aligned} \tag{3-81}$$

For neuron H_2, see Equation 3-82 and Equation 3-83.

$$\begin{aligned} Z_2^1 &= W_{21}^1 * I_1 + W_{22}^1 * I_2 + B_1 * 1.00 = 0.07002698635479371 * 0.01 \\ &+ 0.08005397270958742 * 0.02 + 0.2451453843618641 * 1.00 \\ &= 0.24744673367960376 \end{aligned} \tag{3-82}$$

$$H2 = 1/\left(1+\exp\left(-0.24744673367960376\right)\right) = 0.5615479555799516 \tag{3-83}$$

For neuron H_3, see Equation 3-84.

$$\begin{aligned} Z_2^1 &= W_{31}^1 * I_1 + W_{32}^1 * I_2 + B_1 * 1.00 = 0.09011170978258068 * 0.01 \\ &+ 0.10022341956516137 * 0.02 + 0.2451453843618641 * 1.00 \\ &= 0.24805096985099312 \\ H3 &= 1/\left(1+\exp\left(-0.24805096985099312\right)\right) = 0.5616967201480348 \end{aligned} \tag{3-84}$$

Output Layer

For neuron O_1, here are the steps:

1. Calculate the total net input for neuron O_1 (Equation 3-85).

$$\begin{aligned} Z_1^2 &= W_{11}^2 * H_1 + W_{12}^2 * H_2 + W_{13}^2 * H_3 + B_2 * 1.00 \\ &= 0.13484146172251732 * 0.5502547397403884 \\ &+ 0.1448414617225173 * 0.5504032199355139 \\ &+ 0.1548414617225173 * 0.5505516911502556 \\ &+ 0.29514538436186405 * 1.00 = 0.5343119733119508 \end{aligned} \tag{3-85}$$

2. Use the logistic function σ to get the output from O_1 (Equation 3-86).

$$\begin{aligned} O_1 &= \sigma\left(Z_1^2\right) = 1/\left(1+\exp\left(-Z_1^2\right)\right) = 1/\left(1+\exp\left(-0.5343119733119508\right)\right) \\ &= 0.6304882485312977 \end{aligned} \tag{3-86}$$

The calculated output from neuron O_1 is 0.6304882485312977, while the target output for O_1 is 0.80; therefore, see Equation 3-87 for the squared error for the output for neuron O_1.

$$\begin{aligned} E &= 0.5^*\left(T_1 - O_1\right)^2 = 0.5^*\left(0.80 - 0.6304882485312977\right)^2 \\ &= 0.014367116942993556 \end{aligned} \tag{3-87}$$

On the first iteration, the error was 0.01801223929724783 (see 1.7). Now, on the second iteration, the error has been reduced to 0.014367116942993556.

You can continue these iterations until the network calculates an error that is smaller than the error limit that has been set. Let's look at the formulas for calculating the partial derivative of the error function E with respect to W_{11}^2 and W_{12}^2 for the node H1 (see Equations 3-88, 3-89, and 3-90).

$$\frac{\partial E}{\partial W_{11}^2} = \frac{\partial E}{\partial O_1}^* \frac{\partial O_1}{\partial Z_1^2}^* \frac{\partial Z_1^2}{\partial W_{11}^2} \tag{3-88}$$

$$\frac{\partial E}{\partial W_{12}^2} = \frac{\partial E}{\partial O_1}^* \frac{\partial O_1}{\partial Z_1^2}^* \frac{\partial Z_1^2}{\partial W_{12}^2} \tag{3-89}$$

$$\frac{\partial E}{\partial W_{13}^2} = \frac{\partial E}{\partial O_1}^* \frac{\partial O_1}{\partial Z_1^2}^* \frac{\partial Z_1^2}{\partial W_{13}^2} \tag{3-90}$$

You can see that all three formulas have a common part: $\frac{\partial E}{\partial O_1}^{*}\frac{\partial O_1}{\partial Z_1^2}$. This part is called Node Delta δ. Using δ, you can rewrite Equations 3-88, 3-89, and 3-90 as Equations 3-91, 3-92, and 3-93.

$$\frac{\partial E}{\partial W_{11}^2} = \delta_1^{2*}\frac{\partial Z_1^2}{\partial W_{11}^2} \tag{3-91}$$

$$\frac{\partial E}{\partial W_{12}^2} = \delta_1^{2*}\frac{\partial Z_1^2}{\partial W_{12}^2} \tag{3-92}$$

$$\frac{\partial E}{\partial W_{13}^2} = \delta_1^{2*}\frac{\partial Z_1^2}{\partial W_{13}^2} \tag{3-93}$$

Correspondingly, you can rewrite the formulas for the hidden layer (see Equations 3-94 through 3-99).

$$\frac{\partial E}{\partial W_{11}^1} = \delta_1^{1*}\frac{\partial Z_1^1}{\partial W_{11}^1} \tag{3-94}$$

$$\frac{\partial E}{\partial W_{12}^1} = \delta_1^{1*}\frac{\partial Z_1^1}{\partial W_{12}^1} \tag{3-95}$$

$$\frac{\partial E}{\partial W_{21}^1} = \delta_2^{1*}\frac{\partial Z_2^1}{\partial W_{21}^1} \tag{3-96}$$

$$\frac{\partial E}{\partial W_{22}^1} = \delta_2^{1*}\frac{\partial Z_2^1}{\partial W_{22}^1} \tag{3-97}$$

$$\frac{\partial E}{\partial W_{31}^1} = \delta_3^{1*}\frac{\partial Z_3^1}{\partial W_{31}^1} \tag{3-98}$$

$$\frac{\partial E}{\partial W_{32}^1} = \delta_3^{1*}\frac{\partial Z_3^1}{\partial W_{32}^1} \tag{3-99}$$

In general, calculating the partial derivative of error function E with respect to its weights can be done by multiplying the node's delta by the partial derivative of the error function with respect to the corresponding weight. That saves you from calculating some redundant data. This means you can calculate δ values for each network node and then use Equations 3.94 through 3.99.

Matrix Form of Network Calculation

Let's say that there are two records (two points) to be processed by the network. For the same network, you can put do the calculations using matrices. For example, by introducing the Z vector, W matrix, and B vector, you can get the same calculation results as you get when using scalars. See Figure 3-5.

$$\begin{vmatrix} Z^1_1 \\ Z^1_2 \\ Z^1_3 \end{vmatrix} = \begin{Vmatrix} W^1_{11} & W^1_{12} \\ W^1_{21} & W^1_{22} \\ W^1_{21} & W^1_{22} \end{Vmatrix} \begin{vmatrix} I_1 \\ I_2 \end{vmatrix} + \begin{vmatrix} B1 \\ B1 \\ B1 \end{vmatrix} = \begin{Vmatrix} W^1 11*I_1 + W^1 12*I_2 + B_1 \\ W^1 11*I_1 + W^1 12*I_2 + B_1 \\ W^1 11*I_1 + W^1 12*I_2 + B_1 \end{Vmatrix}$$

Figure 3-5. *Matrix form of network calculation*

Using matrix versus scalar calculations is a matter of preference. Using a good matrix library provides a fast calculation. For example, the cuBLAS library can take advantage of GPUs and FPGA. The downsizing nature of using matrixes causes a high memory demand since matrixes should be kept in memory.

Digging Deeper

When using a neural network, you set the error limit (specifically indicating how close the trained network results should match the target data). The training process works iteratively by gradually moving in the direction toward the error function minimum, therefore reducing the error. The iterative process stops when the difference between the calculated network results and the target results is less than the preset error limit.

Could the network fail to reach the error limit that was set? Unfortunately, yes. Let's discuss this in more detail. Of course, the approximation error depends on the network architecture being selected (the number of hidden layers and the number of neurons within each hidden layer). However, let's presume that the network architecture is being set correctly.

The approximation error also depends on the function topology. Again, let's presume that the function is monotone and continuous (we will discuss noncontinuous function approximation later in this book). Still, the network can fail to reach the network limit. Why? I've already mentioned that backpropagation is an iterative process that looks for the minimum of error function. Error function is typically a function of many variables, but for simplicity Figure 3-6 shows it as the 2-D space chart.

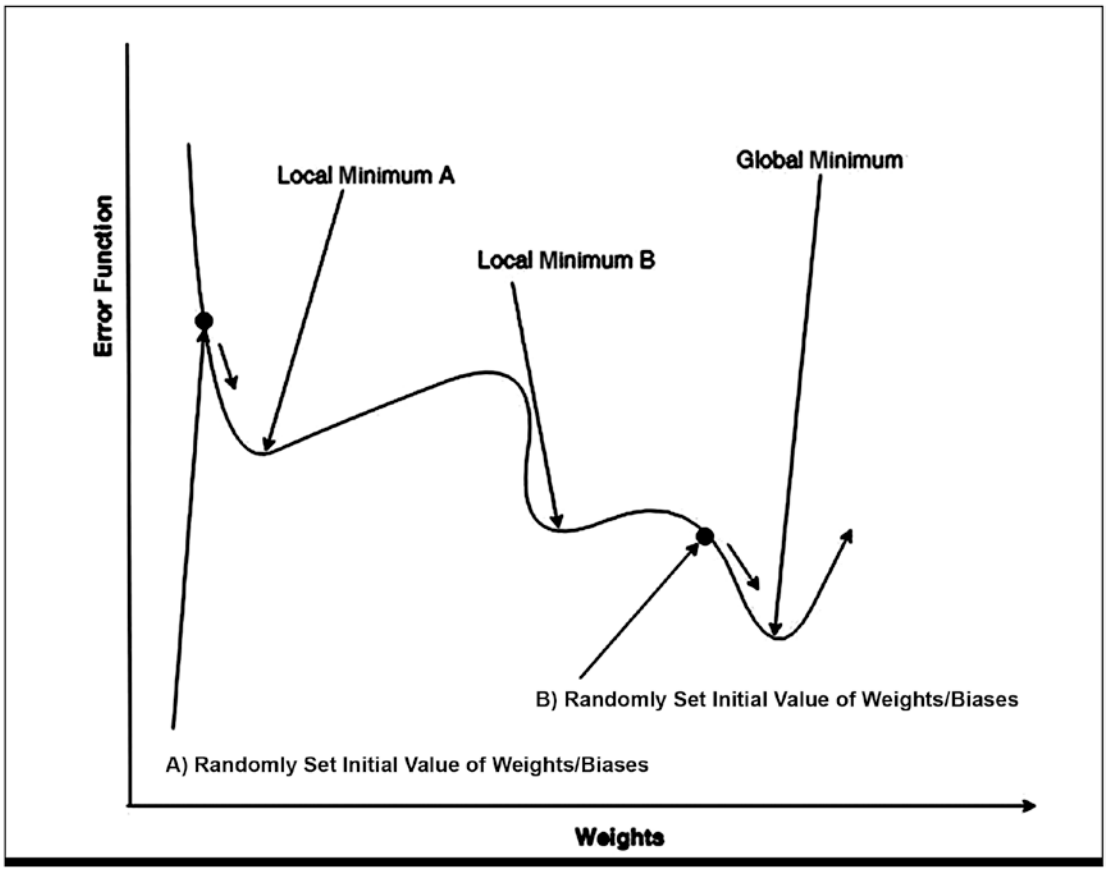


Figure 3-6. *Error function local and global minimums*

The goal of the training process is to find the minimum of the error function. The error function depends on the weights/biases parameters being calibrated during the iterative training process. The initial values of the weights/biases are typically set randomly, and the training process calculates the network error for this initial setting (point). Starting from this point, the training process moves down to the function minimum.

As shown in Figure 3-6, the error function typically has several minimums. The lowest of them is called the *global minimum*, while the rest of them are called the *local minimums*. Depending on the starting point of the training process, it can find some of the local minimums that are close to the starting point. Each local minimum works as a trap for the training process because once the training process reaches a local minim, any further move would show a changing gradient value, and the iterative process would stop, being stacked at the local minimum.

Consider starting points A and B in Figure 3-6. In case of starting point A, the training process will find the local minimum A, which produces a much larger error than in case B. That's why running the same training process multiple times always produces different error results for each run (because for each run, the training process starts at a random initial point).

Tip How do you to achieve the best approximation results? When programming neural network processing, always arrange the logic to start the training process in a loop. After each call of the training method, the logic inside the training method should check whether the error is less than the error limit, and if it is not, it should exit from the training method with a nonzero error code. The control will be returned to the code that calls the training method in a loop. The code calls the training method again if the return code is not zero. The logic continues the loop until the calculated error becomes less than the error limit. It will exit at this point with the zero return code so the training method will no longer be called again. If this is not done, the training logic will just loop over the epochs, not being able to clear the error limit. Some examples of such programming code are shown in later chapters.

Why is it sometimes hard to train the network? The network is considered to be a universal function approximation tool. However, there is an exception to this statement. The network can approximate only continuous functions well. If a function is noncontinuous (making sudden sharp up and down jumps), then the approximation results for such functions show such low-quality results (large errors) that such approximation is useless. I will discuss this issue later in this book and show my method that allows for the approximation of noncontinuous functions with high precision.

The calculation shown here is done for a single function point. When you need to approximate a function of two or more variables at many points, the volume of calculation increases exponentially. Such a resource-intensive process puts high demand on computer resources (memory and CPU). That's why, as mentioned in the introduction, earlier attempts to use artificial intelligence were unable to process serious tasks. Only later, because of dramatically increased computation power, did artificial intelligence achieve a huge success.

Mini-Batches and Stochastic Gradient

When the input data set is very large (millions of records), the volume of calculations is extremely high. Processing such networks takes a long time, and the network learning becomes very slow, because the gradient should be calculated for each input record.

To speed up this process, you can break a large input data set into a number of chunks called *mini-batches* and process each mini-batch independently. Processing all records in a single mini-batch file constitutes an epoch, the point where weight/biases adjustments are made.

Because of the much smaller size of the mini-batch file, processing all mini-batches will be faster than processing of the entire data set as a single file. Finally, instead of calculating the gradient for each record of the entire data set, you calculate here the stochastic gradient, which is the average of gradients calculated for each mini-batch.

If the weight adjustment processed for the neurons in a mini-batch file m is W_n^m, then the weight adjustment for such a neuron for the whole data set is approximately equal to the average of adjustments calculated independently for all mini-batches.

$$adjusted W_s^k \approx W_s^k - \frac{\eta}{m}\sum_j^m \frac{\partial E}{\partial W_s^j}$$, where m is the number of mini-batches

Neural network processing for large input data sets is mostly done using mini-batches.

Summary

This chapter showed all the internal neural network calculations. It explained why (even for a single point) the volume of calculations is quite high. The chapter introduced the δ variable, which allows you to reduce the calculation volume. In the "Digging Deeper" section, it explained how to call the training method to achieve one of the best approximation results. The mini-batch approach was also explained here. The next chapter explains how to configure the Windows environment to use Java and the Java network processing framework.

CHAPTER 4

Configuring Your Development Environment

This book is about neural network processing using Java. Before you can start developing any neural network program, you need to learn several Java tools. If you are a Java developer and are familiar with the tools discussed in this chapter, you can skip this chapter. Just make sure that all the necessary tools are installed on your Windows machine.

Installing the Java 11 Environment on Your Windows Machine

All the examples in this book work well with Java versions 8–11. Here are the steps:

1. Go to https://docs.oracle.com/en/java/javase/11/install/installation-jdk-microsoft-windows-platforms.html#GUID-A740535E-9F97-448C-A141-B95BF1688E6F.
2. Download the latest Java SE Development Kit for Windows. Double-click the downloaded executable file.
3. Follow the installation instructions, and the Java environment will be installed on your machine.
4. On your desktop, select Start ➤ Control Panel ➤ System and security ➤ System ➤ Advanced system setting. The screen shown in Figure 4-1 will appear.

I. Livshin, *Artificial Neural Networks with Java*, https://doi.org/10.1007/978-1-4842-4421-0_4

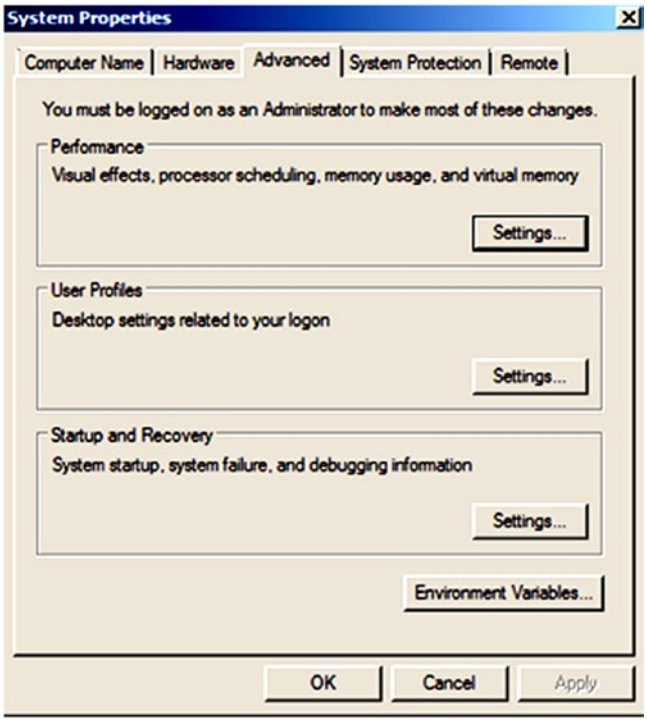


Figure 4-1. *System Properties dialog*

5. Click the Environment Variables button on the Advanced tab.
6. Click New to see the dialog that allows you to enter a new environment variable (Figure 4-2).

Figure 4-2. *New System Variable dialog*

7. Enter **JAVA_HOME** in the "Variable name" field.
8. Enter the path to the installed Java environment in the "Variable value" field (Figure 4-3).

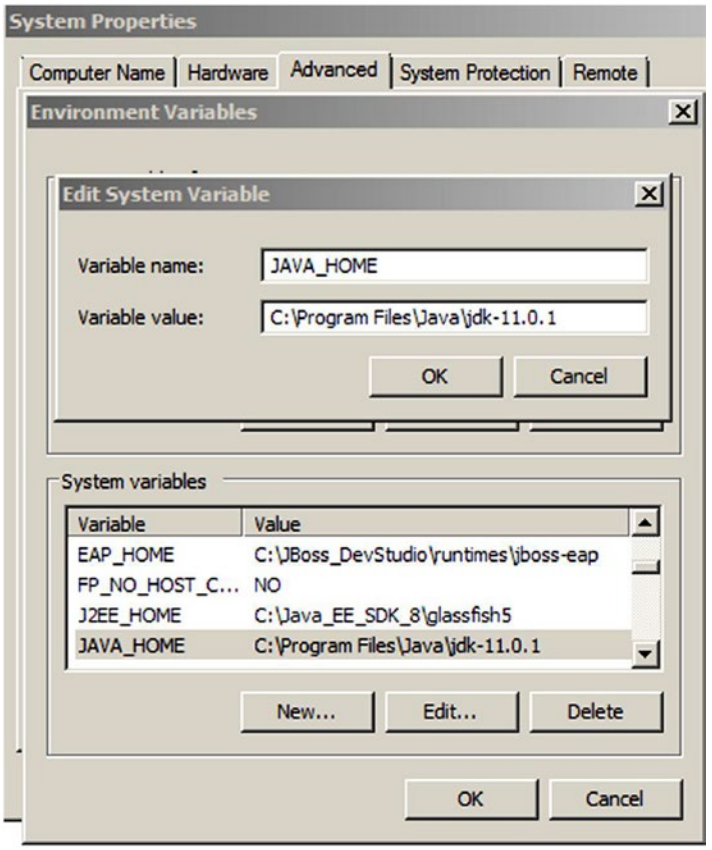


Figure 4-3. *New System Variable dialog, filled in*

9. Click OK. Next, select the CLASSPATH environment variable and click Update.
10. Add the path to the Java JDK `bin` directory, and add the Java JAR file to the CLASSPATH's "Variable value" field (Figure 4-4), as shown here:

```
C:\Program Files\Java\jre1.8.0_144\bin
C:\Program Files\Java\jdk1.8.0_144\jre\bin\java.exe
```

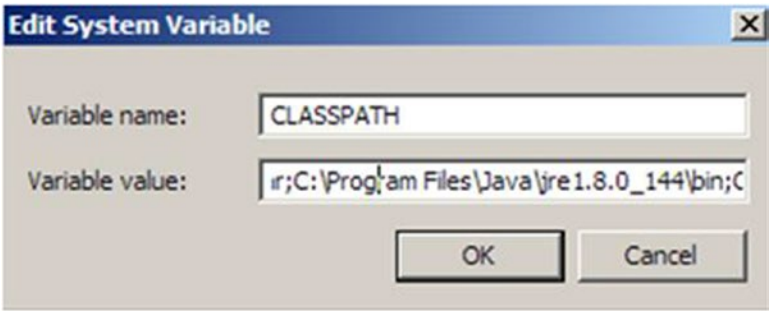


Figure 4-4. *Updated CLASSPATH system variable*

11. Click OK three times.
12. Reboot the system. Your Java environment is set.

Installing the NetBeans IDE

NetBeans is the standard Java development tool currently maintained by Oracle. At the time of writing this book, the current version of NetBeans is 8.2, and it is the official IDE for the Java 8 platform. To install NetBeans, go to `https://netbeans.org/features/index.html` and click Download (Figure 4-5).

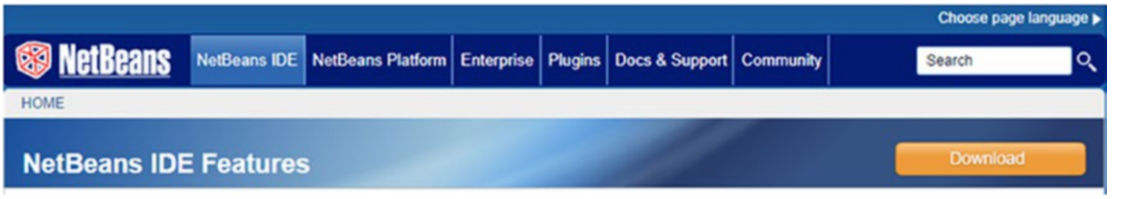


Figure 4-5. *NetBeans home page*

On the Download screen, click the Download button for Java SE (Figure 4-6).

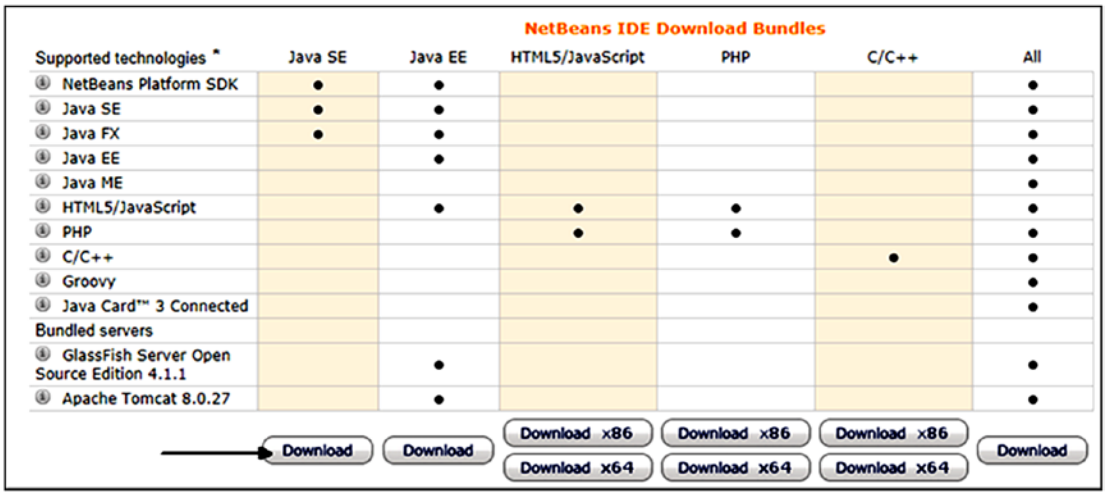


Figure 4-6. *NetBeans home page*

Double-click the downloaded executable file and follow the installation instructions. NetBeans 8.2 will be installed on your Windows machine, and its icon will be placed on your desktop.

Installing the Encog Java Framework

As you can see from some of the examples of manually processing neural networks in Chapter 3, even a simple approximation of a function at a single point involves a large volume of calculations. For any serious work, there are two choices: automate this process yourself or use one of the available frameworks.

Several frameworks are currently available. Here is the list of the most commonly used frameworks and the language they are written for:

- TensorFlow (Python, C++, and R)
- Caffe (C, C++, Python, and MATLAB)
- Torch (C++, Python)
- Keras (Python)
- Deeplearning4j (Java)
- Encog (Java)
- Neurop (Java)

Frameworks implemented in Java are much more efficient compared to those implemented in Python. Here we are interested in a Java framework (for the obvious reason of developing an application on one machine and being able to run it anywhere). We are also interested in a fast Java framework and one that is convenient to use. After examining several Java frameworks, I selected Encog as the best framework for neural network processing. That is the framework used throughout this book. The Encog machine learning framework was developed by Heaton Research, and it is free to use. All Encog documentation can also be found on the web site.

To install Encog, go to the following web page: https://www.heatonresearch.com/encog. Scroll down to the section called Encog Java Links and click the Encog Java Download/Release link. On the next screen, select these two files for Encog release 4.3:

```
encog-core-3.4.jar
encog-java-examples.zip
```

Unzip the second file. Keep these files in a directory that you will remember, and add the following files to the CLASSPATH environment variable (like you did for the Java installation):

```
c:\encog-core-3.4\lib\encog-core-3.4.jar
```

Installing the XChart Package

During data preparation and neural network development/testing, it is useful to be able to chart many results. You will be using the XChart Java charting library in this book. To download XChart, go to the following web site: https://knowm.org/open-source/xchart/.

Click the Download button. The screen shown in Figure 4-7 will appear.

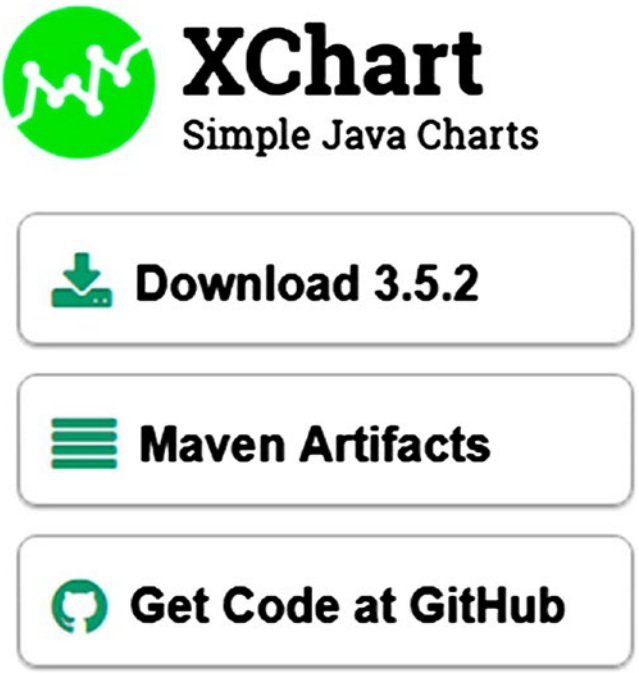


Figure 4-7. *XChart home page*

Unzip the downloaded zip file and double-click the executable installation file. Follow the installation instructions, and the XChart package will be installed on your machine. Add the following two files to the CLASSPATH environment variable (like you did for Java 8 earlier):

```
c:\Download\XChart\xchart-3.5.0\xchart-3.5.0.jar
c:\Download\XChart\xchart-3.5.0\xchart-demo-3.5.0.jar
```

Finally, reboot the system. You are ready for neural network development!

Summary

This chapter introduced you to the Java environment and explained how to download and install a set of tools necessary for building, debugging, testing, and executing neural network applications. All the development examples in the rest of this book will be created using this environment. The next chapter shows how to develop a neural network program using the Java Encog framework.

CHAPTER 5

Neural Network Development Using the Java Encog Framework

To facilitate your learning of network program development using Java, you will develop your first simple program using the function from Example 1 in Chapter 2.

Example 2: Function Approximation Using the Java Environment

Figure 5-1 shows the function that is given to you with its values at nine points.

I. Livshin, *Artificial Neural Networks with Java*, https://doi.org/10.1007/978-1-4842-4421-0_5

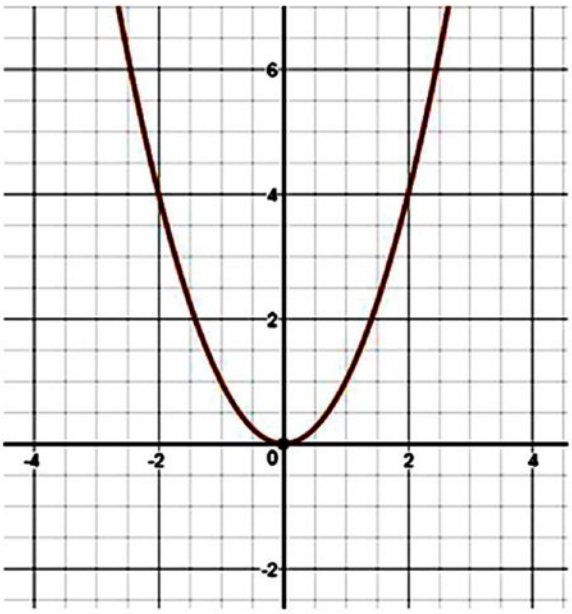


Figure 5-1. *Function to be approximated*

Although Encog can process a set of file formats that belong to the BasicMLDataset format, the easiest file format that Encog can process is the CSV format. The CSV format is a simplified Excel file format that includes comma-separated values in each record, and the files have the extension `.csv`. Encog expects the first record in the processed files to be a label record; accordingly, Table 5-1 shows the input (training) data set with the given function values for this example.

Table 5-1. *Training Data Set*

xPoint	Function Value
0.15	0.0225
0.25	0.0625
0.5	0.25
0.75	0.5625
1	1
1.25	1.5625
1.5	2.25
1.75	3.0625
2	4

Next is the data set for testing the trained network, shown in Table 5-2. The xPoints of this data set are different from the xPoints in the training data set because the file is used to test the network at the xPoints not used for network training.

Table 5-2. *Testing Data Set*

xPoint	Function Value
0.2	0.04
0.3	0.09
0.4	0.16
0.7	0.49
0.95	0.9025
1.3	1.69
1.6	2.56
1.8	3.24
1.95	3.8025

Network Architecture

Figure 5-2 shows the network architecture for the example. As already mentioned, the network architecture for each project is determined experimentally, by trying various configurations and selecting the one that produces the best result. The network is set to have an input layer with a single neuron, seven hidden layers (each having five neurons), and an output layer with a single neuron.

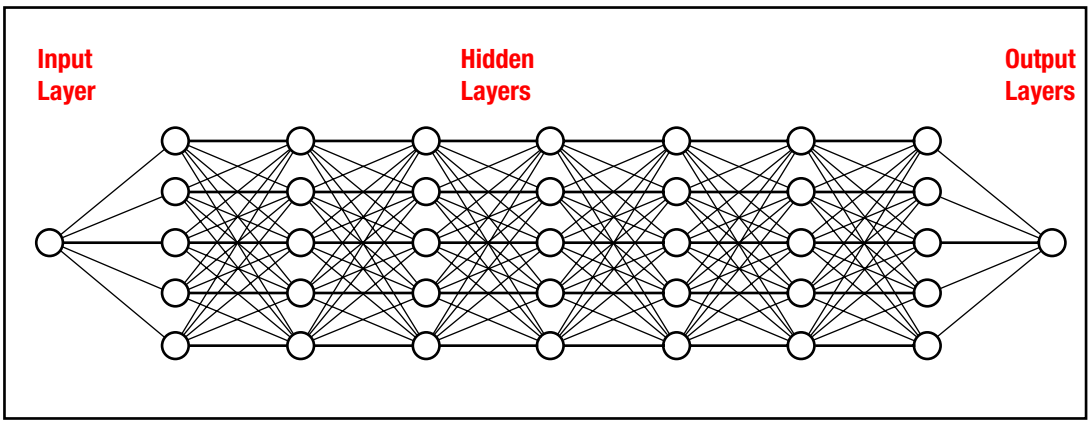


Figure 5-2. *Network architecture*

Normalizing the Input Data Sets

Both the training and testing data sets need to be normalized on the interval [-1, 1]. Let's build the Java program that normalizes those data sets. To normalize a file, it is necessary to know the max and min values for the fields being normalized. The first column of the training data set has the min value 0.15 and the max value 2.00. The second column of the training data set has the min value 0.0225 and the max value 4.00. The first column of the testing data set has the min value 0.20 and the max value 1.95. The second column of the testing data set has the min value 0.04 and the max value 3.8025. Therefore, for simplicity, select the min and max values for both the training and testing data sets as follows: min = 0.00 and max = 5.00.

The formula used to normalize the values on the interval [-1, 1] is shown here:

$$f(x) = \left((x - DL)^{*}(NH - NL)\right) / (DH - DL) + NL$$

where:

x: Input data point

D_L: Minimum (lowest) value of x in the input dataset

D_H: Maximum (highest) value of x in the input dataset

N_L: The left part of the normalized interval [-1, 1] = -1

N_H: The right part of the normalized interval [-1, 1] = 1

When using this formula, make sure that the D_L and D_H values that you use are really the lowest and highest function values on the given interval; otherwise, the result of the optimization will not be good. Chapter 6 discusses function optimization outside of the training range.

Building the Java Program That Normalizes Both Data Sets

Click the NetBeans icon on your desktop to open the NetBeans IDE. The IDE screen is divided into several windows. The Navigation window is where you see your projects (Figure 5-3). Clicking the + icon in front of a project shows the project's components: source package, test package, and libraries.

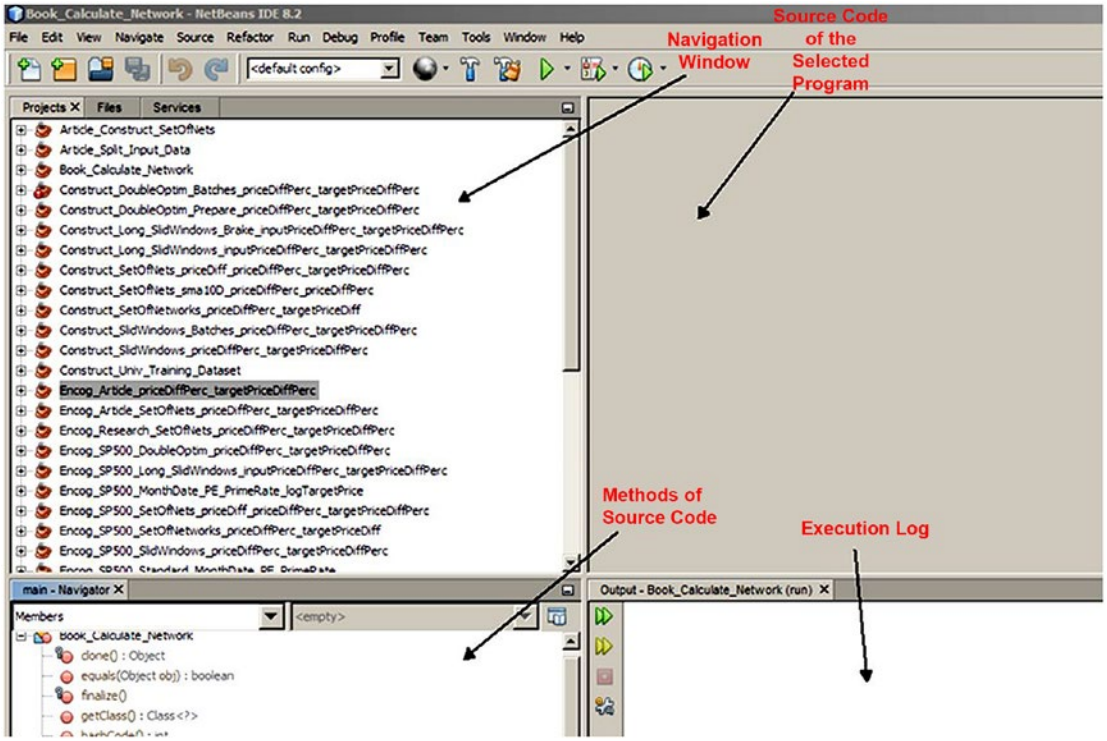


Figure 5-3. *The NetBeans IDE*

To create a new project, select File ➤ New Project. The dialog shown in Figure 5-4 appears.

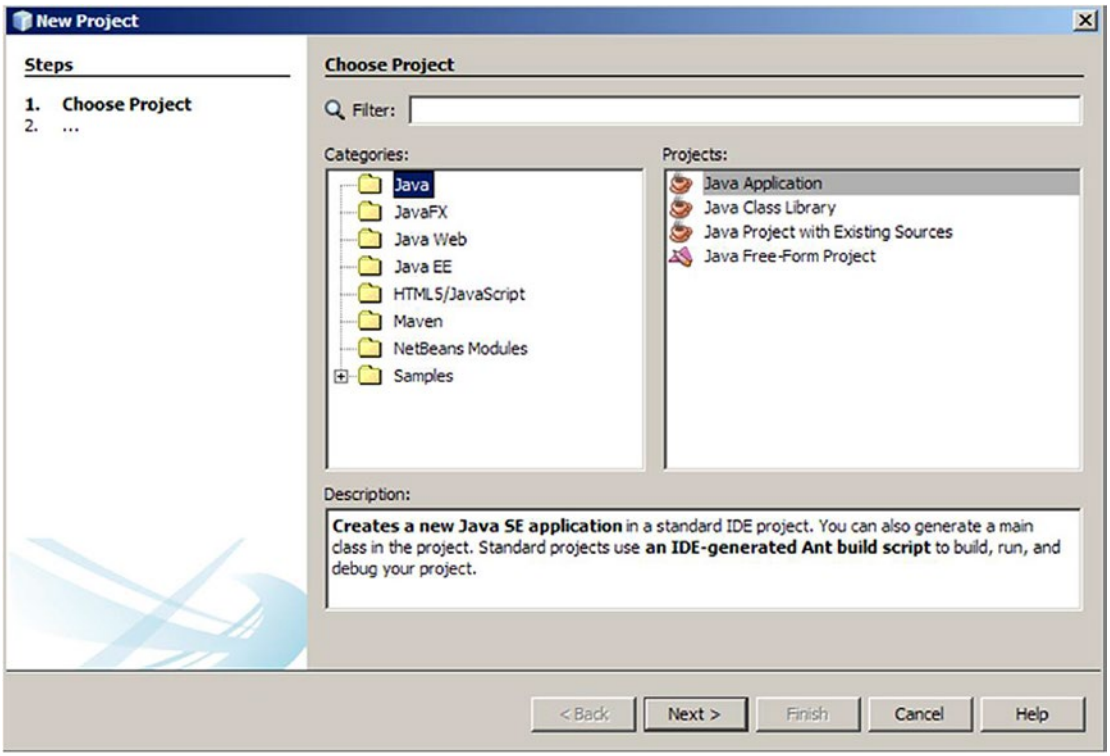


Figure 5-4. *Creating a new project*

Click Next. The dialog shown in Figure 5-5 will appear.

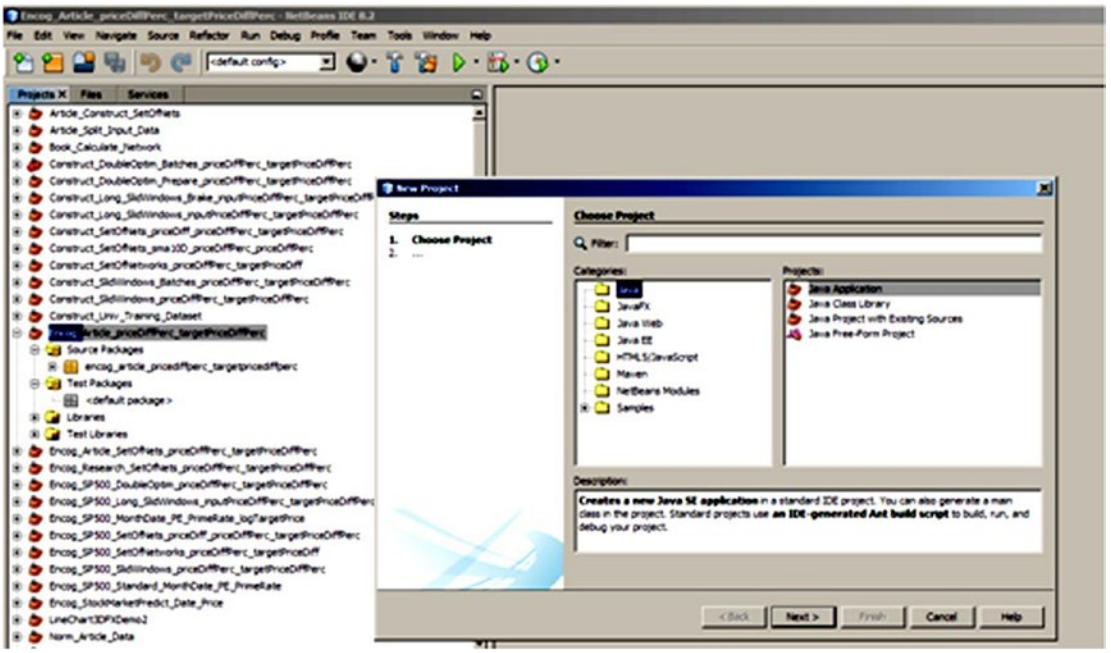


Figure 5-5. *Naming the new project*

Enter the project name **Sample1_Norm** and click the Finish button. Figure 5-6 shows the dialog.

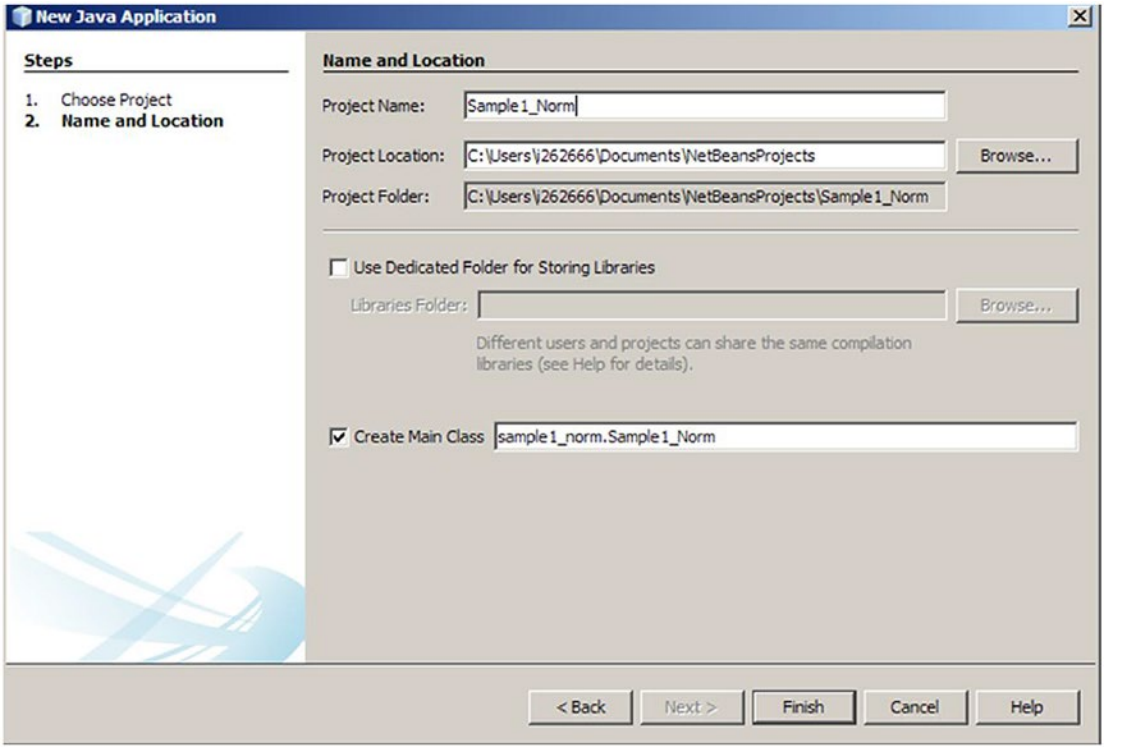


Figure 5-6. *Sample1_Norm project*

The created project is shown in the Navigation window (Figure 5-7).

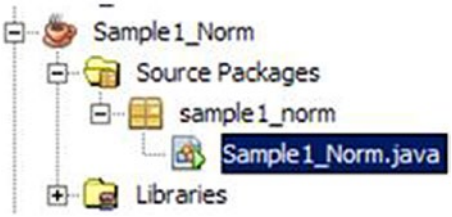


Figure 5-7. *Created project*

The source code became visible in the source code window, as shown in Figure 5-8.

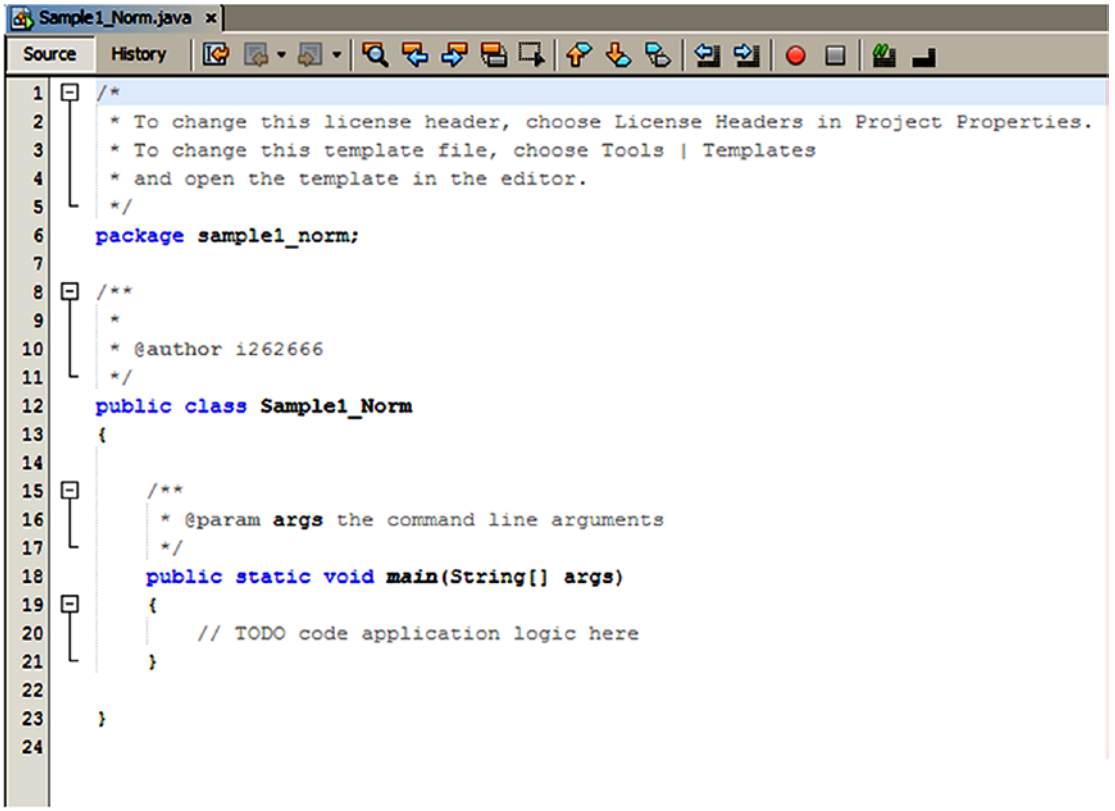


Figure 5-8. *The source code for the new project*

As you can see, this is just a skeleton of the program. Next, add the normalization logic to the program. Listing 5-1 shows the source code for the normalization program.

Listing 5-1. Program Code That Normalizes Both the Training and Test Data Sets

```
// ============================================================================
// Normalize all columns of the input CSV dataset putting the result
// in the output CSV file.
//
// The first column of the input dataset includes the xPoint value and
// the second column is the value of the function at the point X.
// ============================================================================
```

```
package sample2_norm;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.file.*;

public class Sample2_Norm
 {
   // Interval to normalize
   static double Nh =  1;
   static double Nl = -1;

   // First column
   static double minXPointDl = 0.00;
   static double maxXPointDh = 5.00;

   // Second column - target data
   static double minTargetValueDl = 0.00;
   static double maxTargetValueDh = 5.00;

   public static double normalize(double value, double Dh, double Dl)
    {
      double normalizedValue = (value - Dl)*(Nh - Nl)/(Dh - Dl) + Nl;

      return normalizedValue;
    }

   public static void main(String[] args)
    {
      // Config data (comment and uncomment the train or test config data)

      // Config for training
      //String inputFileName = "C:/My_Neural_Network_Book/Book_Examples/
                               Sample2_Train_Real.csv";
      //String outputNormFileName =
```

```
  "C:/My_Neural_Network_Book/Book_Examples/Sample2_Train_Norm.csv";

//Config for testing
String inputFileName = "C:/My_Neural_Network_Book/Book_Examples/
                       Sample2_Test_Real.csv";
String outputNormFileName = "C:/My_Neural_Network_Book/Book_Examples/
                            Sample2_Test_Norm.csv";

 BufferedReader br = null;
 PrintWriter out = null;

 String line = "";
 String cvsSplitBy = ",";
 String strNormInputXPointValue;
 String strNormTargetXPointValue;
 String fullLine;
 double inputXPointValue;
 double targetXPointValue;
 double normInputXPointValue;
 double normTargetXPointValue;
 int i = -1;

 try
  {
   Files.deleteIfExists(Paths.get(outputNormFileName));

   br = new BufferedReader(new FileReader(inputFileName));
   out = new
    PrintWriter(new BufferedWriter(new FileWriter(outputNormFileName)));

   while ((line = br.readLine()) != null)
    {
       i++;

       if(i == 0)
        {
          // Write the label line
          out.println(line);
        }
```

```
            else
            {
             // Break the line using comma as separator
             String[] workFields = line.split(cvsSplitBy);

             inputXPointValue = Double.parseDouble(workFields[0]);
             targetXPointValue = Double.parseDouble( workFields[1]);

             // Normalize these fields
             normInputXPointValue =
               normalize(inputXPointValue, maxXPointDh, minXPointDl);
             normTargetXPointValue =
             normalize(targetXPointValue, maxTargetValueDh, minTargetValueDl);
             // Convert normalized fields to string, so they can be inserted
             //into the output CSV file
             strNormInputXPointValue = Double.toString(normInputXPointValue);
             strNormTargetXPointValue = Double.toString(normTargetXPointValue);

             // Concatenate these fields into a string line with
             //coma separator
             fullLine  =
               strNormInputXPointValue + "," + strNormTargetXPointValue;

             // Put fullLine into the output file
             out.println(fullLine);

            } // End of IF Else


          }    // End of WHILE

        }  // End of TRY
      catch (FileNotFoundException e)
       {
         e.printStackTrace();
         System.exit(1);
       }
      catch (IOException io)
       {
```

```
            io.printStackTrace();
            System.exit(2);
         }
        finally
         {
           if (br != null)
             {
                 try
                  {
                    br.close();
                    out.close();
                  }
                 catch (IOException e1)
                  {
                     e1.printStackTrace();
                     System.exit(3);
                  }
             }
         }

    }

} // End of the class
```

This is a simple program, and it does not need much explanation. Basically, you set the configuration to normalize either the training or testing file by commenting and uncommenting the appropriate configuration sentences. You read the file lines in a loop. For each line, break it into two fields and normalize them. Next, you convert both fields back to strings, combine them into a line, and write the line to the output file.

Table 5-3 shows the normalized training data set.

Table 5-3. *Normalized Training Data Set*

xPoint	Actual Value
-0.94	-0.991
-0.9	-0.975
-0.8	-0.9
-0.7	-0.775
-0.6	-0.6
-0.5	-0.375
-0.4	-0.1
-0.3	0.225
-0.2	0.6

Table 5-4 shows the normalized testing data set.

Table 5-4. *Normalized Testing Data Set*

xPoint	Actual Value
-0.92	-0.984
-0.88	-0.964
-0.84	-0.936
-0.72	-0.804
-0.62	-0.639
-0.48	-0.324
-0.36	0.024
-0.28	0.296
-0.22	0.521

You will use these data sets as the input for the network training and testing.

Building the Neural Network Processing Program

To create a new project, select File ➤ New Project. The dialog shown in Figure 5-9 appears.

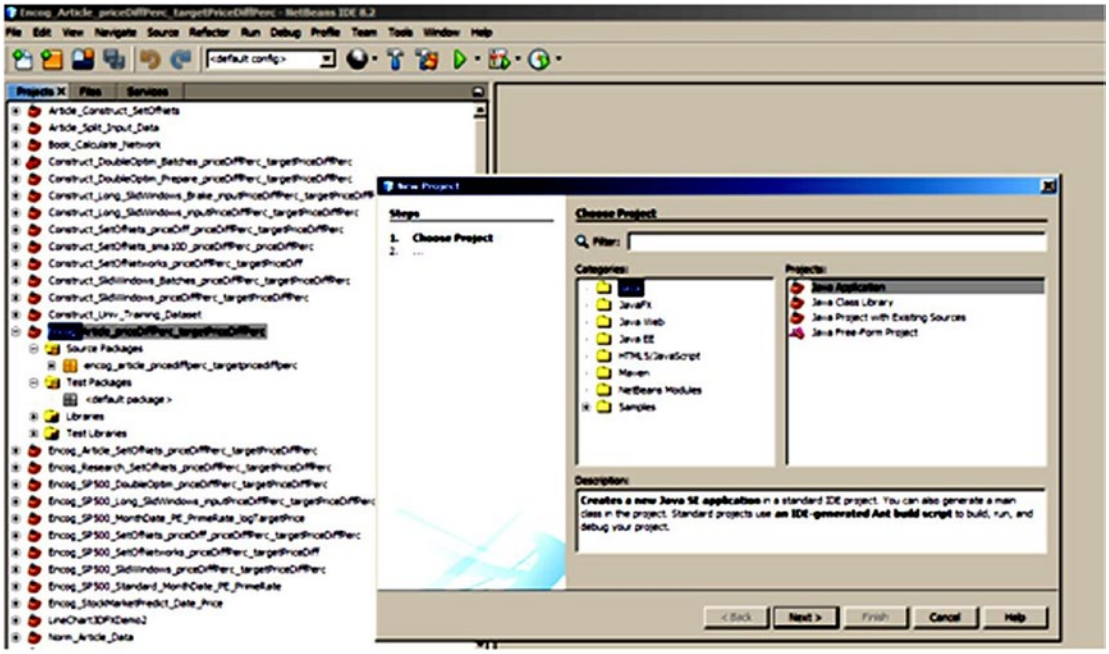


Figure 5-9. *NetBeans IDE, with New Project dialog open*

Click Next. On the next screen, shown in Figure 5-10, enter the project name and click the Finish button.

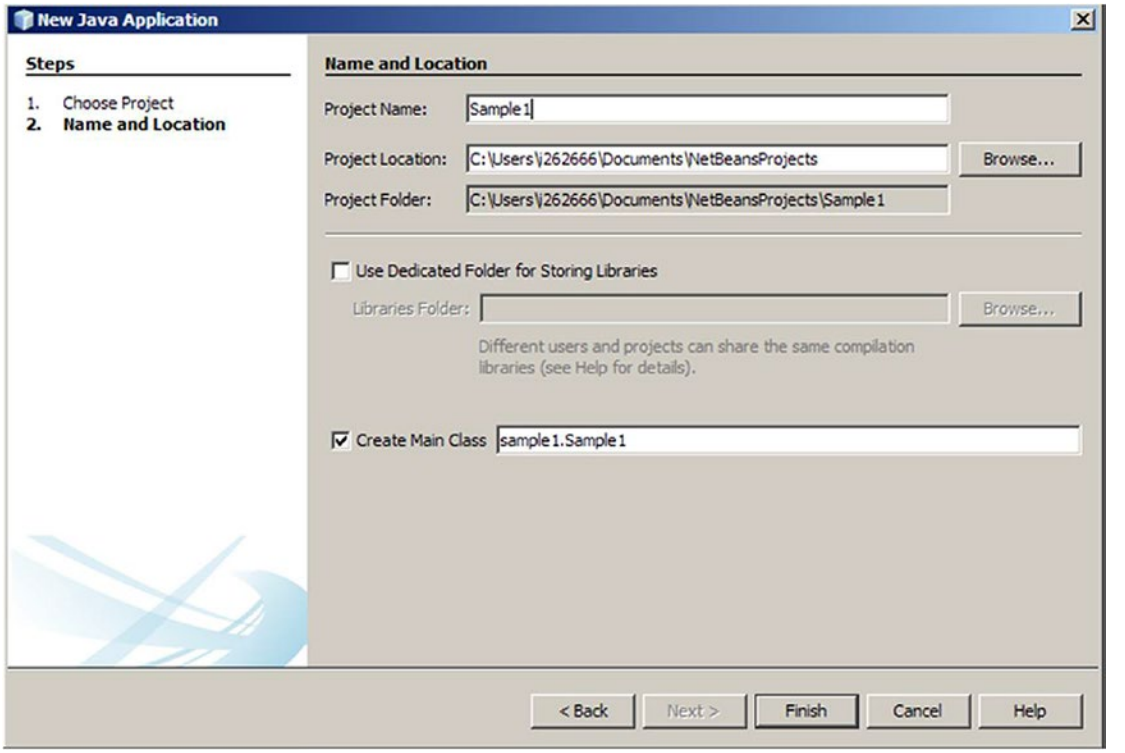


Figure 5-10. *New NetBeans project*

The project is created, and you should see it in the Navigation window (Figure 5-11).

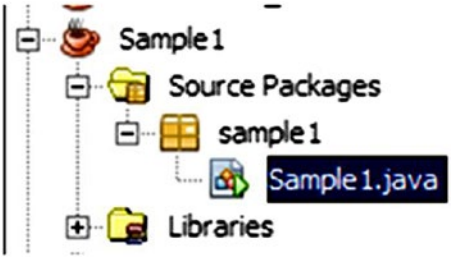


Figure 5-11. *Project Sample1*

The source code of the program appears in the source code window (Figure 5-12).

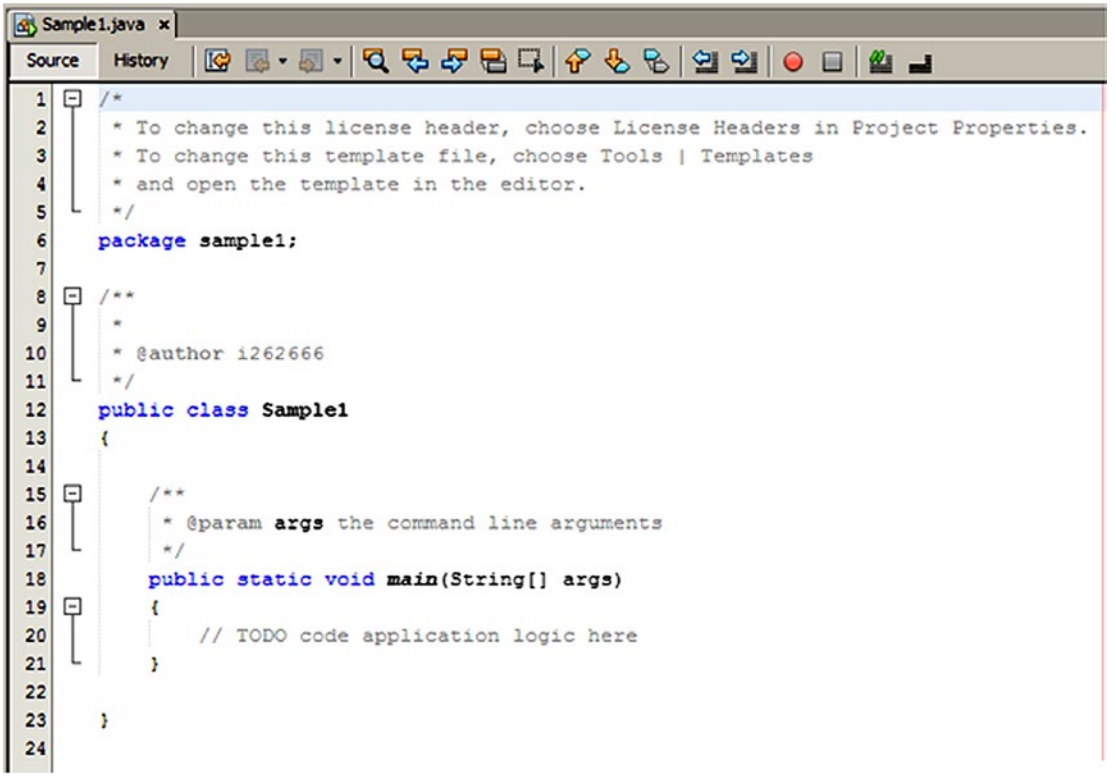


Figure 5-12. *Source code for program Sample1.java1*

Again, this is just the automatically generated skeleton of the Java program. Let's add the necessary logic here. First, include all the necessary import files. There are three groups of import statements (Java imports, Encog imports, and XChart imports), and two of them (the one that belongs to Encog and the next that belongs to XChart) are marked as errors. This is because the NetBeans IDE is unable to find them (Figure 5-13).

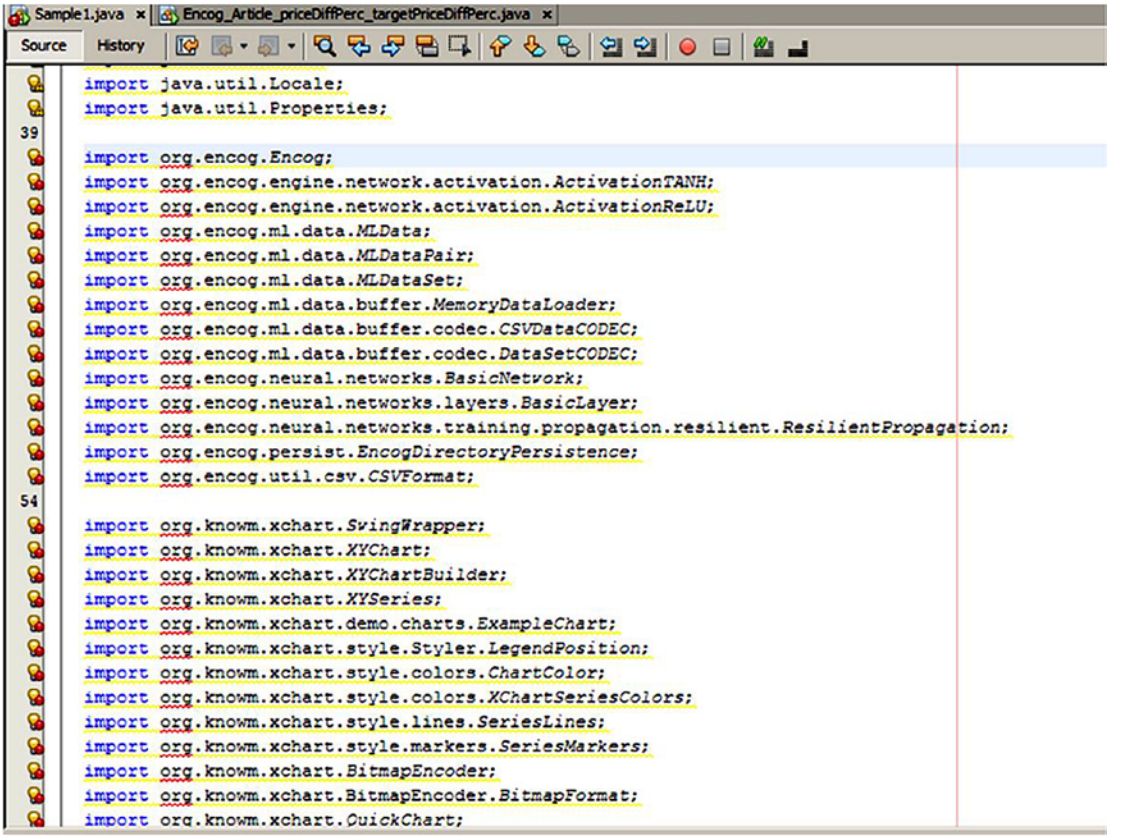


Figure 5-13. *Import statements marked as errors*

To fix this, right-click the project and select Properties. The Project Properties dialog will appear (Figure 5-14).

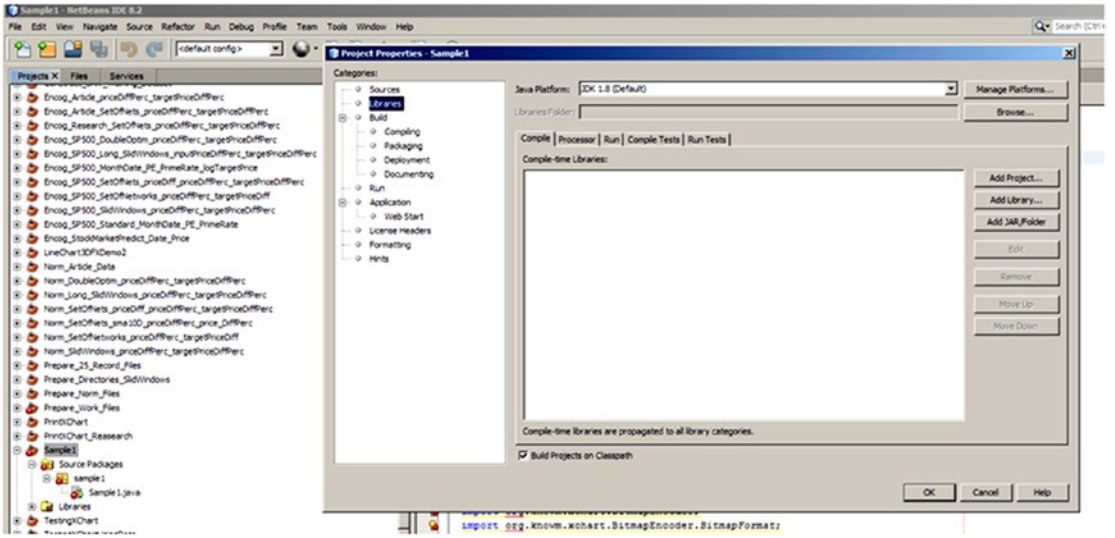


Figure 5-14. *Project Properties dialog*

On the left column of the Project Properties dialog, select Libraries. Click the Add JAR/Folder button on the right of the Project Properties dialog. Click the down arrow in the Java Platform field (at the top of the screen) and go to the location where the Encog package is installed (Figure 5-15).

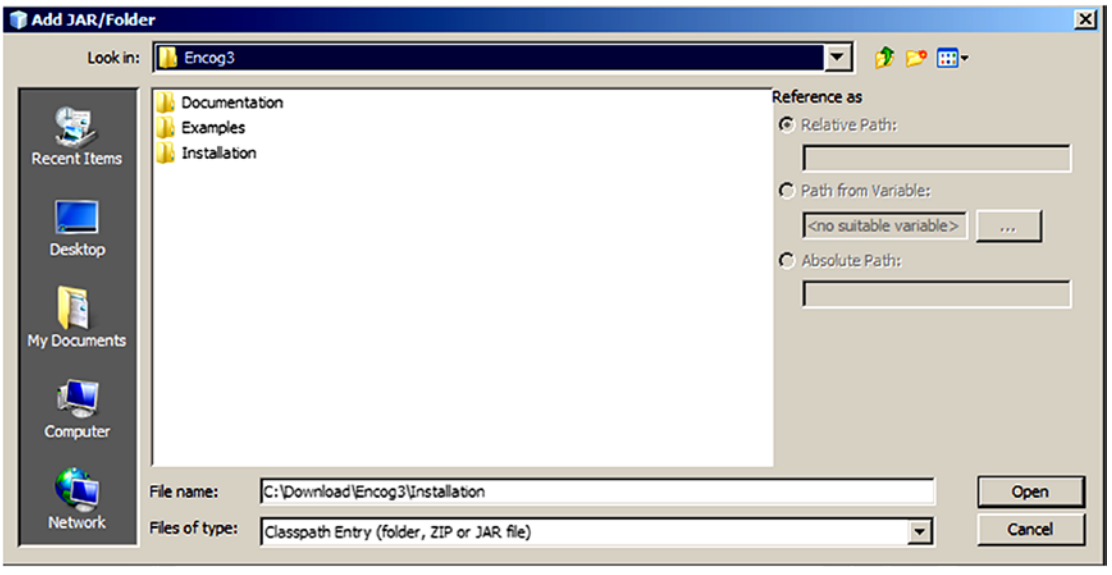


Figure 5-15. *Location where Encog is installed*

Double-click the `Installation` folder, and two JAR files will be displayed (Figure 5-16).

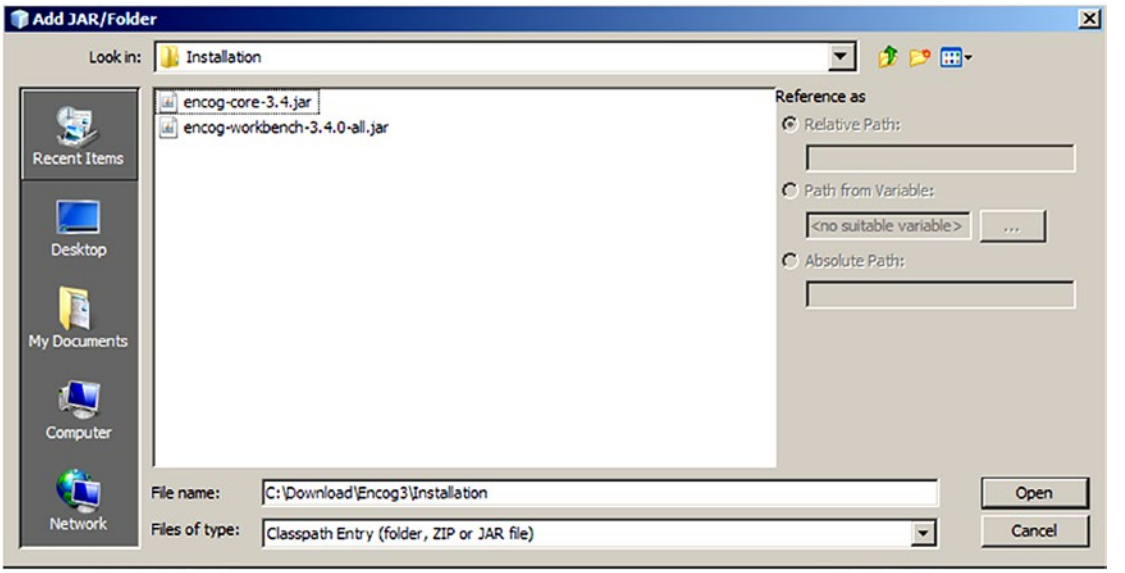


Figure 5-16. *Encog JAR files location*

Select both JAR files and click the Open button. They will be included in a list of JAR files to be added to the NetBeans IDE (Figure 5-17).

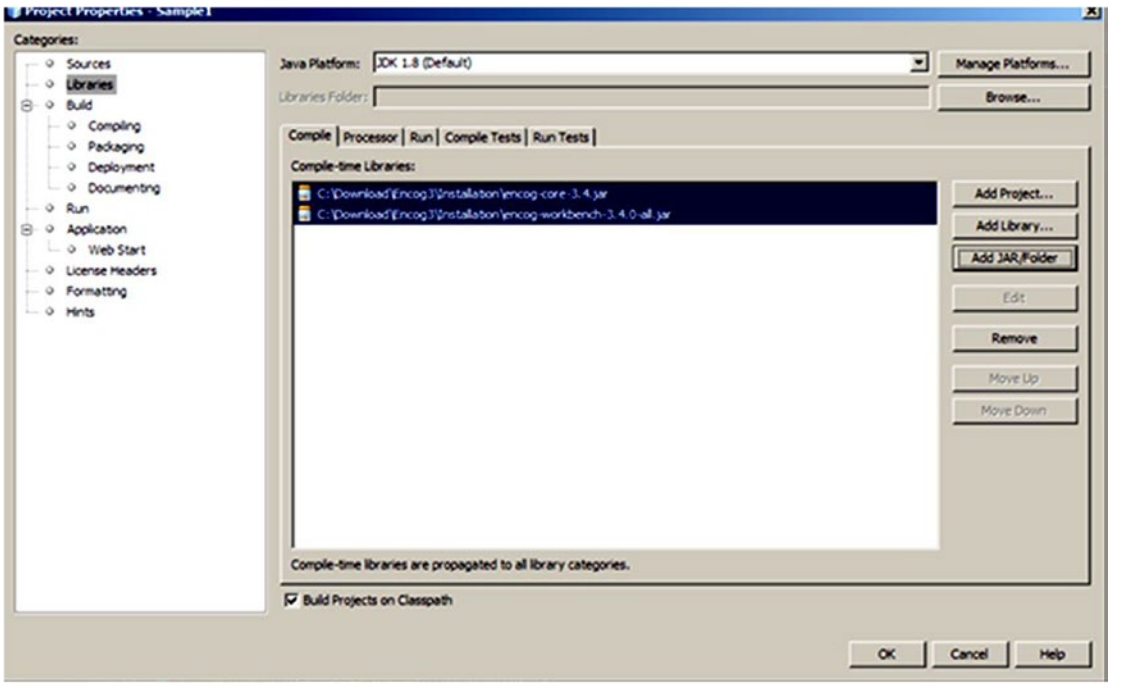


Figure 5-17. *List of Encog JAR files to be included in the NetBeans IDE*

Click the Add JAR/Folder button again. Click again the down arrow of the Java Properties field and go to the location where the XChart package is installed (Figure 5-18).

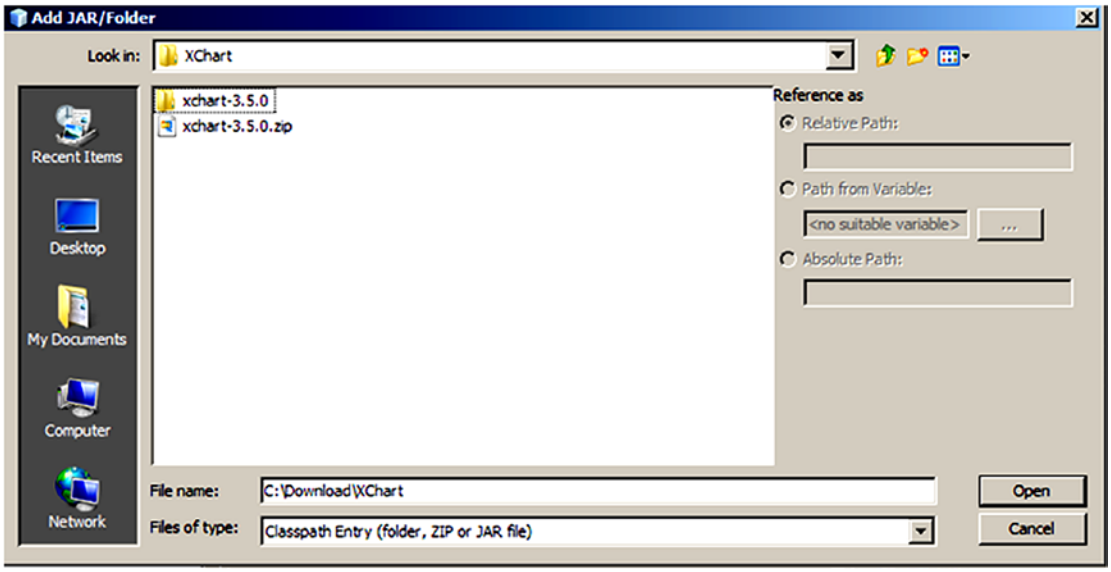


Figure 5-18. *List of XChart JAR files to be included in the NetBeans IDE*

Double-click the `XChart-3.5.0` folder and select two XChart JAR files (Figure 5-19).

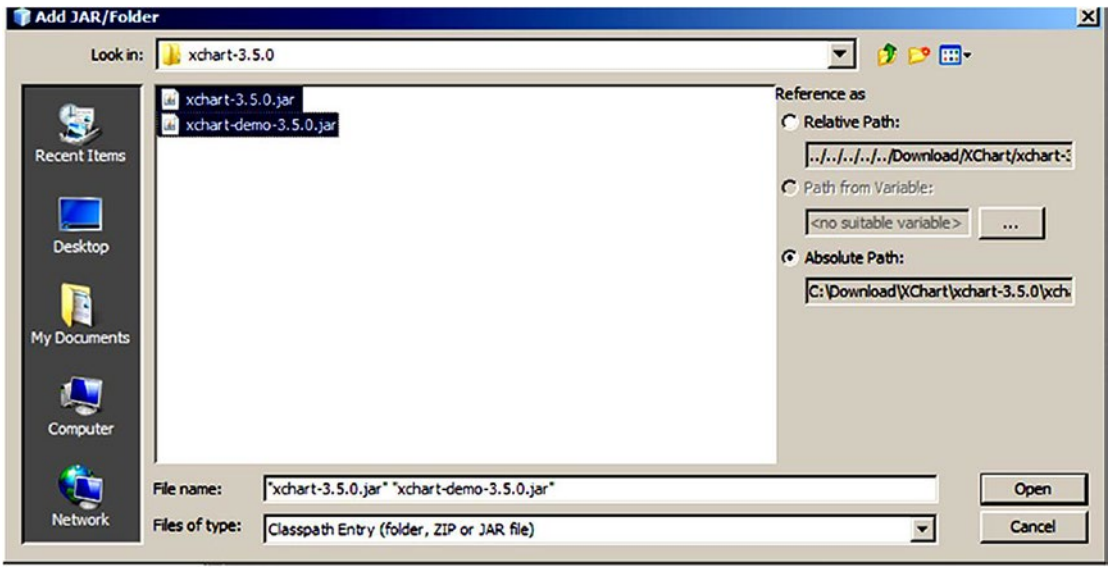


Figure 5-19. *List of XChart JAR files to be included in the NetBeans IDE*

Click the Open button. Now you have the list of four JAR files (from Encog and XChart) to be included in the NetBeans IDE (Figure 5-20).

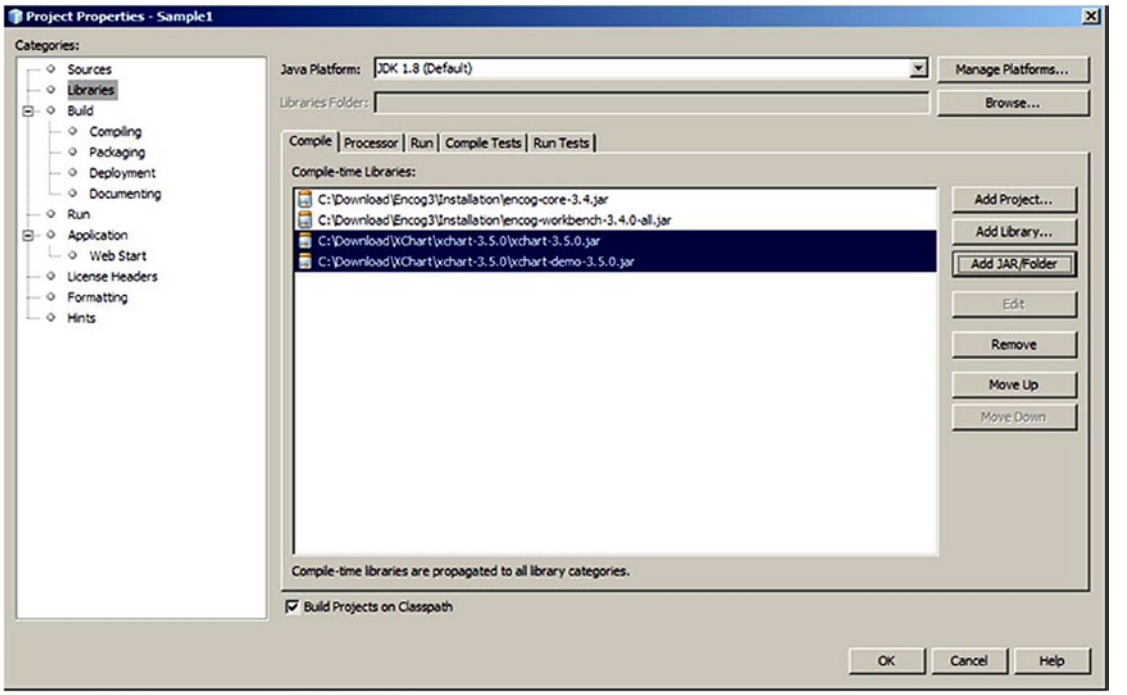


Figure 5-20. *List of JAR files to be included in the NetBeans IDE*

Finally, click OK, and all the errors will disappear.

Instead of doing this for every new project, a better way is to set a new global library. From the main bar, select Tools ➤ Libraries. The dialog shown in Figure 5-21 will appear.

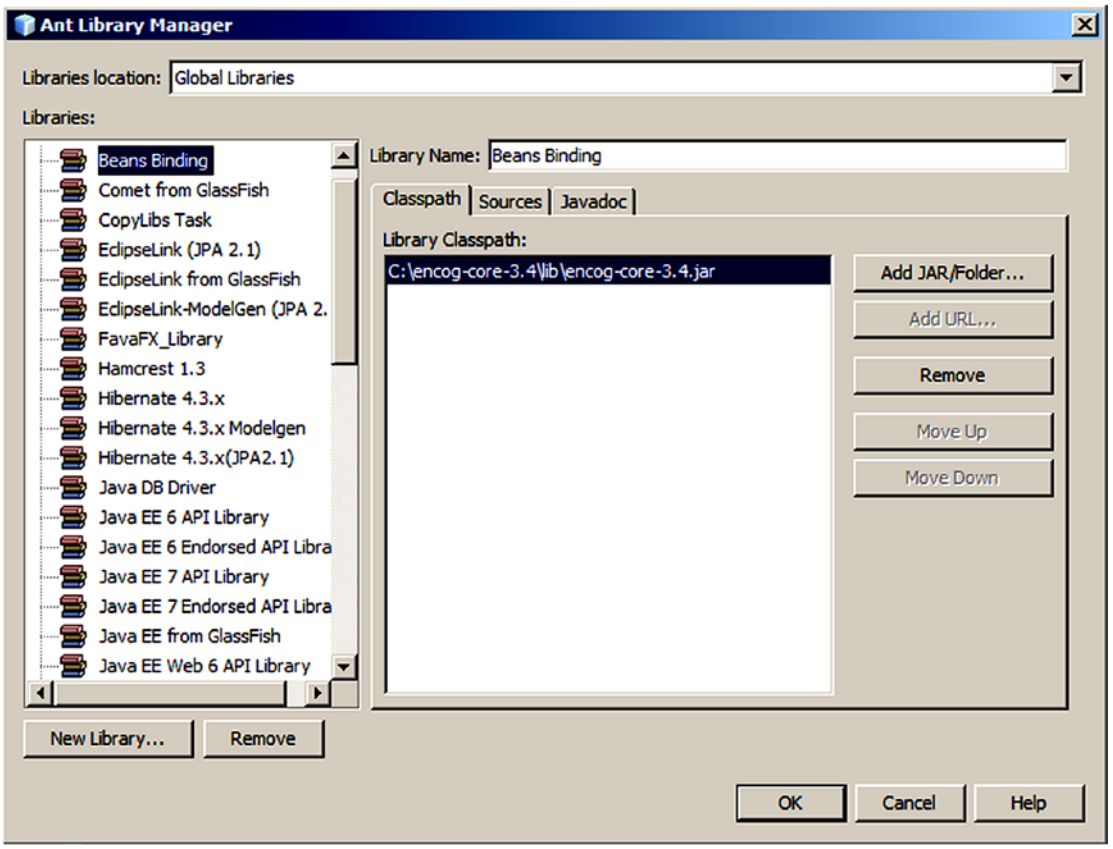


Figure 5-21. *Creating a global library*

Now, you can repeat these same steps on the project level. Do this by clicking the Add JAR/Folder button twice (for Encog and XChart) and add the appropriate JAR files for the Encog and XChart packages.

Program Code

In this section, I will discuss all the important fragments of the program code using Encog. Just remember that you can find documentation about all the Encog APIs and many examples of programming on the Encog website. See Listing 5-2.

Listing 5-2. Network Processing Program Code

```
// ========================================================================
// Approximate the single-variable function which values are given at 9
   points.
// The input train/test files are normalized.
// ========================================================================

package sample2;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.*;
import java.util.Properties;
import java.time.YearMonth;
import java.awt.Color;
import java.awt.Font;
import java.io.BufferedReader;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.LocalDate;
import java.time.Month;
import java.time.ZoneId;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.List;
import java.util.Locale;
import java.util.Properties;
```

```
import org.encog.Encog;
import org.encog.engine.network.activation.ActivationTANH;
import org.encog.engine.network.activation.ActivationReLU;
import org.encog.ml.data.MLData;
import org.encog.ml.data.MLDataPair;
import org.encog.ml.data.MLDataSet;
import org.encog.ml.data.buffer.MemoryDataLoader;
import org.encog.ml.data.buffer.codec.CSVDataCODEC;
import org.encog.ml.data.buffer.codec.DataSetCODEC;
import org.encog.neural.networks.BasicNetwork;
import org.encog.neural.networks.layers.BasicLayer;
import org.encog.neural.networks.training.propagation.resilient.
ResilientPropagation;
import org.encog.persist.EncogDirectoryPersistence;
import org.encog.util.csv.CSVFormat;

import org.knowm.xchart.SwingWrapper;
import org.knowm.xchart.XYChart;
import org.knowm.xchart.XYChartBuilder;
import org.knowm.xchart.XYSeries;
import org.knowm.xchart.demo.charts.ExampleChart;
import org.knowm.xchart.style.Styler.LegendPosition;
import org.knowm.xchart.style.colors.ChartColor;
import org.knowm.xchart.style.colors.XChartSeriesColors;
import org.knowm.xchart.style.lines.SeriesLines;
import org.knowm.xchart.style.markers.SeriesMarkers;
import org.knowm.xchart.BitmapEncoder;
import org.knowm.xchart.BitmapEncoder.BitmapFormat;
import org.knowm.xchart.QuickChart;
import org.knowm.xchart.SwingWrapper;
public class Sample2 implements ExampleChart<XYChart>
{
    // Interval to normalize
   static double Nh =  1;
   static double Nl = -1;
```

```
// First column
static double minXPointDl = 0.00;
static double maxXPointDh = 5.00;

// Second column - target data
static double minTargetValueDl = 0.00;
static double maxTargetValueDh = 5.00;

static double doublePointNumber = 0.00;
static int intPointNumber = 0;
static InputStream input = null;
static int intNumberOfRecordsInTrainFile;
static double[] arrPrices = new double[2500];
static double normInputXPointValue = 0.00;
static double normPredictXPointValue = 0.00;
static double normTargetXPointValue = 0.00;
static double normDifferencePerc = 0.00;
static double denormInputXPointValue = 0.00;
static double denormPredictXPointValue = 0.00;
static double denormTargetXPointValue = 0.00;
static double valueDifference = 0.00;
static int returnCode  = 0;
static int numberOfInputNeurons;
static int numberOfOutputNeurons;
static int intNumberOfRecordsInTestFile;
static String trainFileName;
static String priceFileName;
static String testFileName;
static String chartTrainFileName;
static String chartTestFileName;
static String networkFileName;
static int workingMode;
static String cvsSplitBy = ",";
static List<Double> xData = new ArrayList<Double>();
static List<Double> yData1 = new ArrayList<Double>();
static List<Double> yData2 = new ArrayList<Double>();
```

```
static XYChart Chart;
@Override
public XYChart getChart()
 {

  // Create Chart
  Chart = new  XYChartBuilder().width(900).height(500).title(getClass().
            getSimpleName()).xAxisTitle("x").yAxisTitle("y= f(x)").build();

  // Customize Chart
  Chart.getStyler().setPlotBackgroundColor(ChartColor.
  getAWTColor(ChartColor.GREY));
  Chart.getStyler().setPlotGridLinesColor(new Color(255, 255, 255));
  Chart.getStyler().setChartBackgroundColor(Color.WHITE);
  Chart.getStyler().setLegendBackgroundColor(Color.PINK);
  Chart.getStyler().setChartFontColor(Color.MAGENTA);
  Chart.getStyler().setChartTitleBoxBackgroundColor(new Color(0, 222, 0));
  Chart.getStyler().setChartTitleBoxVisible(true);
  Chart.getStyler().setChartTitleBoxBorderColor(Color.BLACK);
  Chart.getStyler().setPlotGridLinesVisible(true);
  Chart.getStyler().setAxisTickPadding(20);
  Chart.getStyler().setAxisTickMarkLength(15);
  Chart.getStyler().setPlotMargin(20);
  Chart.getStyler().setChartTitleVisible(false);
  Chart.getStyler().setChartTitleFont(new Font(Font.MONOSPACED, Font.BOLD, 24));
  Chart.getStyler().setLegendFont(new Font(Font.SERIF, Font.PLAIN, 18));
  Chart.getStyler().setLegendPosition(LegendPosition.InsideSE);
  Chart.getStyler().setLegendSeriesLineLength(12);
  Chart.getStyler().setAxisTitleFont(new Font(Font.SANS_SERIF, Font.ITALIC, 18));
  Chart.getStyler().setAxisTickLabelsFont(new Font(Font.SERIF, Font.
  PLAIN, 11));
  Chart.getStyler().setDatePattern("yyyy-MM");
  Chart.getStyler().setDecimalPattern("#0.00");

   // Set the workin mode the program should run (workingMode = 1 – training,
   // workingMode = 2 – testing)
```

```
 workingMode = 1;
try
   {
     )
     If (workingMode == 1)
       {
        // Config for training the network
        workingMode = 1;
        intNumberOfRecordsInTrainFile = 10;
        trainFileName = "C:/My_Neural_Network_Book/Book_Examples/
                        Sample2_Train_Norm.csv";
        chartTrainFileName = "Sample2_XYLine_Train_Results_Chart";
     }
   else
     {
       // Config for testing the trained network
       // workingMode = 2;
       // intNumberOfRecordsInTestFile = 10;
       // testFileName = "C:/My_Neural_Network_Book/Book_Examples/
                          Sample2_Test_Norm.csv";
       //  chartTestFileName = "XYLine_Test_Results_Chart";
     }

     // Common configuration data
     networkFileName = "C:/Book_Examples/Sample2_Saved_Network_File.csv";
     numberOfInputNeurons = 1;
     numberOfOutputNeurons = 1;

     // Check the working mode to run

     // Training mode.
     if(workingMode == 1)
      {
         File file1 = new File(chartTrainFileName);
         File file2 = new File(networkFileName);

         if(file1.exists())
           file1.delete();
```

```
            if(file2.exists())
              file2.delete();

            returnCode = 0;    // Clear the return code variable

            do
             {
               returnCode = trainValidateSaveNetwork();

             } while (returnCode > 0);

          }

        // Test mode.
        if(workingMode == 2)
         {
           // Test using the test dataset as input
           loadAndTestNetwork();
         }

       }
      catch (NumberFormatException e)
       {
           System.err.println("Problem parsing workingMode.
           workingMode = " + workingMode);
           System.exit(1);
        }
      catch (Throwable t)
        {
            t.printStackTrace();
           System.exit(1);
        }
      finally
        {
          Encog.getInstance().shutdown();
        }
         Encog.getInstance().shutdown();

          return Chart;

}  // End of the method
```

```
//-------------------------------------------------------------
// Load CSV to memory.
// @return The loaded dataset.
// -------------------------------------------------------------
public static MLDataSet loadCSV2Memory(String filename, int input, int
ideal, boolean headers,
  CSVFormat
      format, boolean significance)
  {
     DataSetCODEC codec = new CSVDataCODEC(new File(filename), format,
     headers, input, ideal,
       significance);
     MemoryDataLoader load = new MemoryDataLoader(codec);
     MLDataSet dataset = load.external2Memory();
     return dataset;
  }

// ================================================================
//  The main method.
//  @param Command line arguments. No arguments are used.
// ================================================================
public static void main(String[] args)
 {
   ExampleChart<XYChart> exampleChart = new Sample2();
   XYChart Chart = exampleChart.getChart();
   new SwingWrapper<XYChart>(Chart).displayChart();
 } // End of the main method

//================================================================
// Training method. Train, validate, and save the trained network file
//================================================================
static public int trainValidateSaveNetwork()
 {
   // Load the training CSV file in memory
   MLDataSet trainingSet =
     loadCSV2Memory(trainFileName,numberOfInputNeurons,numberOfOutputNeurons,
       true,CSVFormat.ENGLISH,false);
```

```
// create a neural network
BasicNetwork network = new BasicNetwork();

// Input layer
network.addLayer(new BasicLayer(null,true,1));

// Hidden layer
network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
network.addLayer(new BasicLayer(new ActivationTANH(),true,5));

// Output layer
network.addLayer(new BasicLayer(new ActivationTANH(),false,1));

network.getStructure().finalizeStructure();
network.reset();

// Train the neural network
final ResilientPropagation train = new ResilientPropagation(network,
trainingSet);

int epoch = 1;
returnCode = 0;

do
 {
     train.iteration();
     System.out.println("Epoch #" + epoch + " Error:" + train.getError());
    epoch++;

    if (epoch >= 500 && network.calculateError(trainingSet) > 0.000000031)
     {
        returnCode = 1;
```

```
            System.out.println("Try again");
             return returnCode;
        }
    } while (network.calculateError(trainingSet) > 0.00000003);

    // Save the network file
    EncogDirectoryPersistence.saveObject(new File(networkFileName),network);

    System.out.println("Neural Network Results:");

    double sumNormDifferencePerc = 0.00;
    double averNormDifferencePerc = 0.00;
    double maxNormDifferencePerc = 0.00;

    int m = -1;
    double xPointer = -1.00;

    for(MLDataPair pair: trainingSet)
      {
          m++;
          xPointer = xPointer + 2.00;

          //if(m == 0)
          // continue;

           final MLData output = network.compute(pair.getInput());

           MLData inputData = pair.getInput();
           MLData actualData = pair.getIdeal();
           MLData predictData = network.compute(inputData);

           // Calculate and print the results
           normInputXPointValue = inputData.getData(0);
           normTargetXPointValue = actualData.getData(0);
           normPredictXPointValue = predictData.getData(0);

           denormInputXPointValue = ((minXPointDl -
             maxXPointDh)*normInputXPointValue - Nh*minXPointDl +
                maxXPointDh *Nl)/(Nl - Nh);
           denormTargetXPointValue = ((minTargetValueDl - maxTargetValueDh)*
```

```
        normTargetXPointValue - Nh*minTargetValueDl +
          maxTargetValueDh*Nl)/(Nl - Nh);

      denormPredictXPointValue =((minTargetValueDl - maxTargetValueDh)*
        normPredictXPointValue - Nh*minTargetValueDl +
          maxTargetValueDh*Nl)/(Nl - Nh);

      valueDifference = Math.abs(((denormTargetXPointValue -
        denormPredictXPointValue)/denormTargetXPointValue)*100.00);

      System.out.println ("xPoint = " + denormTargetXPointValue +
           "  denormPredictXPointValue = " + denormPredictXPointValue +
             "  valueDifference = " + valueDifference);

      sumNormDifferencePerc = sumNormDifferencePerc + valueDifference;

      if (valueDifference > maxNormDifferencePerc)
        maxNormDifferencePerc = valueDifference;

      xData.add(denormInputXPointValue);
      yData1.add(denormTargetXPointValue);
      yData2.add(denormPredictXPointValue);

}   // End for pair loop

XYSeries series1 = Chart.addSeries("Actual data", xData, yData1);
XYSeries series2 = Chart.addSeries("Predict data", xData, yData2);

series1.setLineColor(XChartSeriesColors.BLUE);
series2.setMarkerColor(Color.ORANGE);
series1.setLineStyle(SeriesLines.SOLID);
series2.setLineStyle(SeriesLines.SOLID);

try
 {
   //Save the chart image
   BitmapEncoder.saveBitmapWithDPI(Chart, chartTrainFileName,
   BitmapFormat.JPG, 100);
   System.out.println ("Train Chart file has been saved") ;
 }
```

```
        catch (IOException ex)
         {
          ex.printStackTrace();
          System.exit(3);
         }
         // Finally, save this trained network
         EncogDirectoryPersistence.saveObject(new File(networkFileName),network);
         System.out.println ("Train Network has been saved") ;

         averNormDifferencePerc  = sumNormDifferencePerc/
         intNumberOfRecordsInTrainFile;

         System.out.println(" ");
         System.out.println("maxErrorDifferencePerc = " + maxNormDifferencePerc + "
             averErrorDifferencePerc = " + averNormDifferencePerc);

         returnCode = 0;
         return returnCode;

    }   // End of the method

   //=====================================================================
   // Load and test the trained network at the points not used in training.
   //=====================================================================
   static public void loadAndTestNetwork()
    {
     System.out.println("Testing the networks results");

     List<Double> xData = new ArrayList<Double>();
     List<Double> yData1 = new ArrayList<Double>();
     List<Double> yData2 = new ArrayList<Double>();

     double targetToPredictPercent = 0;
     double maxGlobalResultDiff = 0.00;
     double averGlobalResultDiff = 0.00;
     double sumGlobalResultDiff = 0.00;
     double maxGlobalIndex = 0;
     double normInputXPointValueFromRecord = 0.00;
```

```
double normTargetXPointValueFromRecord = 0.00;
double normPredictXPointValueFromRecord = 0.00;

BufferedReader br4;
BasicNetwork network;
int k1 = 0;
int k3 = 0;

maxGlobalResultDiff = 0.00;
averGlobalResultDiff = 0.00;
sumGlobalResultDiff = 0.00;

// Load the test dataset into memory
MLDataSet testingSet =
loadCSV2Memory(testFileName,numberOfInputNeurons,numberOfOutputNeurons,
  true,CSVFormat.ENGLISH,false);

// Load the saved trained network
network =
 (BasicNetwork)EncogDirectoryPersistence.loadObject(new
 File(networkFileName));

int i = - 1;
double xPoint = -0.00;

for (MLDataPair pair:  testingSet)
 {
      i++;
      xPoint = xPoint + 2.00;

      MLData inputData = pair.getInput();
      MLData actualData = pair.getIdeal();
      MLData predictData = network.compute(inputData);

      // These values are Normalized as the whole input is
      normInputXPointValueFromRecord = inputData.getData(0);
      normTargetXPointValueFromRecord = actualData.getData(0);
      normPredictXPointValueFromRecord = predictData.getData(0);
```

```
        //  De-normalize the obtained values
        denormInputXPointValue = ((minXPointDl - maxXPointDh)*
          normInputXPointValueFromRecord - Nh*minXPointDl +
             maxXPointDh*Nl)/(Nl - Nh);

        denormTargetXPointValue = ((minTargetValueDl - maxTargetValueDh)*
          normTargetXPointValueFromRecord - Nh*minTargetValueDl +
             maxTargetValueDh*Nl)/(Nl - Nh);

        denormPredictXPointValue =((minTargetValueDl - maxTargetValueDh)*
          normPredictXPointValueFromRecord - Nh*minTargetValueDl +
             maxTargetValueDh*Nl)/(Nl - Nh);

        targetToPredictPercent = Math.abs((denormTargetXPointValue -
          denormPredictXPointValue)/denormTargetXPointValue*100);

        System.out.println("xPoint = " + denormInputXPointValue +
          "  denormTargetXPointValue = " + denormTargetXPointValue +
            "  denormPredictXPointValue = " + denormPredictXPointValue +
              "   targetToPredictPercent = " + targetToPredictPercent);

        if (targetToPredictPercent > maxGlobalResultDiff)
           maxGlobalResultDiff = targetToPredictPercent;

        sumGlobalResultDiff = sumGlobalResultDiff + targetToPredictPercent;

        // Populate chart elements
        xData.add(denormInputXPointValue);
        yData1.add(denormTargetXPointValue);
        yData2.add(denormPredictXPointValue);

     }  // End for pair loop

    // Print the max and average results

    System.out.println(" ");
    averGlobalResultDiff = sumGlobalResultDiff/intNumberOfRecordsInTestFile;

    System.out.println("maxErrorDifferencePercent = " + maxGlobalResultDiff);
    System.out.println("averErrorDifferencePercent = " + averGlobalResultDiff);
```

```
        // All testing batch files have been processed
        XYSeries series1 = Chart.addSeries("Actual", xData, yData1);
        XYSeries series2 = Chart.addSeries("Predicted", xData, yData2);

        series1.setLineColor(XChartSeriesColors.BLUE);
        series2.setMarkerColor(Color.ORANGE);
        series1.setLineStyle(SeriesLines.SOLID);
        series2.setLineStyle(SeriesLines.SOLID);

        // Save the chart image
        try
         {
           BitmapEncoder.saveBitmapWithDPI(Chart, chartTestFileName ,
           BitmapFormat.JPG, 100);
         }
        catch (Exception bt)
         {
           bt.printStackTrace();
         }

        System.out.println ("The Chart has been saved");

        System.out.println("End of testing for test records");

    } // End of the method

} // End of the class
```

At the top, there is a set of instructions required by the XChart package, and they allow you to configure the way the chart should look (Listing 5-3).

Listing 5-3. Set of Instructions That Is Required by the XChart Package

```
static XYChart Chart;

  @Override
  public XYChart getChart()
   {

    // Create Chart
    Chart = new  XYChartBuilder().width(900).height(500).title(getClass().
              getSimpleName()).xAxisTitle("x").yAxisTitle("y= f(x)").build();
```

```
// Customize Chart
Chart.getStyler().setPlotBackgroundColor(ChartColor.
getAWTColor(ChartColor.GREY));
Chart.getStyler().setPlotGridLinesColor(new Color(255, 255, 255));
Chart.getStyler().setChartBackgroundColor(Color.WHITE);
Chart.getStyler().setLegendBackgroundColor(Color.PINK);
Chart.getStyler().setChartFontColor(Color.MAGENTA);
Chart.getStyler().setChartTitleBoxBackgroundColor(new Color(0, 222, 0));
Chart.getStyler().setChartTitleBoxVisible(true);
Chart.getStyler().setChartTitleBoxBorderColor(Color.BLACK);
Chart.getStyler().setPlotGridLinesVisible(true);
Chart.getStyler().setAxisTickPadding(20);
Chart.getStyler().setAxisTickMarkLength(15);
Chart.getStyler().setPlotMargin(20);
Chart.getStyler().setChartTitleVisible(false);
Chart.getStyler().setChartTitleFont(new Font(Font.MONOSPACED, Font.BOLD, 24));
Chart.getStyler().setLegendFont(new Font(Font.SERIF, Font.PLAIN, 18));
Chart.getStyler().setLegendPosition(LegendPosition.InsideSE);
Chart.getStyler().setLegendSeriesLineLength(12);
Chart.getStyler().setAxisTitleFont(new Font(Font.SANS_SERIF, Font.ITALIC, 18));
Chart.getStyler().setAxisTickLabelsFont(new Font(Font.SERIF, Font.PLAIN, 11));
Chart.getStyler().setDatePattern("yyyy-MM");
Chart.getStyler().setDecimalPattern("#0.00");
```

The program can be run in two modes. In the first mode (training, `workingMode` = 1), the program trains the network, saves the trained network on disk, prints the results, displays the chart results, and saves the chart on disk. In the second mode (testing, `workingMode` = 2), the program loads the previously saved trained network, calculates the predicted values at the points that were not used in the network training, prints the results, displays the chart, and saves the chart on disk.

The program should always be run in the training mode first, because the second mode depends on the training results produced in the training mode. The configuration is currently set to run the program in training mode (see Listing 5-4).

Listing 5-4. Code Fragment of the Training Method Code

```
// Set the workin mode the program should run:
(workingMode = 1 – training, workingMode = 2 – testing)

workingMode = 1;

try
   {
     If (workingMode == 1)
       {
        // Config for training the network
        workingMode = 1;
        intNumberOfRecordsInTrainFile = 10;
        trainFileName = "C:/My_Neural_Network_Book/Book_Examples/
                        Sample2_Train_Norm.csv";
        chartTrainFileName = "Sample2_XYLine_Train_Results_Chart";
       }
     else
       {
         // Config for testing the trained network
         // workingMode = 2;
         // intNumberOfRecordsInTestFile = 10;
         // testFileName = "C:/My_Neural_Network_Book/Book_Examples/
                           Sample2_Test_Norm.csv";
         //  chartTestFileName = "XYLine_Test_Results_Chart";
       }

// Common configuration statements  (stays always uncommented)
networkFileName = "C:/Book_Examples/Saved_Network_File.csv";
numberOfInputNeurons = 1;
numberOfOutputNeurons = 1;
```

Because `workingMode` is currently set to 1, the program executes the training method called `trainValidateSaveNetwork()`; otherwise, it calls the testing method called `loadAndTestNetwork()` (see Listing 5-5).

Listing 5-5. Checking the workingMode Value and Executing the Appropriate Method

```
// Check the working mode

if(workingMode == 1)
  {
      // Training mode.

      File file1 = new File(chartTrainFileName);
      File file2 = new File(networkFileName);

      if(file1.exists())
        file1.delete();

      if(file2.exists())
        file2.delete();

     trainValidateSaveNetwork();
  }

  if(workingMode == 2)
    {
       // Test using the test dataset as input
       loadAndTestNetwork();
    }

  }
catch (NumberFormatException e)
  {
     System.err.println("Problem parsing workingMode. workingMode = " +
     workingMode);
     System.exit(1);
  }
catch (Throwable t)
  {
      t.printStackTrace();
      System.exit(1);
  }
```

```
finally
  {
      Encog.getInstance().shutdown();
   }
```

Listing 5-6 shows the training method logic. This method trains the network, validates it, and saves the trained network file on disk (to be used later by the testing method). The method loads the training data set into memory. The first parameter is the name of the input training data set. The second and third parameters indicate the number of input and output neurons in the network. The fourth parameter (`true`) indicates that the data set has a label record. The remaining parameters specify the file format and the language.

Listing 5-6. Fragments of the Network Training Logic

```
MLDataSet trainingSet =
loadCSV2Memory(trainFileName,numberOfInputNeurons,numberOfOutputNeurons,
          true,CSVFormat.ENGLISH,false);
```

After loading the training data set in memory, a new neural network is built by creating the basic network and adding the input, hidden, and output layers to it.

```
// create a neural network
BasicNetwork network = new BasicNetwork();
```

Here's how to add the input layer:

```
network.addLayer(new BasicLayer(null,true,1));
```

The first parameter (`null`) indicates that this is the input layer (no activation function). Enter `true` as the second parameter for the input and hidden layers, and enter `false` for the output layer. The third parameter shows the number of neurons in the layer. Next you add the hidden layer.

```
network.addLayer(new BasicLayer(new ActivationTANH(),true,2));
```

The first parameter specifies the activation function to be used (`ActivationTANH()`).

Alternatively, other activation functions can be used such as the sigmoid function called `ActivationSigmoid()`, the logarithmic function called `ActivationLOG()`, the linear relay called `ActivationReLU()`, and so on. The third parameter specifies the number of neurons in this layer. To add the second hidden layer, simply repeat the previous statement.

Finally, add the output layer, as shown here:

```
network.addLayer(new BasicLayer(new ActivationTANH(),false,1));
```

The third parameter specifies the number of neurons in the output layer. The next two statements finalize the creation of the network:

```
network.getStructure().finalizeStructure();
network.reset();
```

To train the newly built network, you specify the type of backpropagation. Here, you specify resilient propagation; it's the most advanced propagation type. Alternatively, the regular backpropagation type can be specified here.

```
final ResilientPropagation train = new ResilientPropagation(network,
trainingSet);
```

While the network is trained, you loop over the network. On each step of the loop, you get the next training iteration number, increase the epoch number (see Chapter 2 for the epoch definition), and check whether the network error for the current iteration can clear the error limit being set to 0.00000003. When the error on the current iteration finally becomes less than the error limit, you exit the loop. The network has been trained, and you save the trained network on disk. The network also stays in memory.

```
int epoch = 1;

do
     {
         train.iteration();
         System.out.println("Epoch #" + epoch + " Error:" + train.getError());

         epoch++;
    } while (network.calculateError(trainingSet) > 0.00000046);

   // Save the network file
   EncogDirectoryPersistence.saveObject(new File(networkFileName),network);
```

The next section of the code retrieves input, actual, and predict values for each record in the training data set. First, the `inputData`, `actualData`, and `predictData` objects are created.

```
MLData inputData = pair.getInput();
MLData actualData = pair.getIdeal();
MLData predictData = network.compute(inputData);
```

Having done that, you iterate over the `MLDataPair` pair object by executing the following instructions:

```
normInputXPointValue = inputData.getData(0);
normTargetXPointValue = actualData.getData(0);
normPredictXPointValue = predictData.getData(0);
```

A single field in `inputData`, `actualData`, and `predictData` objects has the displacement zero. In this example, there is only one input field and one output field in a record. Should the record have two input fields, you would use the following statements to retrieve all the input fields:

```
normInputXPointValue1 = inputData.getData(0);
normInputXPointValue2 = inputData.getData(1);
```

Conversely, should the record have two target fields, you would use similar statements to retrieve all the target fields, as shown here:

```
normTargeValue1 = actualData.getData(0);
normTargeValue2 = actualData.getData(1);
```

The predicted value is processed in a similar way. The predicted value predicts the target value for the next point. The values being retrieved from the network are normalized because the training data set that the network processes has been normalized. After those values are retrieved, you can denormalize them. Denormalization is done by using the following formula:

$$f(x) = \left((D_L - D_H)^* x - N_H{}^* D_L + D_H{}^* N_L \right) / (N_L - N_H)$$

where:

x: Input data point

D_L: Min (lowest) value of x in the input data set

D_H: Max (highest) value of x in the input data set

N_L: The left part of the normalized interval [-1, 1]

N_H: The right part of the normalized interval [-1, 1]

```
denormInputXPointValue = ((minXPointDl - maxXPointDh)*normInputXPointValue -
Nh*minXPointDl + maxXPointDh *Nl)/(Nl - Nh);

denormTargetXPointValue = ((minTargetValueDl - maxTargetValueDh)*normTarget
XPointValue – Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

denormPredictXPointValue =((minTargetValueDl - maxTargetValueDh)*
normPredictXPointValue - Nh*minTargetValueDl +  maxTargetValueDh*Nl)/(Nl - Nh);
```

You also calculate the error percent as the difference percent between the `denormTargetXPointValue` and `denormPredictXPointValue` fields. You can print the results, and you can also populate the values `denormTargetXPointValue` and `denormPredictXPointValue` as the graph element for the currently processed record `xPointer`.

```
xData.add(denormInputXPointValue);
yData1.add(denormTargetXPointValue);
yData2.add(denormPredictXPointValue);

}   // End for pair loop // End for the pair loop
```

Now save the chart file on disk and also calculate the average and maximum percent difference between the actual and predict values for all processed records. After exiting the pair loop, you can add some instructions needed by the chart to print the chart series and save the chart file on disk.

```
XYSeries series1 = Chart.addSeries("Actual data", xData, yData1);
XYSeries series2 = Chart.addSeries("Predict data", xData, yData2);

series1.setLineColor(XChartSeriesColors.BLUE);
series2.setMarkerColor(Color.ORANGE);
series1.setLineStyle(SeriesLines.SOLID);
series2.setLineStyle(SeriesLines.SOLID);

try
  {
     //Save the chart image
     BitmapEncoder.saveBitmapWithDPI(Chart, chartTrainFileName,
     BitmapFormat.JPG, 100);
     System.out.println ("Train Chart file has been saved") ;
  }
catch (IOException ex)
  {
     ex.printStackTrace();
     System.exit(3);
   }

 // Finally, save this trained network
 EncogDirectoryPersistence.saveObject(new File(networkFileName),network);
 System.out.println ("Train Network has been saved") ;

 averNormDifferencePerc  = sumNormDifferencePerc/4.00;
 System.out.println(" ");

  System.out.println("maxErrorPerc = " + maxNormDifferencePerc +
      "averErrorPerc = " + averNormDifferencePerc);
}   // End of the method
```

Debugging and Executing the Program

When the program coding is complete, you can try executing the project, but it seldom works correctly. You will need to debug the program. To set a breakpoint, simply click the program source line number. Figure 5-22 shows the result of clicking line 180. The red line confirms that the breakpoint is set. If you click the same number again, the breakpoint will be removed.

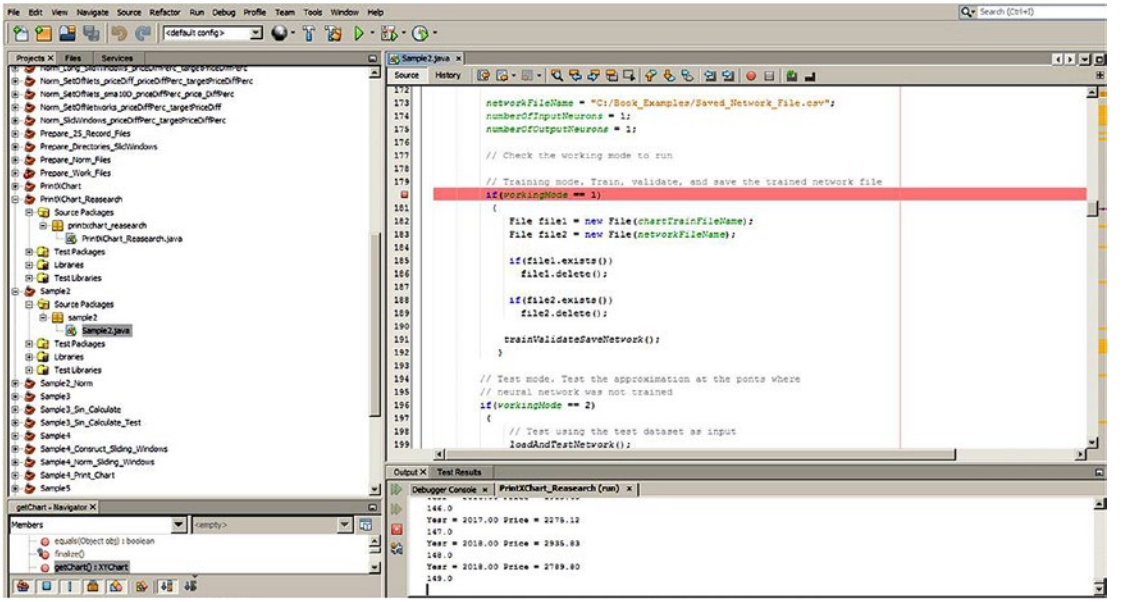


Figure 5-22. *Setting the breakpoint*

Here, you should set a breakpoint at the logic that checks which working mode to run. After setting the breakpoint, select Debug ➤ Debug Project from the main menu. The program starts executing and then stops at the breakpoint. Here, if you move the cursor on top of any variable, its value will be displayed in the pop-up window.

To advance execution of the program, click one of the menu arrow icons, depending on whether you want to advance execution by one line, go inside the executing method, exit the current method, and so on (see Figure 5-23).

Figure 5-23. *Icon for advancing execution while debugging*

To run the program, select Run ➤ Run Project from the menu. The execution results are shown in the log window.

Processing Results for the Training Method

Listing 5-7 shows the training results.

Listing 5-7. Training Processing Results

```
RecordNumber = 0  TargetValue = 0.0224 PredictedValue = 0.022898  DiffPerc = 1.77
RecordNumber = 1  TargetValue = 0.0625 PredictedValue = 0.062009  DiffPerc = 0.79
RecordNumber = 2  TargetValue = 0.25   PredictedValue = 0.250359  DiffPerc = 0.14
RecordNumber = 3  TargetValue = 0.5625 PredictedValue = 0.562112  DiffPerc = 0.07
RecordNumber = 4  TargetValue = 1.0    PredictedValue = 0.999552  DiffPerc = 0.04
RecordNumber = 5  TargetValue = 1.5625 PredictedValue = 1.563148  DiffPerc = 0.04
RecordNumber = 6  TargetValue = 2.25   PredictedValue = 2.249499  DiffPerc = 0.02
RecordNumber = 7  TargetValue = 3.0625 PredictedValue = 3.062648  DiffPerc = 0.00
RecordNumber = 8  TargetValue = 4.0    PredictedValue = 3.999920  DiffPerc = 0.00

maxErrorPerc = 1.769902752691229
 averErrorPerc = 0.2884023848904945
```

The average error difference percent for all records is 0.29 percent, and the max error difference percent for all records is 1.77 percent.

The chart in Figure 5-24 shows the approximation results at nine points where the network was trained.

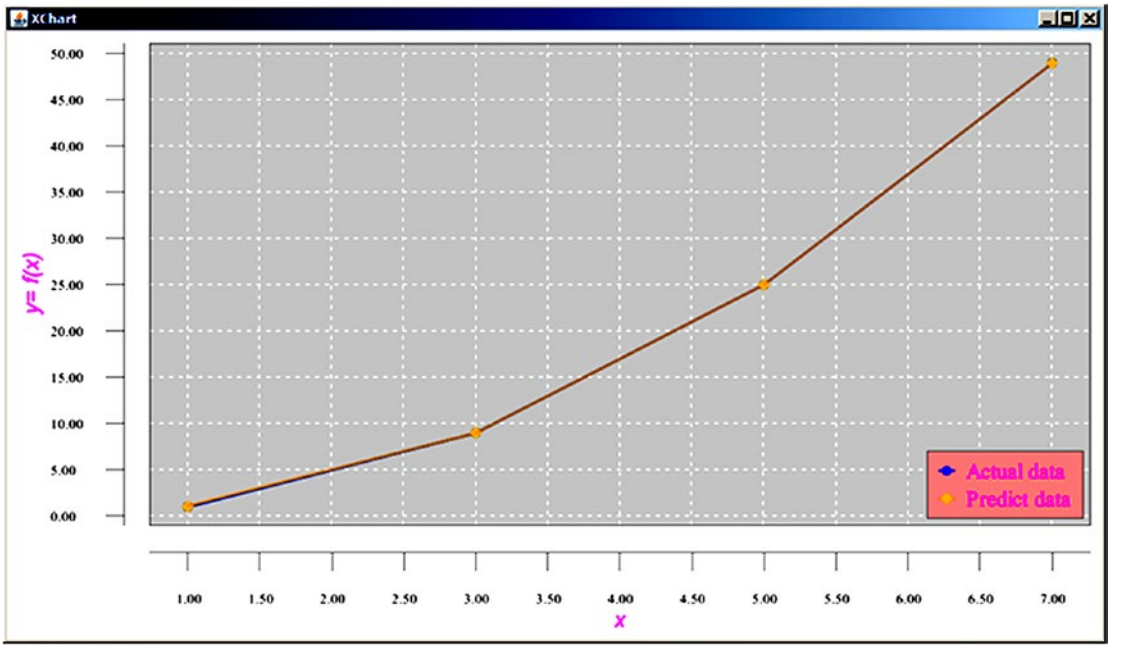


Figure 5-24. *The chart of the training results*

The actual chart and the predicted (approximation) chart are practically overlapping at the points where the network was trained.

Testing the Network

The test data set includes records that were not used during the network training. To test the network, you need to adjust the program configuration statements to execute the program in test mode. To do this, you comment out the configuration statements for the training mode and uncomment the configuration statements for the testing mode (Listing 5-8).

Listing 5-8. Configuration to Run the Program in Test Mode

```
If (workingMode == 1)
           {
            // Config for training the network
            workingMode = 1;
            intNumberOfRecordsInTrainFile = 10;
```

```
            trainFileName = "C:/My_Neural_Network_Book/Book_Examples/
                            Sample2_Train_Norm.csv";
            chartTrainFileName = "Sample2_XYLine_Train_Results_Chart";
          }
        else
          {
            // Config for testing the trained network
            // workingMode = 2;
            // intNumberOfRecordsInTestFile = 10;
            // testFileName = "C:/My_Neural_Network_Book/Book_Examples/
                              Sample2_Test_Norm.csv";
            //  chartTestFileName = "XYLine_Test_Results_Chart";
          }

//-------------------------------------------------------
// Common configuration
//-------------------------------------------------------
networkFileName = "C:/Book_Examples/Saved_Network_File.csv";
numberOfInputNeurons = 1;
numberOfOutputNeurons = 1;
```

The processing logic of the test method is similar to the training method; however, there are some differences. The input file that the method processes is now the testing data set, and the method does not include the network training logic because the network has been already trained and saved on disk during the execution of the training method. Instead, this method loads the previously saved trained network file in memory (Listing 5-9).

You then load the testing data set and the previously saved trained network file in memory.

Listing 5-9. Fragments of the Testing Method

```
// Load the test dataset into memory
MLDataSet testingSet =
loadCSV2Memory(testFileName,numberOfInputNeurons,numberOfOutputNeurons,
       true,CSVFormat.ENGLISH,false);
```

```
 // Load the saved trained network
network =
      (BasicNetwork)EncogDirectoryPersistence.loadObject(new
      File(networkFileName));
```

You iterate over the pair data set and obtain from the network the normalized input and the actual and predicted values for each record. Next, you denormalize those values and calculate the average and maximum difference percents (between the denormalized actual and predicted values). After getting those values, you print them and also populate the chart element for each record. Finally, you add some code for controlling the chart series and save the chart on disk.

```
int i = - 1;
double xPoint = -0.00;

for (MLDataPair pair:  testingSet)
      {
           i++;
           xPoint = xPoint + 2.00;  // The chart accepts only double and
           Date variable types, not integer

           MLData inputData = pair.getInput();
           MLData actualData = pair.getIdeal();
           MLData predictData = network.compute(inputData);

           // These values are Normalized as the whole input is
           normInputXPointValueFromRecord = inputData.getData(0);
           normTargetXPointValueFromRecord = actualData.getData(0);
           normPredictXPointValueFromRecord = predictData.getData(0);

           denormInputXPointValue = ((minXPointDl - maxXPointDh)*
             normInputXPointValueFromRecord - Nh*minXPointDl +
                maxXPointDh*Nl)/(Nl - Nh);
           denormTargetXPointValue = ((minTargetValueDl - maxTargetValueDh)*
             normTargetXPointValueFromRecord - Nh*minTargetValueDl +
                maxTargetValueDh*Nl)/(Nl - Nh);
           denormPredictXPointValue =((minTargetValueDl - maxTargetValueDh)*
             normPredictXPointValueFromRecord - Nh*minTargetValueDl +
                maxTargetValueDh*Nl)/(Nl - Nh);
```

```
      targetToPredictPercent = Math.abs((denormTargetXPointValue -
        denormPredictXPointValue)/denormTargetXPointValue*100);

      System.out.println("xPoint = " + xPoint +
        "  denormTargetXPointValue = " + denormTargetXPointValue +
          "  denormPredictXPointValue = " + denormPredictXPointValue +
            "   targetToPredictPercent = " + targetToPredictPercent);

      if (targetToPredictPercent > maxGlobalResultDiff)
         maxGlobalResultDiff = targetToPredictPercent;

      sumGlobalResultDiff = sumGlobalResultDiff + targetToPredictPercent;

      // Populate chart elements
      xData.add(denormInputXPointValue);
      yData1.add(denormTargetXPointValue);
      yData2.add(denormPredictXPointValue);
  }  // End for pair loop

// Print the max and average results

System.out.println(" ");
averGlobalResultDiff = sumGlobalResultDiff/intNumberOfRecordsInTestFile;

System.out.println("maxErrorPerc = " + maxGlobalResultDiff );
System.out.println("averErrorPerc = " + averGlobalResultDiff);

// All testing batch files have been processed
XYSeries series1 = Chart.addSeries("Actual", xData, yData1);
XYSeries series2 = Chart.addSeries("Predicted", xData, yData2);

series1.setLineColor(XChartSeriesColors.BLUE);
series2.setMarkerColor(Color.ORANGE);
series1.setLineStyle(SeriesLines.SOLID);
series2.setLineStyle(SeriesLines.SOLID);

// Save the chart image
try
  {
    BitmapEncoder.saveBitmapWithDPI(Chart, chartTestFileName,
    BitmapFormat.JPG, 100);
```

```
        }
       catch (Exception bt)
        {
          bt.printStackTrace();
        }

       System.out.println ("The Chart has been saved");

       System.out.println("End of testing for test records");

   } // End of the method
```

Testing Results

Listing 5-10 shows the testing results.

Listing 5-10. Testing Results

```
xPoint = 0.20 TargetValue = 0.04000 PredictedValue = 0.03785 targetToPredictDiffPerc = 5.37
xPoint = 0.30 TargetValue = 0.09000 PredictedValue = 0.09008 targetToPredictDiffPerc = 0.09
xPoint = 0.40 TargetValue = 0.16000 PredictedValue = 0.15798 targetToPredictDiffPerc = 1.26
xPoint = 0.70 TargetValue = 0.49000 PredictedValue = 0.48985 targetToPredictDiffPerc = 0.03
xPoint = 0.95 TargetValue = 0.90250 PredictedValue = 0.90208 targetToPredictDiffPerc = 0.05
xPoint = 1.30 TargetValue = 1.69000 PredictedValue = 1.69096 targetToPredictDiffPerc = 0.06
xPoint = 1.60 TargetValue = 2.56000 PredictedValue = 2.55464 targetToPredictDiffPerc = 0.21
xPoint = 1.80 TargetValue = 3.24000 PredictedValue = 3.25083 targetToPredictDiffPerc = 0.33
xPoint = 1.95 TargetValue = 3.80250 PredictedValue = 3.82933 targetToPredictDiffPerc = 0.71

maxErrorPerc = 5.369910680518282
averErrorPerc = 0.8098656579029523
```

The average error (the percent difference between the actual and predicted values) is 5.37 percent.

The max error (the percent difference between the actual and predicted values) is 0.81 percent. Figure 5-25 shows the chart for the testing results.

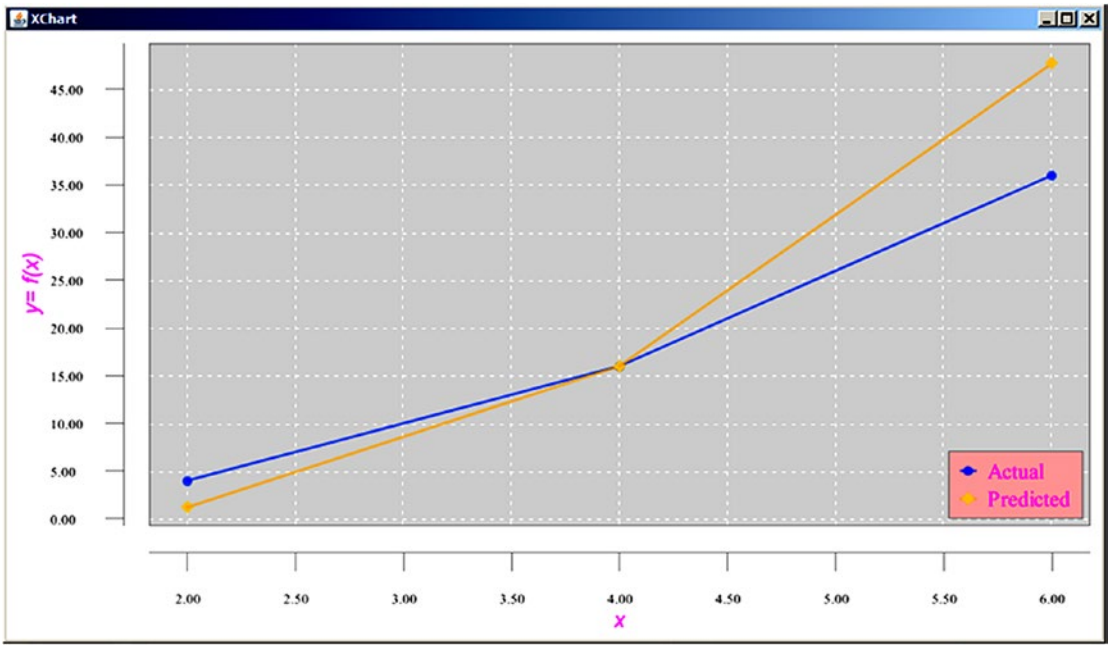


Figure 5-25. *Approximated chart at the points where the network was not trained*

The noticeable discrepancies between the actual and predicted values are because of the rough function approximation. You usually can improve the approximation precision by tinkering with the architecture of the network (number of hidden layers, number of neurons in layers). However, the main problem here is a small number of points and correspondingly relatively large distance between points used to train the network. To get substantially better function approximation results, you can use many more points (with a much smaller difference between them) to approximate this function.

Should the training data set include many more points (100; 1,000; or even 10,000) and correspondingly much smaller distances between points (0.01, 0.001, or even 0.0001), the approximation results would be substantially more precise. However, that is not the goal of this first simple example.

Digging Deeper

Why are so many more points needed for approximating this function? Function approximation controls the behavior of the approximated function at the points processed during training. The network learns to make the approximated results closely match the actual function values at the training points, but it has much less control of the function behavior between the training points. Consider Figure 5-26.

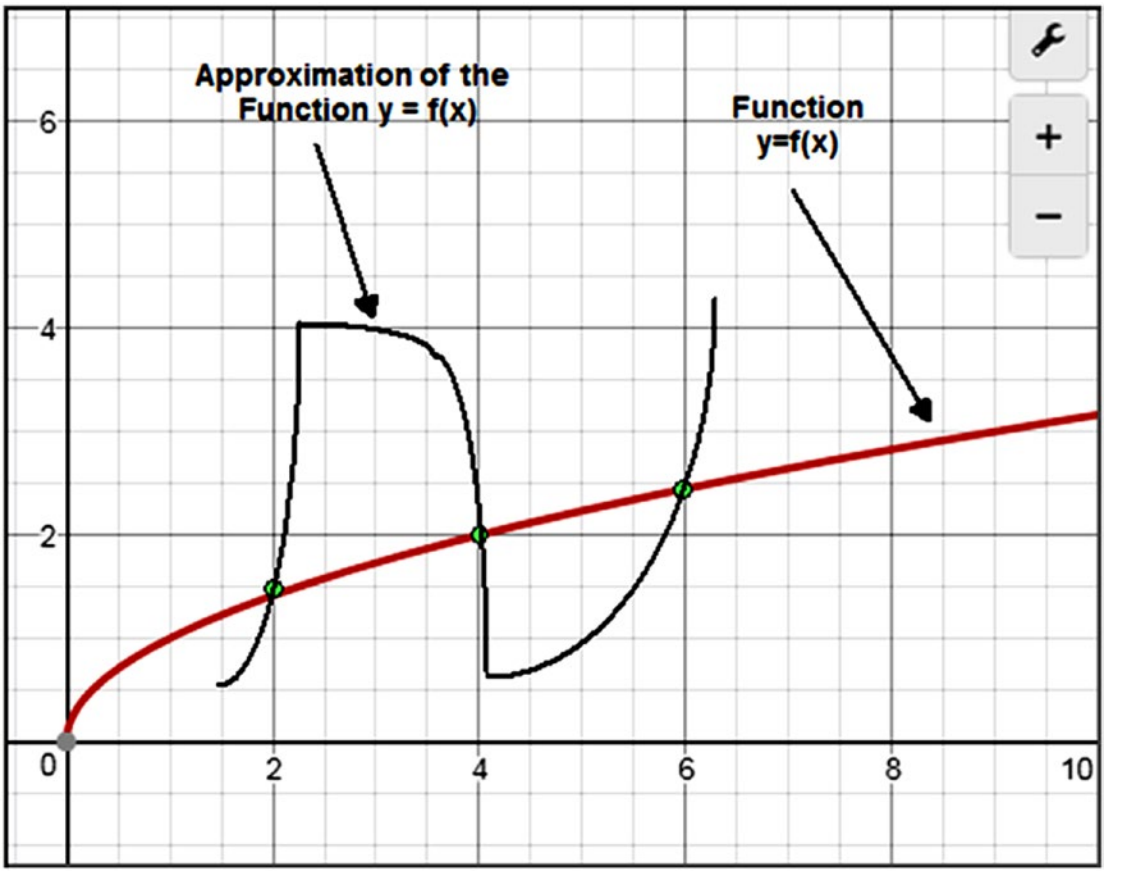


Figure 5-26. *Original and approximated functions*

In Figure 5-26, the approximation function values closely match the original function values at the training points, but not between points. The errors for testing points are deliberately exaggerated to make the point clearer. If many more training points are used, then the testing points will always be much closer to one of the training points, and the testing results at the testing points will be much closer to the original function value at those points.

Summary

This chapter described how to develop neural network applications using the Java Encog framework. You saw a step-by-step approach to coding a neural network application using Encog. All the examples in the rest of this book use the Encog framework.

CHAPTER 6

Neural Network Prediction Outside the Training Range

Preparing data for neural network processing is typically the most difficult and time-consuming task you'll encounter when working with neural networks. In addition to the enormous volume of data that could easily reach millions and even billions of records, the main difficulty is in preparing the data in the correct format for the task in question. In this and the following chapters, I will demonstrate several techniques of data preparations/transformation.

The goal of this chapter's example is to show how to bypass the major restriction for the neural network approximation, which states that predictions should be used only inside the training interval. This restriction exists for any function approximation mechanism (not only for approximation by neural networks but also for any type of approximations such as Taylor series and Newtonian approximation calculus). Getting function values outside of the training interval is called *forecasting* or *extrapolation* (rather than prediction). Forecasting function values is based on extrapolation, while the neural network processing mechanism is based on the approximation mechanism. Getting the function approximation value outside the training interval simply produces the wrong result. This is one of the important concepts to be aware of.

I. Livshin, *Artificial Neural Networks with Java*, https://doi.org/10.1007/978-1-4842-4421-0_6

Example 3a: Approximating Periodic Functions Outside of the Training Range

For this example, you will use the tangent periodic function y = tan(x). Let's pretend that you don't know what type of periodic function is given to you; the function is given to you by its values at certain points. Table 6-1 shows function values on the interval [0, 1.2]. You will use this data for network training.

Table 6-1. *Function Values on the Interval [0, 1.2]*

Point x	y
0	10
0.12	10.12058
0.24	10.24472
0.36	10.3764
0.48	10.52061
0.6	10.68414
0.72	10.87707
0.84	11.11563
0.96	11.42836
1.08	11.87122
1.2	12.57215

Figure 6-1 shows the chart of the function values on the interval [0, 1.2].

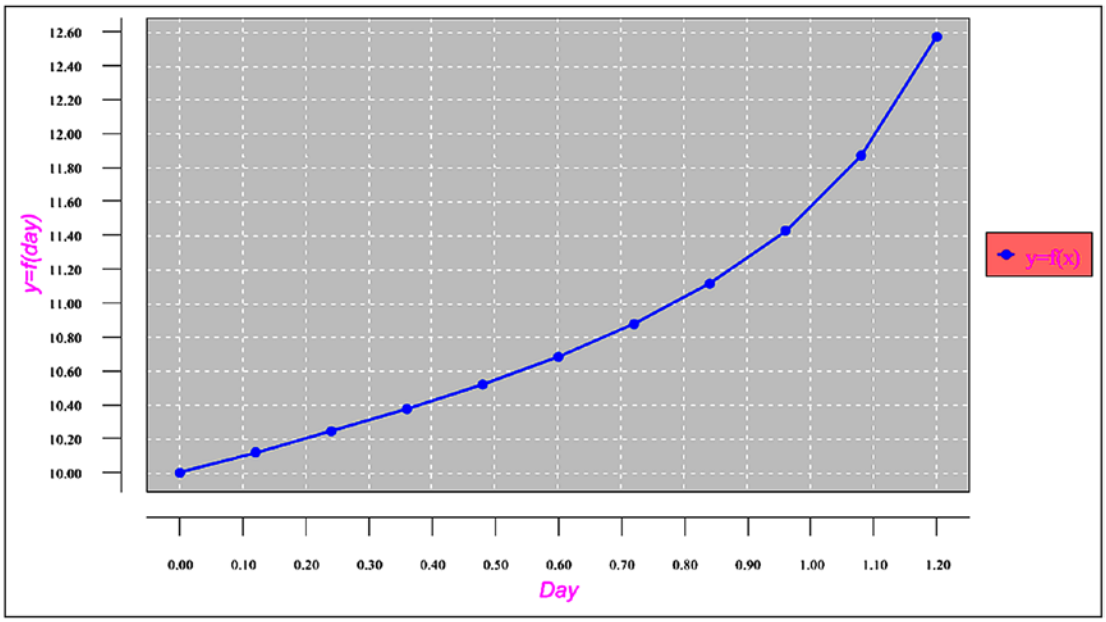


Figure 6-1. *Chart of the function values on the interval [0, 1.2]*

Table 6-2 shows function values on the interval [3.141592654**,** 4.341592654]. You will use this data for testing the trained network.

Table 6-2. *Function Values on the Interval [3.141592654, 4.341592654]*

Point x	y
3.141593	10
3.261593	10.12058
3.381593	10.24472
3.501593	10.3764
3.621593	10.52061
3.741593	10.68414
3.861593	10.87707
3.981593	11.11563
4.101593	11.42836
4.221593	11.87122
4.341593	12.57215

Figure 6-2 shows the chart of the function values on the interval [3.141592654, 4.341592654].

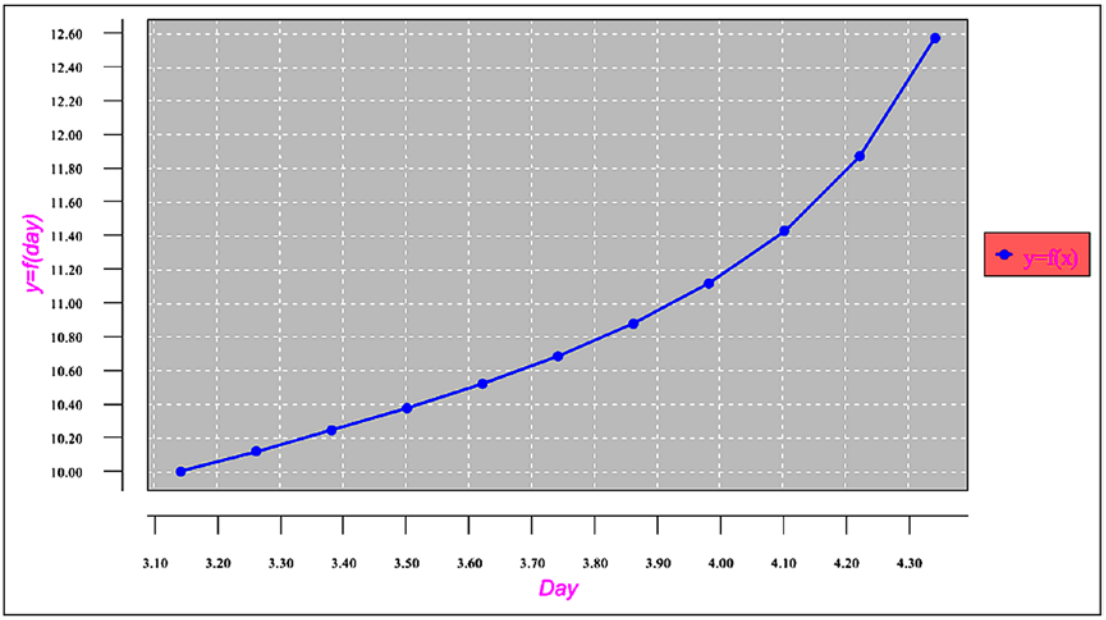


Figure 6-2. *Chart of the function values on the interval [3.141592654, 4.341592654]*

The goal of this example is to approximate the function on the given interval [0, 1.2] and then use the trained network to predict the function values on the next interval, which is [3.141592654, 4.341592654].

For Example 3a, you will try to approximate the function in a conventional way, by using the given data as it is. This data needs to be normalized on the interval [-1, 1]. Table 6-3 shows the normalized training data set.

Table 6-3. *Normalized Training Data Set*

Point x	y
-0.666666667	-0.5
-0.626666667	-0.43971033
-0.586666667	-0.37764165
-0.546666667	-0.311798575
-0.506666667	-0.23969458
-0.466666667	-0.157931595
-0.426666667	-0.06146605
-0.386666667	0.057816175
-0.346666667	0.214178745
-0.306666667	0.43560867
-0.266666667	0.78607581

Table 6-4 shows the normalized testing data set.

Table 6-4. *Normalized Testing Data Set*

Point x	y
0.380530885	-0.5
0.420530885	-0.43971033
0.460530885	-0.37764165
0.500530885	-0.311798575
0.540530885	-0.23969458
0.580530885	-0.157931595
0.620530885	-0.06146605
0.660530885	0.057816175
0.700530885	0.214178745
0.740530885	0.43560867
0.780530885	0.786075815

Network Architecture for Example 3a

Figure 6-3 shows the network architecture for this example. Typically, the architecture for a specific project is determined experimentally, by trying many and selecting the one that produces the best approximation results. The network consists of a single neuron in the input layer, three hidden layers (each with five neurons), and a single neuron in the output layer.

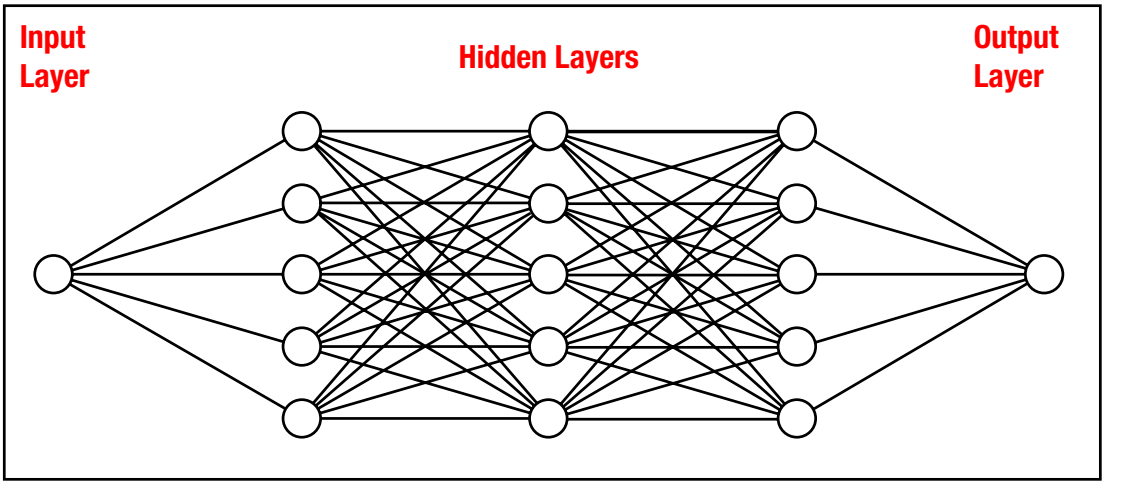


Figure 6-3. *Network architecture*

Program Code for Example 3a

Listing 6-1 shows the program code.

Listing 6-1. Program Code

```
// ====================================================
// Approximation of the periodic function outside of the training range.
//
// The input is the file consisting of records with two fields:
// - The first field is the xPoint value.
// - The second field is the target function value at that xPoint
// ====================================================

package sample3a;
```

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.*;
import java.util.Properties;
import java.time.YearMonth;
import java.awt.Color;
import java.awt.Font;
import java.io.BufferedReader;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.LocalDate;
import java.time.Month;
import java.time.ZoneId;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.List;
import java.util.Locale;
import java.util.Properties;

import org.encog.Encog;
import org.encog.engine.network.activation.ActivationTANH;
import org.encog.engine.network.activation.ActivationReLU;
import org.encog.ml.data.MLData;
import org.encog.ml.data.MLDataPair;
import org.encog.ml.data.MLDataSet;
import org.encog.ml.data.buffer.MemoryDataLoader;
import org.encog.ml.data.buffer.codec.CSVDataCODEC;
```

```
import org.encog.ml.data.buffer.codec.DataSetCODEC;
import org.encog.neural.networks.BasicNetwork;
import org.encog.neural.networks.layers.BasicLayer;
import org.encog.neural.networks.training.propagation.resilient.
ResilientPropagation;
import org.encog.persist.EncogDirectoryPersistence;
import org.encog.util.csv.CSVFormat;

import org.knowm.xchart.SwingWrapper;
import org.knowm.xchart.XYChart;
import org.knowm.xchart.XYChartBuilder;
import org.knowm.xchart.XYSeries;
import org.knowm.xchart.demo.charts.ExampleChart;
import org.knowm.xchart.style.Styler.LegendPosition;
import org.knowm.xchart.style.colors.ChartColor;
import org.knowm.xchart.style.colors.XChartSeriesColors;
import org.knowm.xchart.style.lines.SeriesLines;
import org.knowm.xchart.style.markers.SeriesMarkers;
import org.knowm.xchart.BitmapEncoder;
import org.knowm.xchart.BitmapEncoder.BitmapFormat;
import org.knowm.xchart.QuickChart;
import org.knowm.xchart.SwingWrapper;

public class Sample3a implements ExampleChart<XYChart>
{

    static double Nh =  1;
    static double Nl = -1;

   // First column
   static double maxXPointDh = 5.00;
   static double minXPointDl = -1.00;

   // Second column - target data
   static double maxTargetValueDh = 13.00;
   static double minTargetValueDl = 9.00;
```

```
static double doublePointNumber = 0.00;
static int intPointNumber = 0;
static InputStream input = null;
static double[] arrFunctionValue = new double[500];
static double inputDiffValue = 0.00;
static double predictDiffValue = 0.00;
static double targetDiffValue = 0.00;
static double valueDifferencePerc = 0.00;
 static String strFunctionValuesFileName;
static int returnCode  = 0;
static int numberOfInputNeurons;
static int numberOfOutputNeurons;
static int numberOfRecordsInFile;
static int intNumberOfRecordsInTestFile;
static double realTargetValue               ;
static double realPredictValue         ;
static String functionValuesTrainFileName;
static String functionValuesTestFileName;
static String trainFileName;
static String priceFileName;
static String testFileName;
static String chartTrainFileName;
static String chartTestFileName;
static String networkFileName;
static int workingMode;
static String cvsSplitBy = ",";
static double denormTargetDiffPerc;
static double denormPredictDiffPerc;

static List<Double> xData = new ArrayList<Double>();
static List<Double> yData1 = new ArrayList<Double>();
static List<Double> yData2 = new ArrayList<Double>();

static XYChart Chart;
```

```
  @Override
  public XYChart getChart()
   {

    // Create Chart

    Chart = new  XYChartBuilder().width(900).height(500).title(getClass().
              getSimpleName()).xAxisTitle("x").yAxisTitle("y= f(x)").build();

    // Customize Chart
    Chart.getStyler().setPlotBackgroundColor(ChartColor.
    getAWTColor(ChartColor.GREY));
    Chart.getStyler().setPlotGridLinesColor(new Color(255, 255, 255));
    Chart.getStyler().setChartBackgroundColor(Color.WHITE);
    Chart.getStyler().setLegendBackgroundColor(Color.PINK);
    Chart.getStyler().setChartFontColor(Color.MAGENTA);
    Chart.getStyler().setChartTitleBoxBackgroundColor(new Color(0, 222, 0));
    Chart.getStyler().setChartTitleBoxVisible(true);
    Chart.getStyler().setChartTitleBoxBorderColor(Color.BLACK);
    Chart.getStyler().setPlotGridLinesVisible(true);
    Chart.getStyler().setAxisTickPadding(20);
    Chart.getStyler().setAxisTickMarkLength(15);
    Chart.getStyler().setPlotMargin(20);
    Chart.getStyler().setChartTitleVisible(false);
    Chart.getStyler().setChartTitleFont(new Font(Font.MONOSPACED, Font.BOLD, 24));
    Chart.getStyler().setLegendFont(new Font(Font.SERIF, Font.PLAIN, 18));
    Chart.getStyler().setLegendPosition(LegendPosition.InsideSE);
    Chart.getStyler().setLegendSeriesLineLength(12);
    Chart.getStyler().setAxisTitleFont(new Font(Font.SANS_SERIF, Font.ITALIC, 18));
    Chart.getStyler().setAxisTickLabelsFont(new Font(Font.SERIF, Font.
    PLAIN, 11));
    Chart.getStyler().setDatePattern("yyyy-MM");
    Chart.getStyler().setDecimalPattern("#0.00");

    // Configuration
```

```
 // Train
 workingMode = 1;
 trainFileName = "C:/My_Neural_Network_Book/Book_Examples/Sample3a_Norm_
 Tan_Train.csv";
 unctionValuesTrainFileName =
    "C:/My_Neural_Network_Book/Book_Examples/Sample3a_Tan_Calculate_
    Train.csv";
chartTrainFileName =
    "C:/My_Neural_Network_Book/Book_Examples/Sample3a_XYLine_Tan_Train_Chart";
numberOfRecordsInFile = 12;

 // Test the trained network at non-trained points
// workingMode = 2;
// testFileName = "C:/My_Neural_Network_Book/Book_Examples/Sample3a_
   Norm_Tan_Test.csv";
// functionValuesTestFileName =
      "C:/My_Neural_Network_Book/Book_Examples/Sample3a_Tan_Calculate_
      Test.csv";
 //chartTestFileName =
   "C:/My_Neural_Network_Book/Book_Examples/Sample3a_XYLine_Tan_Test_Chart";
 //numberOfRecordsInFile = 12;

 // Common configuration
 networkFileName =
    "C:/My_Neural_Network_Book/Book_Examples/Sample3a_Saved_Tan_Network_
    File.csv";
 numberOfInputNeurons = 1;
 numberOfOutputNeurons = 1;

 try
  {
      // Check the working mode to run

      if(workingMode == 1)
       {
         // Train mode
         loadFunctionValueTrainFileInMemory();
```

```
            File file1 = new File(chartTrainFileName);
            File file2 = new File(networkFileName);

            if(file1.exists())
              file1.delete();

            if(file2.exists())
              file2.delete();

            returnCode = 0;    // Clear the return code variable

            do
             {
               returnCode = trainValidateSaveNetwork();

             } while (returnCode > 0);

         }   // End the train logic
        else
         {
           // Testing mode.

           // Load testing file in memory
           loadTestFileInMemory();

           File file1 = new File(chartTestFileName);


           if(file1.exists())
             file1.delete();

           loadAndTestNetwork();

         }
      }
     catch (Throwable t)
      {
             t.printStackTrace();
            System.exit(1);
       }
```

```
    finally
     {
         Encog.getInstance().shutdown();
     }

 Encog.getInstance().shutdown();

 return Chart;

}  // End of the method

// ==============================
// Load CSV to memory.
// @return The loaded dataset.
// ==============================
public static MLDataSet loadCSV2Memory(String filename, int input, int
ideal, boolean headers, CSVFormat format, boolean significance)
  {
     DataSetCODEC codec = new CSVDataCODEC(new File(filename), format,
     headers, input, ideal, significance);

     MemoryDataLoader load = new MemoryDataLoader(codec);
     MLDataSet dataset = load.external2Memory();
     return dataset;
  }

// ============================================
//  The main method.
//  @param Command line arguments. No arguments are used.
// ============================================
public static void main(String[] args)
 {
   ExampleChart<XYChart> exampleChart = new Sample3a();
   XYChart Chart = exampleChart.getChart();
   new SwingWrapper<XYChart>(Chart).displayChart();
 } // End of the main method
```

```
//=====================================
// Train, validate, and save the trained network file
//=====================================
static public int trainValidateSaveNetwork()
 {
   double functionValue = 0.00;

   // Load the training CSV file in memory
   MLDataSet trainingSet =
    loadCSV2Memory(trainFileName,numberOfInputNeurons,numberOfOutputNeurons,
       true,CSVFormat.ENGLISH,false);

   // create a neural network
   BasicNetwork network = new BasicNetwork();

   // Input layer
   network.addLayer(new BasicLayer(null,true,1));

   // Hidden layer
   network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
   network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
   network.addLayer(new BasicLayer(new ActivationTANH(),true,5));

   // Output layer
   network.addLayer(new BasicLayer(new ActivationTANH(),false,1));

   network.getStructure().finalizeStructure();
   network.reset();

   // train the neural network
   final ResilientPropagation train = new ResilientPropagation(network,
   trainingSet);

   int epoch = 1;
   returnCode = 0;

   do
    {
        train.iteration();
        System.out.println("Epoch #" + epoch + " Error:" + train.getError());
```

```
    epoch++;

    if (epoch >= 500 && network.calculateError(trainingSet) > 0.000000061)
       {
        returnCode = 1;

        System.out.println("Try again");
        return returnCode;
       }

 } while(train.getError() > 0.00000006);

// Save the network file
EncogDirectoryPersistence.saveObject(new File(networkFileName),network);

System.out.println("Neural Network Results:");

double sumDifferencePerc = 0.00;
double averNormDifferencePerc = 0.00;
double maxErrorPerc = 0.00;

int m = -1;
double xPoint_Initial = 0.00;
double xPoint_Increment = 0.12;
double xPoint = xPoint_Initial - xPoint_Increment;

realTargetValue = 0.00;
realPredictValue = 0.00;

for(MLDataPair pair: trainingSet)
  {
      m++;
      xPoint = xPoint + xPoint_Increment;

      //if(xPoint >  3.14)
      //   break;

       final MLData output = network.compute(pair.getInput());

       MLData inputData = pair.getInput();
       MLData actualData = pair.getIdeal();
       MLData predictData = network.compute(inputData);
```

```
        // Calculate and print the results
        inputDiffValue = inputData.getData(0);
        targetDiffValue = actualData.getData(0);
        predictDiffValue = predictData.getData(0);

        //De-normalize the values
        denormTargetDiffPerc = ((minTargetValueDl - maxTargetValueDh)*
        targetDiffValue - Nh*minTargetValueDl + maxTargetValueDh*Nl)/
        (Nl - Nh);
        denormPredictDiffPerc =((minTargetValueDl - maxTargetValueDh)*
        predictDiffValue - Nh*minTargetValueDl + maxTargetValueDh*Nl)/
        (Nl - Nh);

        valueDifferencePerc =
          Math.abs(((denormTargetDiffPerc - denormPredictDiffPerc)/
          denormTargetDiffPerc)*100.00);

        System.out.println ("xPoint = " + xPoint + " realTargetValue = " +
          denormTargetDiffPerc + "  realPredictValue = " +
          denormPredictDiffPerc + " valueDifferencePerc = " + value
          DifferencePerc);

        sumDifferencePerc = sumDifferencePerc + valueDifferencePerc;

        if (valueDifferencePerc > maxErrorPerc && m > 0)
          maxErrorPerc = valueDifferencePerc;

        xData.add(xPoint);
        yData1.add(denormTargetDiffPerc);
        yData2.add(denormPredictDiffPerc);

    }   // End for pair loop

    XYSeries series1 = Chart.addSeries("Actual data", xData, yData1);
    XYSeries series2 = Chart.addSeries("Predict data", xData, yData2);

    series1.setLineColor(XChartSeriesColors.BLUE);
    series2.setMarkerColor(Color.ORANGE);
    series1.setLineStyle(SeriesLines.SOLID);
    series2.setLineStyle(SeriesLines.SOLID);
```

```
     try
      {
         //Save the chart image
         BitmapEncoder.saveBitmapWithDPI(Chart, chartTrainFileName,
         BitmapFormat.JPG, 100);
         System.out.println ("Train Chart file has been saved") ;
      }
    catch (IOException ex)
     {
      ex.printStackTrace();
      System.exit(3);
     }

     // Finally, save this trained network
     EncogDirectoryPersistence.saveObject(new File(networkFileName),network);
     System.out.println ("Train Network has been saved") ;

     averNormDifferencePerc  = sumDifferencePerc/numberOfRecordsInFile;

     System.out.println(" ");
     System.out.println("maxErrorPerc = " + maxErrorPerc +
       "  averNormDifferencePerc = " + averNormDifferencePerc);

     returnCode = 0;

     return returnCode;

 }   // End of the method

//=====================================================
// This method load and test the trained network at the points not
// used for training.
//=====================================================
static public void loadAndTestNetwork()
 {
  System.out.println("Testing the networks results");
```

```
List<Double> xData = new ArrayList<Double>();
List<Double> yData1 = new ArrayList<Double>();
List<Double> yData2 = new ArrayList<Double>();

double sumDifferencePerc = 0.00;
double maxErrorPerc = 0.00;
double maxGlobalResultDiff = 0.00;
double averErrorPerc = 0.00;
double sumGlobalResultDiff = 0.00;
double functionValue;

BufferedReader br4;
BasicNetwork network;
int k1 = 0;

// Process test records
maxGlobalResultDiff = 0.00;
averErrorPerc = 0.00;
sumGlobalResultDiff = 0.00;

MLDataSet testingSet =
loadCSV2Memory(testFileName,numberOfInputNeurons,numberOfOutput
Neurons,true,CSVFormat.ENGLISH,false);

// Load the saved trained network
network =
  (BasicNetwork)EncogDirectoryPersistence.loadObject(new File
  (networkFileName));

int i = - 1; // Index of the current record
int m = -1;

double xPoint_Initial = 3.141592654;
double xPoint_Increment = 0.12;
double xPoint = xPoint_Initial - xPoint_Increment;

realTargetValue = 0.00;
realPredictValue = 0.00;
```

```
for (MLDataPair pair:  testingSet)
 {
    m++;
       xPoint = xPoint + xPoint_Increment;

       //if(xPoint >  3.14)
       //   break;

        final MLData output = network.compute(pair.getInput());

        MLData inputData = pair.getInput();
        MLData actualData = pair.getIdeal();
        MLData predictData = network.compute(inputData);

        // Calculate and print the results
        inputDiffValue = inputData.getData(0);
        targetDiffValue = actualData.getData(0);
        predictDiffValue = predictData.getData(0);

        // De-normalize the values
        denormTargetDiffPerc = ((minTargetValueDl - maxTargetValueDh)*
        targetDiffValue - Nh*minTargetValueDl + maxTarget
        ValueDh*Nl)/(Nl - Nh);
        denormPredictDiffPerc =((minTargetValueDl - maxTargetValueDh)*
        predictDiffValue - Nh*minTargetValueDl + maxTargetValue
        Dh*Nl)/(Nl - Nh);

        valueDifferencePerc =
          Math.abs(((denormTargetDiffPerc - denormPredictDiffPerc)/
          denormTargetDiffPerc)*100.00);

        System.out.println ("xPoint = " + xPoint + " realTargetValue = " +
          denormTargetDiffPerc + "  realPredictValue = " +
          denormPredictDiffPerc + "
             valueDifferencePerc = " +
                valueDifferencePerc);

        sumDifferencePerc = sumDifferencePerc + valueDifferencePerc;
```

```
            if (valueDifferencePerc > maxErrorPerc && m > 0)
              maxErrorPerc = valueDifferencePerc;

            xData.add(xPoint);
            yData1.add(denormTargetDiffPerc);
            yData2.add(denormPredictDiffPerc);

      }  // End for pair loop

      // Print max and average results

      System.out.println(" ");
      averErrorPerc = sumDifferencePerc/numberOfRecordsInFile;

      System.out.println("maxErrorPerc = " + maxErrorPerc);
      System.out.println("averErrorPerc = " + averErrorPerc);

      // All testing batch files have been processed
      XYSeries series1 = Chart.addSeries("Actual", xData, yData1);
      XYSeries series2 = Chart.addSeries("Predicted", xData, yData2);

      series1.setLineColor(XChartSeriesColors.BLUE);
      series2.setMarkerColor(Color.ORANGE);
      series1.setLineStyle(SeriesLines.SOLID);
      series2.setLineStyle(SeriesLines.SOLID);

      // Save the chart image
      try
       {
         BitmapEncoder.saveBitmapWithDPI(Chart, chartTestFileName ,
         BitmapFormat.JPG, 100);
       }
      catch (Exception bt)
       {
         bt.printStackTrace();
       }

      System.out.println ("The Chart has been saved");

   } // End of the method
```

```
//=========================================
// Load Training Function Values file in memory
//=========================================
public static void loadFunctionValueTrainFileInMemory()
 {
   BufferedReader br1 = null;

   String line = "";
   String cvsSplitBy = ",";
   double tempYFunctionValue = 0.00;

    try
      {
        br1 = new BufferedReader(new FileReader(functionValuesTrain
              FileName));

       int i = -1;
       int r = -2;

       while ((line = br1.readLine()) != null)
        {
           i++;
           r++;
         // Skip the header line
         if(i > 0)
           {
            // Break the line using comma as separator
            String[] workFields = line.split(cvsSplitBy);

            tempYFunctionValue = Double.parseDouble(workFields[1]);
            arrFunctionValue[r] = tempYFunctionValue;
          }

      }  // end of the while loop

       br1.close();

     }
    catch (IOException ex)
```

```
        {
          ex.printStackTrace();
          System.err.println("Error opening files = " + ex);
          System.exit(1);
        }

     }

   //===================================
   // Load testing Function Values file in memory
   //===================================
   public static void loadTestFileInMemory()
    {
      BufferedReader br1 = null;

      String line = "";
      String cvsSplitBy = ",";
      double tempYFunctionValue = 0.00;

       try
         {
           br1 = new BufferedReader(new FileReader(functionValuesTestFileName));

          int i = -1;
          int r = -2;

          while ((line = br1.readLine()) != null)
           {
             i++;
             r++;

            // Skip the header line
            if(i > 0)
              {
                // Break the line using comma as separator
                String[] workFields = line.split(cvsSplitBy);

                tempYFunctionValue = Double.parseDouble(workFields[1]);
                arrFunctionValue[r] = tempYFunctionValue;
```

```
            }
         }  // end of the while loop
          br1.close();
        }
       catch (IOException ex)
        {
          ex.printStackTrace();
          System.err.println("Error opening files = " + ex);
          System.exit(1);
        }
    }
} // End of the class
```

This code represents regular neural network processing and does not need any explanation.

Listing 6-2 shows the training processing results.

Listing 6-2. Training Processing Results

```
xPoint = 0.00 TargetValue = 10.00000  PredictedValue = 10.00027 DiffPerc = 0.00274
xPoint = 0.12 TargetValue = 10.12058  PredictedValue = 10.12024 DiffPerc = 0.00336
xPoint = 0.24 TargetValue = 10.24471  PredictedValue = 10.24412 DiffPerc = 0.00580
xPoint = 0.36 TargetValue = 10.37640  PredictedValue = 10.37629 DiffPerc = 0.00102
xPoint = 0.48 TargetValue = 10.52061  PredictedValue = 10.52129 DiffPerc = 0.00651
xPoint = 0.60 TargetValue = 10.68414  PredictedValue = 10.68470 DiffPerc = 0.00530
xPoint = 0.72 TargetValue = 10.87707  PredictedValue = 10.87656 DiffPerc = 0.00467
xPoint = 0.84 TargetValue = 11.11563  PredictedValue = 11.11586 DiffPerc = 0.00209
xPoint = 0.96 TargetValue = 11.42835  PredictedValue = 11.42754 DiffPerc = 0.00712
xPoint = 1.08 TargetValue = 11.87121  PredictedValue = 11.87134 DiffPerc = 0.00104
xPoint = 1.20 TargetValue = 12.57215  PredictedValue = 12.57200 DiffPerc = 0.00119

maxErrorPerc = 0.007121086942321541
averErrorPerc = 0.0034047471040211954
```

Figure 6-4 shows the chart of the training results.

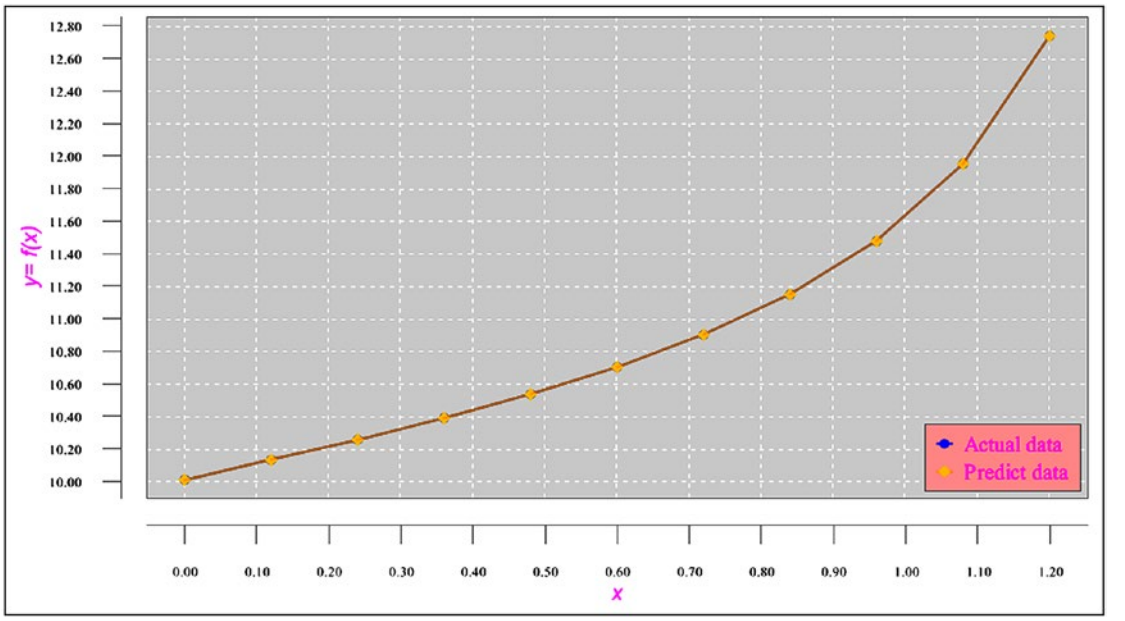


Figure 6-4. *Chart of the training results on the interval [0, 1.2]*

Testing the Network

While processing the test data set, you extract the xPoint value (column 1) from the record, feed this value to the trained network, obtain from the network the predicted function value, and compare the results against the function values that you happen to know (see Listing 6-2, column 2).

Listing 6-3 shows the test processing results.

Listing 6-3. Test Processing Results

```
xPoint = 3.141594 TargetValue = 10.00000 PredictedValue = 12.71432 DiffPerc = 27.14318
xPoint = 3.261593 TargetValue = 10.12059 PredictedValue = 12.71777 DiffPerc = 25.66249
xPoint = 3.381593 TargetValue = 10.24471 PredictedValue = 12.72100 DiffPerc = 24.17133
xPoint = 3.501593 TargetValue = 10.37640 PredictedValue = 12.72392 DiffPerc = 22.62360
xPoint = 3.621593 TargetValue = 10.52061 PredictedValue = 12.72644 DiffPerc = 20.96674
xPoint = 3.741593 TargetValue = 10.68413 PredictedValue = 12.72849 DiffPerc = 19.13451
xPoint = 3.861593 TargetValue = 10.87706 PredictedValue = 12.73003 DiffPerc = 17.03549
xPoint = 3.981593 TargetValue = 11.11563 PredictedValue = 12.73102 DiffPerc = 14.53260
xPoint = 4.101593 TargetValue = 11.42835 PredictedValue = 12.73147 DiffPerc = 11.40249
```

```
xPoint = 4.221593 TargetValue = 11.87121 PredictedValue = 12.73141 DiffPerc = 7.246064
xPoint = 4.341593 TargetValue = 12.57215 PredictedValue = 12.73088 DiffPerc = 1.262565

maxErrorPerc = 25.662489243649677
averErrorPerc = 15.931756451553364
```

Figure 6-5 shows the chart of the test processing results on the interval [3.141592654, 4.341592654].

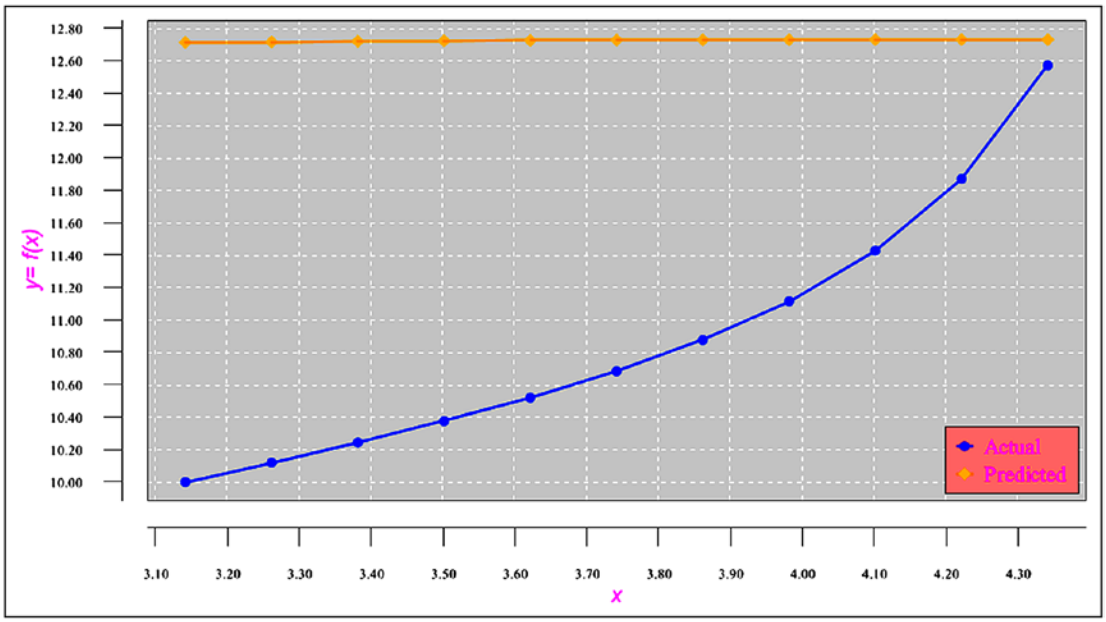


Figure 6-5. *Chart of the testing results on the interval [3.141592654, 4.341592654]*

Notice how different the predicted chart (top line) looks comparing to the actual chart (curved line). The large errors of the test processing results (`maxErrorPerc` = 25.66% and `averErrorPerc` > 15.93%), as shown in Listing 6-7, and the chart in Figure 6-5 show that such function approximation is useless. The network returns those values when it is fed the input xPoints values from the test records that are outside the training range. By sending to the network such xPoints, you can attempt to extrapolate the function values rather than approximate them.

Example 3b: Correct Way of Approximating Periodic Functions Outside the Training Range

In this example, you'll see how (with special data preparation) it is possible for periodic functions to be correctly approximated outside the network training range. As you will see later, you can also use this technique for more complex periodic functions and even some nonperiodic functions.

Preparing the Training Data

As a reminder, this example needs to use the network trained on the interval [0, 1.2] to predict the function results on the interval [3.141592654 – 4.341592654], which is outside the training range. You will see here how to sidestep this neural network restriction for the periodic function. To do this, you will first transform the given function values to a data set with each record consisting of two fields.

- Field 1 is the difference between xPoint values of the current point (record) and the first point (record).
- Field 2 is the difference between the function values at the next point (record) and the current point (record).

Tip When expressing the first field of the record as the difference between xPoint values instead of just the original xPoint values, you are no longer getting outside of the training interval even when you try to predict the function values for any next interval (in this case, [3.141592654 – 4.341592654]). In other words, the difference between xPoint values on the next interval, which is [3.141592654 – 4.341592654], becomes within the training range.

By constructing the input data set in such way, you essentially teach the network to learn that when the difference in the function values between the current and first xPoints is equal to some value "a," then the difference in function values between the next and current points must be equal to some value "b." That allows the network to predict the next day's function value by knowing the current day's function value. Table 6-5 shows the transform data set.

Table 6-5. *Transformed Training Data Set*

Point x	y
-0.12	9.879420663
0	10
0.12	10.12057934
0.24	10.2447167
0.36	10.37640285
0.48	10.52061084
0.6	10.68413681
0.72	10.8770679
0.84	11.11563235
0.96	11.42835749
1.08	11.87121734
1.2	12.57215162
1.32	13.90334779

You normalize the training data set on the interval [-1,1]. Table 6-6 shows the results.

Table 6-6. *Normalized Training Data Set*

xDiff	yDiff
-0.968	-0.967073056
-0.776	-0.961380224
-0.584	-0.94930216
-0.392	-0.929267216
-0.2	-0.898358448
-0.008	-0.851310256
0.184	-0.77829688
0.376	-0.659639776

(*continued*)

Table 6-6. (*continued*)

xDiff	yDiff
0.568	-0.45142424
0.76	-0.038505152
0.952	0.969913872

Table 6-7 shows the transformed testing data set.

Table 6-7. *Transformed Testing Data Set*

xPointDiff	yDiff
3.021592654	9.879420663
3.141592654	10
3.261592654	10.12057934
3.381592654	10.2447167
3.501592654	10.37640285
3.621592654	10.52061084
3.741592654	10.68413681
3.861592654	10.8770679
3.981592654	11.11563235
4.101592654	11.42835749
4.221592654	11.87121734
4.341592654	12.57215163
4.461592654	13.90334779

Table 6-8 shows the normalized testing data set.

Table 6-8. *Normalized Testing Data Set*

xDiff	yDiff
-0.968	-0.967073056
-0.776	-0.961380224
-0.584	-0.94930216
-0.392	-0.929267216
-0.2	-0.898358448
-0.008	-0.851310256
0.184	-0.77829688
0.376	-0.659639776
0.568	-0.45142424
0.76	-0.038505136
0.952	0.969913856

You actually don't need the second column in the test data set for processing. I just included it in the test data set to be able to compare the predicted values against the actual values programmatically. Feeding the difference between the xPoint values at the current and previous points (Field 1 of the currently processed record) to the trained network, you will get back the predicted difference between the function values at the next point and the current point. Therefore, the predicted function value at the next point is equal to the sum of the target function value at the current point (record) and the network-predicted difference value.

Network Architecture for Example 3b

For this example, you will use a network with the output layer consisting of a single neuron, three hidden layers (each holding five neurons), and the output layer holding a single neuron. Again, I came up with this architecture experimentally (by trying and testing). Figure 6-6 shows the training architecture.

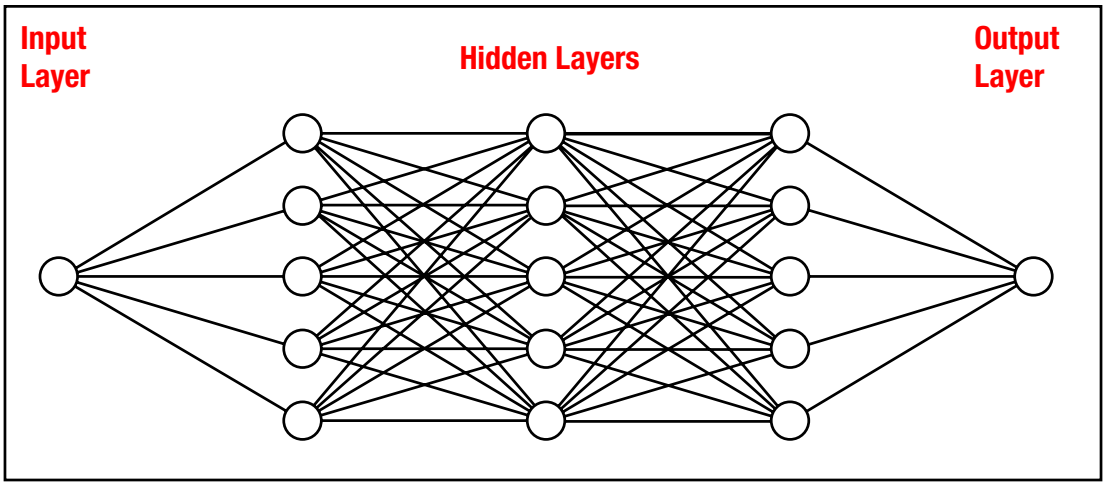


Figure 6-6. *Network architecture for the example*

Now you are ready to develop the network processing program and run the training and testing methods.

Program Code for Example 3b

Listing 6-4 shows the program code.

Listing 6-4. Program Code

```
// ============================================================================
// Approximation of the periodic function outside of the training range.
//
// The input is the file consisting of records with two fields:
// - The first field holds the difference between the function values of the
// current and first records.
// - The second field holds the difference between the function values of the
// next and current records.
// ============================================================================

package sample3b;
import java.io.BufferedReader;
import java.io.File;
```

```
import java.io.FileInputStream;
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.*;
import java.util.Properties;
import java.time.YearMonth;
import java.awt.Color;
import java.awt.Font;
import java.io.BufferedReader;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.LocalDate;
import java.time.Month;
import java.time.ZoneId;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.List;
import java.util.Locale;
import java.util.Properties;

import org.encog.Encog;
import org.encog.engine.network.activation.ActivationTANH;
import org.encog.engine.network.activation.ActivationReLU;
import org.encog.ml.data.MLData;
import org.encog.ml.data.MLDataPair;
import org.encog.ml.data.MLDataSet;
import org.encog.ml.data.buffer.MemoryDataLoader;
import org.encog.ml.data.buffer.codec.CSVDataCODEC;
import org.encog.ml.data.buffer.codec.DataSetCODEC;
import org.encog.neural.networks.BasicNetwork;
```

```
import org.encog.neural.networks.layers.BasicLayer;
import org.encog.neural.networks.training.propagation.resilient.
ResilientPropagation;
import org.encog.persist.EncogDirectoryPersistence;
import org.encog.util.csv.CSVFormat;

import org.knowm.xchart.SwingWrapper;
import org.knowm.xchart.XYChart;
import org.knowm.xchart.XYChartBuilder;
import org.knowm.xchart.XYSeries;
import org.knowm.xchart.demo.charts.ExampleChart;
import org.knowm.xchart.style.Styler.LegendPosition;
import org.knowm.xchart.style.colors.ChartColor;
import org.knowm.xchart.style.colors.XChartSeriesColors;
import org.knowm.xchart.style.lines.SeriesLines;
import org.knowm.xchart.style.markers.SeriesMarkers;
import org.knowm.xchart.BitmapEncoder;
import org.knowm.xchart.BitmapEncoder.BitmapFormat;
import org.knowm.xchart.QuickChart;
import org.knowm.xchart.SwingWrapper;

/**
 *
 * @author i262666
 */
public class Sample3b implements ExampleChart<XYChart>
{

    static double Nh =  1;
    static double Nl = -1;

   // First column
   static double maxXPointDh = 1.35;
   static double minXPointDl = 0.10;

   // Second column - target data
   static double maxTargetValueDh = 1.35;
   static double minTargetValueDl = 0.10;
```

```
static double doublePointNumber = 0.00;
static int intPointNumber = 0;
static InputStream input = null;
static double[] arrFunctionValue = new double[500];
static double inputDiffValue = 0.00;
static double predictDiffValue = 0.00;
static double targetDiffValue = 0.00;
static double valueDifferencePerc = 0.00;
 static String strFunctionValuesFileName;
static int returnCode  = 0;
static int numberOfInputNeurons;
static int numberOfOutputNeurons;
static int numberOfRecordsInFile;
static int intNumberOfRecordsInTestFile;
static double realTargetValue               ;
static double realPredictValue         ;
static String functionValuesTrainFileName;
static String functionValuesTestFileName;
static String trainFileName;
static String priceFileName;
static String testFileName;
static String chartTrainFileName;
static String chartTestFileName;
static String networkFileName;
static int workingMode;
static String cvsSplitBy = ",";
static double denormTargetDiffPerc;
static double denormPredictDiffPerc;

static List<Double> xData = new ArrayList<Double>();
static List<Double> yData1 = new ArrayList<Double>();
static List<Double> yData2 = new ArrayList<Double>();

static XYChart Chart;
```

```
  @Override
  public XYChart getChart()
   {

    // Create Chart

    Chart = new  XYChartBuilder().width(900).height(500).title(getClass().
              getSimpleName()).xAxisTitle("x").yAxisTitle("y= f(x)").build();

    // Customize Chart
    Chart.getStyler().setPlotBackgroundColor(ChartColor.
    getAWTColor(ChartColor.GREY));
    Chart.getStyler().setPlotGridLinesColor(new Color(255, 255, 255));
    Chart.getStyler().setChartBackgroundColor(Color.WHITE);
    Chart.getStyler().setLegendBackgroundColor(Color.PINK);
    Chart.getStyler().setChartFontColor(Color.MAGENTA);
    Chart.getStyler().setChartTitleBoxBackgroundColor(new Color(0, 222, 0));
    Chart.getStyler().setChartTitleBoxVisible(true);
    Chart.getStyler().setChartTitleBoxBorderColor(Color.BLACK);
    Chart.getStyler().setPlotGridLinesVisible(true);
    Chart.getStyler().setAxisTickPadding(20);
    Chart.getStyler().setAxisTickMarkLength(15);
    Chart.getStyler().setPlotMargin(20);
    Chart.getStyler().setChartTitleVisible(false);
    Chart.getStyler().setChartTitleFont(new Font(Font.MONOSPACED, Font.BOLD, 24));
    Chart.getStyler().setLegendFont(new Font(Font.SERIF, Font.PLAIN, 18));
    Chart.getStyler().setLegendPosition(LegendPosition.InsideSE);
    Chart.getStyler().setLegendSeriesLineLength(12);
    Chart.getStyler().setAxisTitleFont(new Font(Font.SANS_SERIF, Font.
    ITALIC, 18));
    Chart.getStyler().setAxisTickLabelsFont(new Font(Font.SERIF, Font.
    PLAIN, 11));
    Chart.getStyler().setDatePattern("yyyy-MM");
    Chart.getStyler().setDecimalPattern("#0.00");

    // Configuration
```

```
  // Set the mode of program run
  workingMode = 1;  // Training mode

  if (workingMode ==  1)
    {
      trainFileName = "C:/My_Neural_Network_Book/Book_Examples/Sample3b_
      Norm_Tan_Train.csv";
      functionValuesTrainFileName =
         "C:/My_Neural_Network_Book/Book_Examples/Sample3b_Tan_Calculate_
         Train.csv";
      chartTrainFileName =
         "C:/My_Neural_Network_Book/Book_Examples/Sample3b_XYLine_Tan_
         Train_Chart";
       numberOfRecordsInFile = 12;
   }
else
  {
    // Testing  mode
            testFileName = "C:/My_Neural_Network_Book/Book_Examples/
                           Sample3b_Norm_Tan_Test.csv";
       functionValuesTestFileName =
         "C:/My_Neural_Network_Book/Book_Examples/Sample3b_Tan_Calculate_
         Test.csv";
       chartTestFileName =
        "C:/My_Neural_Network_Book/Book_Examples/Sample3b_XYLine_Tan_
        Test_Chart";
       numberOfRecordsInFile = 12;
     }

  // Common configuration
  networkFileName =
      "C:/My_Neural_Network_Book/Book_Examples/Sample3b_Saved_Tan_
      Network_File.csv";
  numberOfInputNeurons = 1;
  numberOfOutputNeurons = 1;
```

```
    try
     {
         // Check the working mode to run

         if(workingMode == 1)
          {
            // Train mode
            loadFunctionValueTrainFileInMemory();

             File file1 = new File(chartTrainFileName);
             File file2 = new File(networkFileName);

             if(file1.exists())
               file1.delete();

             if(file2.exists())
               file2.delete();

             returnCode = 0;    // Clear the return code variable

             do
              {
                returnCode = trainValidateSaveNetwork();

              } while (returnCode > 0);

          }   // End the train logic
         else
          {
            // Testing mode.

            // Load testing file in memory
            loadTestFileInMemory();

            File file1 = new File(chartTestFileName);

            if(file1.exists())
              file1.delete();

            loadAndTestNetwork();

          }
```

```
    }
   catch (Throwable t)
     {
       t.printStackTrace();
       System.exit(1);
     }
    finally
     {
       Encog.getInstance().shutdown();
     }

 Encog.getInstance().shutdown();

 return Chart;

}  // End of the method
// =========================================================
// Load CSV to memory.
// @return The loaded dataset.
// =========================================================
public static MLDataSet loadCSV2Memory(String filename, int input, int
ideal, boolean headers, CSVFormat format, boolean significance)
  {
     DataSetCODEC codec = new CSVDataCODEC(new File(filename), format,
     headers, input, ideal, significance);
     MemoryDataLoader load = new MemoryDataLoader(codec);
     MLDataSet dataset = load.external2Memory();
     return dataset;
  }

// =========================================================
//  The main method.
//  @param Command line arguments. No arguments are used.
// =========================================================
```

```
public static void main(String[] args)
 {
   ExampleChart<XYChart> exampleChart = new Sample3b();
   XYChart Chart = exampleChart.getChart();
   new SwingWrapper<XYChart>(Chart).displayChart();
 } // End of the main method

//=======================================
// Train, validate, and saves the trained network file
//=======================================
static public int trainValidateSaveNetwork()
 {
   double functionValue = 0.00;

   // Load the training CSV file in memory
   MLDataSet trainingSet =
     loadCSV2Memory(trainFileName,numberOfInputNeurons,numberOfOutputNeurons,
       true,CSVFormat.ENGLISH,false);

   // create a neural network
   BasicNetwork network = new BasicNetwork();

   // Input layer
   network.addLayer(new BasicLayer(null,true,1));

   // Hidden layer
   network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
   network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
   network.addLayer(new BasicLayer(new ActivationTANH(),true,5));

   // Output layer
   network.addLayer(new BasicLayer(new ActivationTANH(),false,1));

   network.getStructure().finalizeStructure();
   network.reset();

   // train the neural network
   final ResilientPropagation train = new ResilientPropagation(network,
   trainingSet);
```

```
int epoch = 1;
returnCode = 0;

do
 {
     train.iteration();
     System.out.println("Epoch #" + epoch + " Error:" + train.getError());

    epoch++;

    if (epoch >= 500 && network.calculateError(trainingSet) > 0.000000061)
       {
        returnCode = 1;

        System.out.println("Try again");
        return returnCode;
       }

 } while(train.getError() > 0.00000006);

// Save the network file
EncogDirectoryPersistence.saveObject(new File(networkFileName),network);

System.out.println("Neural Network Results:");

double sumDifferencePerc = 0.00;
double averNormDifferencePerc = 0.00;
double maxErrorPerc = 0.00;

int m = -1;
double xPoint_Initial = 0.00;
double xPoint_Increment = 0.12;
//double xPoint = xPoint_Initial - xPoint_Increment;
double xPoint = xPoint_Initial;

realTargetValue = 0.00;
realPredictValue = 0.00;

for(MLDataPair pair: trainingSet)
  {
      m++;
```

```
xPoint = xPoint + xPoint_Increment;

final MLData output = network.compute(pair.getInput());

 MLData inputData = pair.getInput();
 MLData actualData = pair.getIdeal();
 MLData predictData = network.compute(inputData);

 // Calculate and print the results
 inputDiffValue = inputData.getData(0);
 targetDiffValue = actualData.getData(0);
 predictDiffValue = predictData.getData(0);

 // De-normalize the values
 denormTargetDiffPerc = ((minXPointDl -
 maxXPointDh)*targetDiffValue - Nh*minXPointDl +
      maxXPointDh*Nl)/(Nl - Nh);
 denormPredictDiffPerc =((minTargetValueDl - maxTargetValueDh)*
 predictDiffValue - Nh*minTargetValueDl + maxTarget
 ValueDh*Nl)/(Nl - Nh);

 functionValue = arrFunctionValue[m+1];

 realTargetValue = functionValue + denormTargetDiffPerc;
 realPredictValue = functionValue + denormPredictDiffPerc;

 valueDifferencePerc =
   Math.abs(((realTargetValue - realPredictValue)/
   realPredictValue)*100.00);

 System.out.println ("xPoint = " + xPoint + "  realTargetValue = " +
   denormTargetDiffPerc + "  realPredictValue = " +
   denormPredictDiffPerc + " valueDifferencePerc = “ +  value
   DifferencePerc);

 sumDifferencePerc = sumDifferencePerc + valueDifferencePerc;

 if (valueDifferencePerc > maxErrorPerc && m > 0)
   maxErrorPerc = valueDifferencePerc;
```

```
            xData.add(xPoint);
            yData1.add(denormTargetDiffPerc);
            yData2.add(denormPredictDiffPerc);

        }   // End for pair loop

        XYSeries series1 = Chart.addSeries("Actual data", xData, yData1);
        XYSeries series2 = Chart.addSeries("Predict data", xData, yData2);

        series1.setLineColor(XChartSeriesColors.BLUE);
        series2.setMarkerColor(Color.ORANGE);
        series1.setLineStyle(SeriesLines.SOLID);
        series2.setLineStyle(SeriesLines.SOLID);

        try
         {
           //Save the chart image
           BitmapEncoder.saveBitmapWithDPI(Chart, chartTrainFileName,
           BitmapFormat.JPG, 100);
           System.out.println ("Train Chart file has been saved") ;
         }
       catch (IOException ex)
        {
         ex.printStackTrace();
         System.exit(3);
        }

        // Finally, save this trained network
        EncogDirectoryPersistence.saveObject(new File(networkFileName),network);
        System.out.println ("Train Network has been saved") ;

        averNormDifferencePerc  = sumDifferencePerc/numberOfRecordsInFile;

        System.out.println(" ");
        System.out.println("maxErrorPerc = " + maxErrorPerc +
          "  averNormDifferencePerc = " + averNormDifferencePerc);

        returnCode = 0;

        return returnCode;

    }   // End of the method
```

```
//================================================
// This method load and test the trained network at the points not
// used for training.
//================================================
static public void loadAndTestNetwork()
 {
  System.out.println("Testing the networks results");

  List<Double> xData = new ArrayList<Double>();
  List<Double> yData1 = new ArrayList<Double>();
  List<Double> yData2 = new ArrayList<Double>();

  double sumDifferencePerc = 0.00;
  double maxErrorPerc = 0.00;
  double maxGlobalResultDiff = 0.00;
  double averErrorPerc = 0.00;
  double sumGlobalResultDiff = 0.00;
  double functionValue;

  BufferedReader br4;
  BasicNetwork network;
  int k1 = 0;

  // Process test records
  maxGlobalResultDiff = 0.00;
  averErrorPerc = 0.00;
  sumGlobalResultDiff = 0.00;

  MLDataSet testingSet =
  loadCSV2Memory(testFileName,numberOfInputNeurons,numberOfOutput
  Neurons,true,CSVFormat.ENGLISH,false);

  int i = - 1; // Index of the current record
  int m = -1;

  double xPoint_Initial = 3.141592654;
  double xPoint_Increment = 0.12;
  double xPoint = xPoint_Initial;
```

```
realTargetValue = 0.00;
realPredictValue = 0.00;

for (MLDataPair pair:  testingSet)
 {
   m++;
   xPoint = xPoint + xPoint_Increment;

   final MLData output = network.compute(pair.getInput());

   MLData inputData = pair.getInput();
   MLData actualData = pair.getIdeal();
   MLData predictData = network.compute(inputData);

   // Calculate and print the results
   inputDiffValue = inputData.getData(0);
   targetDiffValue = actualData.getData(0);
   predictDiffValue = predictData.getData(0);

   // De-normalize the values
   denormTargetDiffPerc = ((minXPointDl -
   maxXPointDh)*targetDiffValue - Nh*minXPointDl +
        maxXPointDh*Nl)/(Nl - Nh);
   denormPredictDiffPerc =((minTargetValueDl - maxTargetValueDh)
   *predictDiffValue - Nh*minTargetValueDl + maxTargetValueDh*Nl)/
   (Nl - Nh);

   functionValue = arrFunctionValue[m+1];

   realTargetValue = functionValue + denormTargetDiffPerc;
   realPredictValue = functionValue + denormPredictDiffPerc;

   valueDifferencePerc =
      Math.abs(((realTargetValue - realPredictValue)/realPredictValue)*100.00);

   System.out.println ("xPoint = " + xPoint + "  realTargetValue = " +
   realTargetValue + "  realPredictValue = " + realPredictValue +
   "  valueDifferencePerc = " + valueDifferencePerc);

   sumDifferencePerc = sumDifferencePerc + valueDifferencePerc;
```

```
        if (valueDifferencePerc > maxErrorPerc && m > 0)
          maxErrorPerc = valueDifferencePerc;

        xData.add(xPoint);
        yData1.add(realTargetValue);
        yData2.add(realPredictValue);

      }  // End for pair loop

      // Print max and average results

      System.out.println(" ");
      averErrorPerc = sumDifferencePerc/numberOfRecordsInFile;

      System.out.println("maxErrorPerc = " + maxErrorPerc);
      System.out.println("averErrorPerc = " + averErrorPerc);

      // All testing batch files have been processed
      XYSeries series1 = Chart.addSeries("Actual", xData, yData1);
      XYSeries series2 = Chart.addSeries("Predicted", xData, yData2);

      series1.setLineColor(XChartSeriesColors.BLUE);
      series2.setMarkerColor(Color.ORANGE);
      series1.setLineStyle(SeriesLines.SOLID);
      series2.setLineStyle(SeriesLines.SOLID);

      // Save the chart image
      try
       {
         BitmapEncoder.saveBitmapWithDPI(Chart, chartTestFileName ,
         BitmapFormat.JPG, 100);
       }
      catch (Exception bt)
       {
         bt.printStackTrace();
       }

      System.out.println ("The Chart has been saved");

   } // End of the method
```

```
//=======================================
// Load Training Function Values file in memory
//=======================================
public static void loadFunctionValueTrainFileInMemory()
 {
   BufferedReader br1 = null;

   String line = "";
   String cvsSplitBy = ",";
   double tempYFunctionValue = 0.00;

    try
      {
        br1 = new BufferedReader(new FileReader(functionValuesTrainFileName));

       int i = -1;
       int r = -2;

       while ((line = br1.readLine()) != null)
        {
          i++;
          r++;

         // Skip the header line
         if(i > 0)
           {
            // Break the line using comma as separator
            String[] workFields = line.split(cvsSplitBy);

            tempYFunctionValue = Double.parseDouble(workFields[1]);
            arrFunctionValue[r] = tempYFunctionValue;
           }

      }  // end of the while loop

       br1.close();

     }
    catch (IOException ex)
     {
```

```
            ex.printStackTrace();
            System.err.println("Error opening files = " + ex);
            System.exit(1);
        }
     }

   //====================================
   // Load testing Function Values file in memory
   //====================================
   public static void loadTestFileInMemory()
    {
      BufferedReader br1 = null;

      String line = "";
      String cvsSplitBy = ",";
      double tempYFunctionValue = 0.00;

       try
         {
           br1 = new BufferedReader(new FileReader(functionValuesTestFileName));

          int i = -1;
          int r = -2;

          while ((line = br1.readLine()) != null)
           {
             i++;
             r++;

            // Skip the header line
            if(i > 0)
              {
                // Break the line using comma as separator
                String[] workFields = line.split(cvsSplitBy);

                tempYFunctionValue = Double.parseDouble(workFields[1]);
                arrFunctionValue[r] = tempYFunctionValue;
              }

         }  // end of the while loop
```

```
            br1.close();
          }
        catch (IOException ex)
          {
            ex.printStackTrace();
            System.err.println("Error opening files = " + ex);
            System.exit(1);
          }
      }
} // End of the class
```

As usual, some miscellaneous statements are present at the top of the program. They are required by the XChart package. The program begins with some initialization imports and code required by XCharts (see Listing 6-5).

Listing 6-5. Calling the Training Method in a Loop

```
returnCode = 0;    // Clear the error Code

do
  {
      returnCode = trainValidateSaveNetwork();
  } while (returnCode > 0);
```

This logic calls the training method and then checks for the `returnCode` value. If the `returnCode` field is not zero, the training method is called again in a loop. Each time the method is called, the initial weight/bias parameters are assigned different random values, which helps in selecting their best values when the method is repeatedly called in a loop.

Inside the called method, the logic checks for the error value after 500 iterations. If the network-calculated error is still larger than the error limit, the method exits with a `returnCode` value of 1. And, as you just saw, the method will be called again. Finally, when the calculated error clears the error limit, the method exits with a `returnCode` value of 0 and is no longer called again. You select the error limit value experimentally, making it difficult for the network to clear the error code limit but still making sure that after enough iterations the error will pass the error limit.

The next code fragment (Listing 6-6) shows the beginning of the training method. It loads the training data set in memory and creates the neural network consisting of the input layer (with a single neuron), three hidden layers (each with five neurons), and the output layer (with a single neuron). Then, you train the network using the most efficient `ResilientPropagation` value as the backpropagation method.

Listing 6-6. Loading the Training Data Set and Building and Training the Network

```
// Load the training CSV file in memory
MLDataSet trainingSet =
    loadCSV2Memory(trainFileName,numberOfInputNeurons,numberOfOutputNeurons,
          true,CSVFormat.ENGLISH,false);

// create a neural network
BasicNetwork network = new BasicNetwork();

// Input layer
network.addLayer(new BasicLayer(null,true,1));

// Hidden layer (seven hidden layers are created
network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
network.addLayer(new BasicLayer(new ActivationTANH(),true,5));

// Output layer
network.addLayer(new BasicLayer(new ActivationTANH(),false,1));

network.getStructure().finalizeStructure();
network.reset();

// train the neural network
final ResilientPropagation train = new ResilientPropagation(network,
trainingSet);
```

That follows by the fragment that trains the network. You train the network by looping over epochs. On each iteration you check whether the calculated error is less than the established error limit (in this case, 0.00000006). When the network error becomes less than the error limit, you exit the loop. The network is trained with the required precision, so you save the trained network on disk.

```
int epoch = 1;

returnCode = 0;

do
    {
         train.iteration();
         System.out.println("Epoch #" + epoch + " Error:" + train.getError());

         epoch++;

         if (epoch >= 10000 && network.calculateError(trainingSet) > 0.000000061)
           {
               returnCode = 1;

               System.out.println("Try again");
               return returnCode;
            }

       } while(train.getError()0.00000006);

     // Save the network file
     EncogDirectoryPersistence.saveObject(new File(networkFileName),network);
```

Notice the logic shown in Listing 6-7 that checks whether the network error became less than the error limit.

Listing 6-7. Checking the Network Error

```
if (epoch >= 10000 && network.calculateError(trainingSet) > 0.00000006)
  {
      returnCode = 1;

      System.out.println("Try again");
       return returnCode;
  }
```

This code checks whether after 500 iterations the network error is still not less than the error limit. If that is the case, the `returnCode` value is set to 1, and you exit from the training method, returning to the point where the training method is called in a loop. There, it will call the training method again with a new random set of weights/bias

parameters. Without that code, the looping would continue indefinitely if the calculated network error is unable to clear the error limit with the randomly selected set of the initial weights/bias parameters.

There are two APIs that can check the calculated network error. The results differ slightly depending on which method is used.

- `train.getError()`: The error is calculated before the training is applied.
- `network.CalculateError()`: The error is calculated after the training is applied.

The next code fragment (shown in Listing 6-8) loops over the pair data set. The xPoint in the loop is set to be on the interval [0, 1.2]. For each record it retrieves the input, actual, and predicted values; denormalizes them; and by having the function value calculates the `realTargetValue` and `realPredictValue` values, adding them to the chart data (along with the corresponding xPoint value). It also calculates the maximum and average value difference percent for all records. All this data is printed as the training log. Finally, the trained network and the chart image files are saved on disk. Notice that the return code is set to zero at that point, before you return from the training method, so the method will no longer be called again.

Listing 6-8. Looping Over the Pair Data Set

```
for(MLDataPair pair: trainingSet)
   {
        m++;
        xPoint = xPoint + xPoint_Increment;

        final MLData output = network.compute(pair.getInput());


       MLData inputData = pair.getInput();
       MLData actualData = pair.getIdeal();
       MLData predictData = network.compute(inputData);

        // Calculate and print the results
        inputDiffValue = inputData.getData(0);
        targetDiffValue = actualData.getData(0);
        predictDiffValue = predictData.getData(0);
```

```
    // De-normalize the values
    denormTargetDiffPerc = ((minXPointDl - maxXPointDh)*targetDiffValue -
    Nh*minXPointDl + maxXPointDh*Nl)/(Nl - Nh);

    denormPredictDiffPerc =((minTargetValueDl - maxTargetValueDh)
    *predictDiffValue - Nh*minTargetValueDl + maxTargetValueDh*Nl)/
     (Nl - Nh);

    functionValue = arrFunctionValue[m];

    realTargetValue = functionValue + targetDiffValue;
    realPredictValue = functionValue + predictDiffValue;

    valueDifferencePerc =
       Math.abs(((realTargetValue - realPredictValue)/
       realPredictValue)*100.00);

    System.out.println ("xPoint = " + xPoint + "  realTargetValue = " +
          realTargetValue + "  realPredictValue = " + realPredictValue);

   sumDifferencePerc = sumDifferencePerc + valueDifferencePerc;

   if (valueDifferencePerc > maxDifferencePerc)
           maxDifferencePerc = valueDifferencePerc;

    xData.add(xPoint);
    yData1.add(realTargetValue);
    yData2.add(realPredictValue);

}   // End for pair loop

 XYSeries series1 = Chart.addSeries("Actual data", xData, yData1);
 XYSeries series2 = Chart.addSeries("Predict data", xData, yData2);

 series1.setLineColor(XChartSeriesColors.BLUE);
 series2.setMarkerColor(Color.ORANGE);
 series1.setLineStyle(SeriesLines.SOLID);
 series2.setLineStyle(SeriesLines.SOLID);
```

```
    try
     {
         //Save the chart image
        BitmapEncoder.saveBitmapWithDPI(Chart, chartTrainFileName,
        BitmapFormat.JPG, 100);
        System.out.println ("Train Chart file has been saved") ;
     }
   catch (IOException ex)
     {
          ex.printStackTrace();
          System.exit(3);
      }

    // Finally, save this trained network
    EncogDirectoryPersistence.saveObject(new File(networkFileName),network);
    System.out.println ("Train Network has been saved") ;

    averNormDifferencePerc  = sumDifferencePerc/numberOfRecordsInFile;

   System.out.println(" ");
   System.out.println("maxDifferencePerc = " + maxDifferencePerc +
   "  averNormDifferencePerc = " + averNormDifferencePerc);
    returnCode = 0;
    return returnCode;

 }   // End of the method
```

The test method has a similar processing logic with the exception of building and training the network. Instead of building and training the network, it loads the previously saved trained network in memory. It also loads the test data set in memory. By looping over the pair data set, it gets the input, target, and predicted values for each record. The `xPoint` value in the loop is taken from the interval [3.141592654, 4.341592654].

Training Results for Example 3b

Listing 6-9 shows the training results.

Listing 6-9. Training Results

```
xPoint = 0.12 TargetValue = 0.12058 PredictedValue = 0.12072 DiffPerc = 0.00143
xPoint = 0.24 TargetValue = 0.12414 PredictedValue = 0.12427 DiffPerc = 0.00135
xPoint = 0.36 TargetValue = 0.13169 PredictedValue = 0.13157 DiffPerc = 9.6467E-4
xPoint = 0.48 TargetValue = 0.14421 PredictedValue = 0.14410 DiffPerc = 0.00100
xPoint = 0.60 TargetValue = 0.16353 PredictedValue = 0.16352 DiffPerc = 5.31138E-5
xPoint = 0.72 TargetValue = 0.19293 PredictedValue = 0.19326 DiffPerc = 0.00307
xPoint = 0.84 TargetValue = 0.23856 PredictedValue = 0.23842 DiffPerc = 0.00128
xPoint = 0.96 TargetValue = 0.31273 PredictedValue = 0.31258 DiffPerc = 0.00128
xPoint = 1.08 TargetValue = 0.44286 PredictedValue = 0.44296 DiffPerc = 8.16305E-4
xPoint = 1.20 TargetValue = 0.70093 PredictedValue = 0.70088 DiffPerc = 4.05989E-4
xPoint = 1.32 TargetValue = 1.33119 PredictedValue = 1.33123 DiffPerc = 2.74089E-4

maxErrorPerc = 0.0030734810314331077
averErrorPerc = 9.929718215067468E-4
```

Figure 6-7 shows the chart of the actual function values versus the validation results.

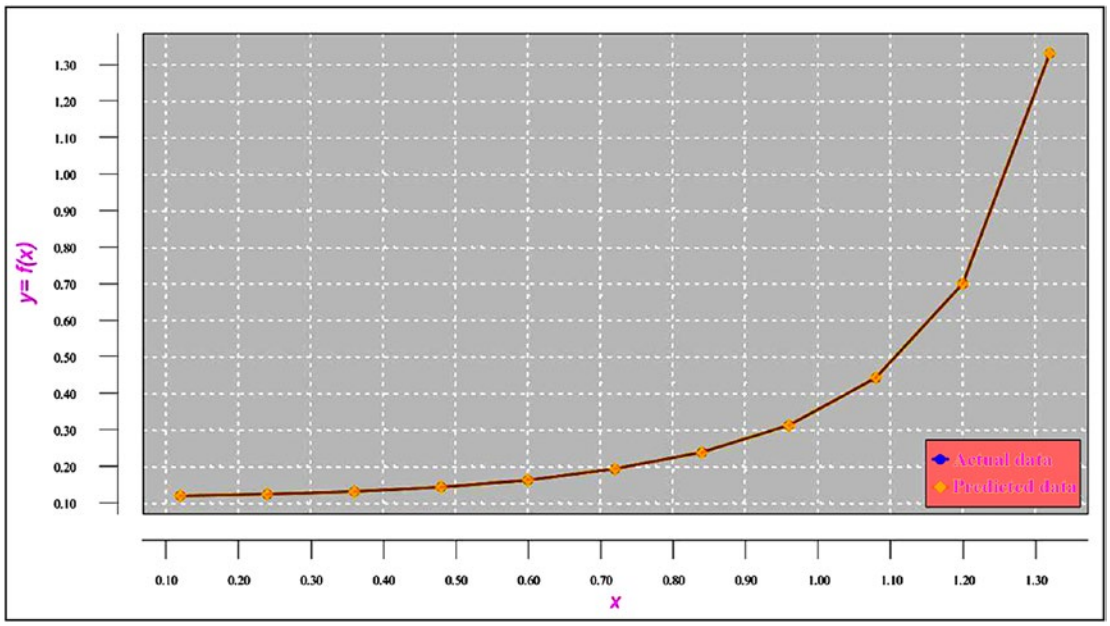


Figure 6-7. *Chart of the training/validation results*

As shown in Figure 6-7, both charts practically overlap.

Testing Results for Example 3b

Listing 6-10 shows the testing results on the interval [3.141592654, 4.341592654].

Listing 6-10. Testing Results on the Interval [3.141592654, 4.341592654]

```
xPoint = 3.26159 TargetValue = 10.12058 PredictedValue = 10.12072 DiffPerc = 0.00143
xPoint = 3.38159 TargetValue = 10.24472 PredictedValue = 10.24485 DiffPerc = 0.00135
xPoint = 3.50159 TargetValue = 10.37640 PredictedValue = 10.37630 DiffPerc = 9.64667E-4
xPoint = 3.62159 TargetValue = 10.52061 PredictedValue = 10.52050 DiffPerc = 0.00100
xPoint = 3.74159 TargetValue = 10.68414 PredictedValue = 10.68413 DiffPerc = 5.31136E-5
xPoint = 3.86159 TargetValue = 10.87707 PredictedValue = 10.87740 DiffPerc = 0.00307
xPoint = 3.98159 TargetValue = 11.11563 PredictedValue = 11.11549 DiffPerc = 0.00127
xPoint = 4.10159 TargetValue = 11.42836 PredictedValue = 11.42821 DiffPerc = 0.00128
xPoint = 4.22159 TargetValue = 11.87122 PredictedValue = 11.87131 DiffPerc = 8.16306E-4
xPoint = 4.34159 TargetValue = 12.57215 PredictedValue = 12.57210 DiffPerc = 4.06070E-4
xPoint = 4.46159 TargetValue = 13.90335 PredictedValue = 13.90338 DiffPerc = 2.74161E-4

maxErrorPerc = 0.003073481240844822
averErrorPerc = 9.929844994337172E-4
```

Figure 6-8 shows the chart of the testing results (actual function values versus the predicted function values) on the interval [3.141592654, 9.424777961].

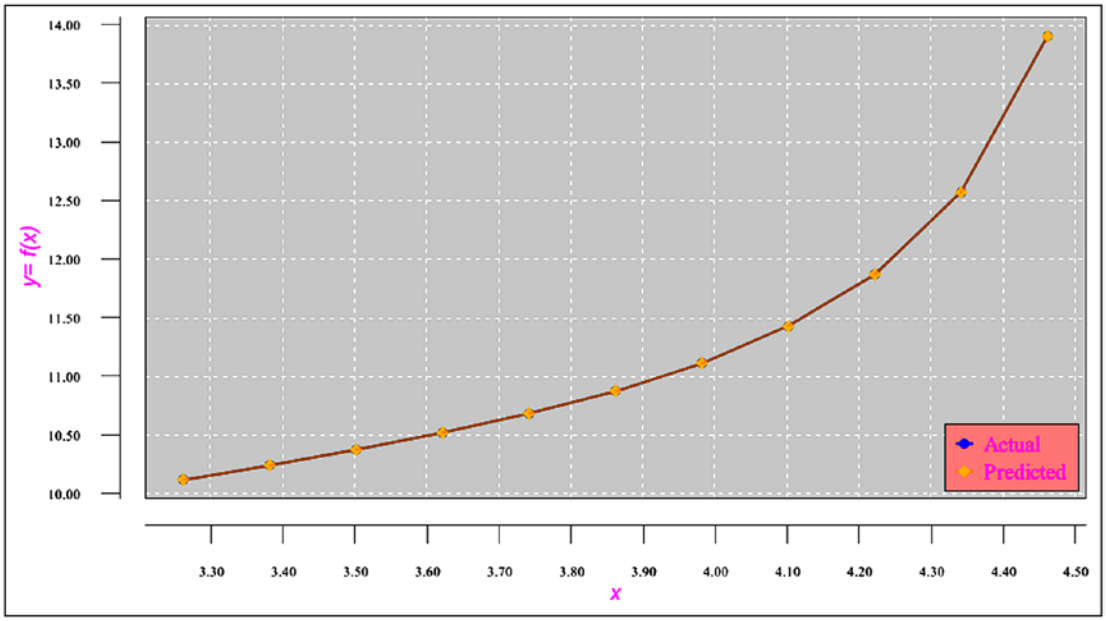


Figure 6-8. *Chart of the testing results*

Both actual and predicted charts practically overlap.

Summary

Again, neural networks are the universal function approximation mechanism. That means that once you have approximated the function on some interval, you can use such a trained neural network to predict the function values at any point within the training interval. However, you cannot use such a trained network for predicting the function values outside the training range. A neural network is not a function extrapolation mechanism.

This chapter explained how, for a certain class of functions (in this case, periodic functions), it is possible get the predicted data outside the training range. You will continue exploring this concept in the next chapter.

CHAPTER 7

Processing Complex Periodic Functions

This chapter continues the discussion of how to process periodic functions, concentrating on more complex periodic functions.

Example 4: Approximation of a Complex Periodic Function

Let's take a look at the function chart shown in Figure 7-1. The function represents some experimental data measured in days (x is the consecutive days of the experiment). This is a periodic function with a period equal to 50 days.

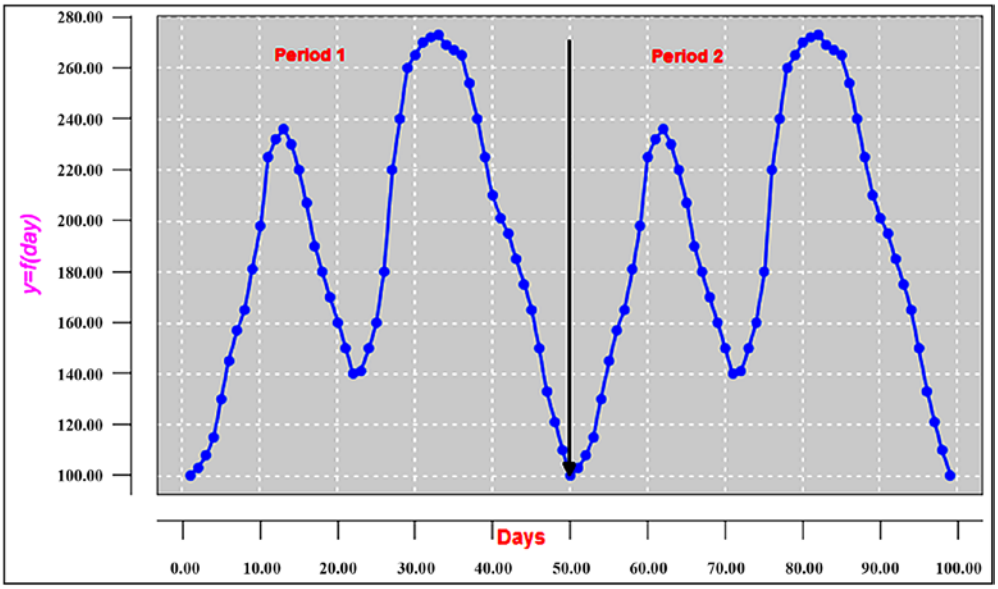


Figure 7-1. *Chart of the periodic function at two intervals: 1–50 and 51–100 days*

I. Livshin, *Artificial Neural Networks with Java*, https://doi.org/10.1007/978-1-4842-4421-0_7

Table 7-1 shows the function values for two periods (1–50 and 50–100 days).

Table 7-1. *Function Values at Two Periods*

Day	Function Value (Period 1)	Day	Function Value (Period 2)
1	100	51	103
2	103	52	108
3	108	53	115
4	115	54	130
5	130	55	145
6	145	56	157
7	157	57	165
8	165	58	181
9	181	59	198
10	198	60	225
11	225	61	232
12	232	62	236
13	236	63	230
14	230	64	220
15	220	65	207
16	207	66	190
17	190	67	180
18	180	68	170
19	170	69	160
20	160	70	150
21	150	71	140
22	140	72	141
23	141	73	150
24	150	74	160
25	160	75	180

(*continued*)

Table 7-1. (*continued*)

Day	Function Value (Period 1)	Day	Function Value (Period 2)
26	180	76	220
27	220	77	240
28	240	78	260
29	260	79	265
30	265	80	270
31	270	81	272
32	272	82	273
33	273	83	269
34	269	84	267
35	267	85	265
36	265	86	254
37	254	87	240
38	240	88	225
39	225	89	210
40	210	90	201
41	201	91	195
42	195	92	185
43	185	93	175
44	175	94	165
45	165	95	150
46	150	96	133
47	133	97	121
48	121	98	110
49	110	99	100
50	100	100	103

Data Preparation

For this example, you will train the neural network using the function values at the first interval and then test the network by getting the network-predicted function values at the second interval. Like the previous example, to be able to determine the function approximation results outside the training range, we would use the difference between xPoint values and the difference between the function values instead of the given xPoints and function values. However, in this example, we will use the difference between xPoint values between the current and previous points as field 1 and the difference between the function values between the next and current points as field 2.

With such settings in the input file, we will teach the network to learn that when the difference between the xPoint values is equal to some value "a," then the difference in function values between the next day and the current day must be equal to some value "b." This allows the network to predict the next day's function value by knowing the current day's (record's) function value.

During the test, we will calculate the function's next-day value in the following way. By being at point x = 50, you want to calculate the predicted function value at the next point, x = 51. Feeding the difference between the xPoint values at the current and previous points (field 1) to the trained network, we will get back the predicted difference between the function values at the next and current points. Therefore, the predicted function value at the next point is equal to the sum of the actual function value at the current point and the predicted value difference obtained from the trained network.

However, this will not work for this example, simply because many parts of the chart may have the same difference and direction in function values between the current and previous days. It will confuse the neural network learning process when it tries to determine to which part of the chart such a point belongs to (see Figure 7-2).

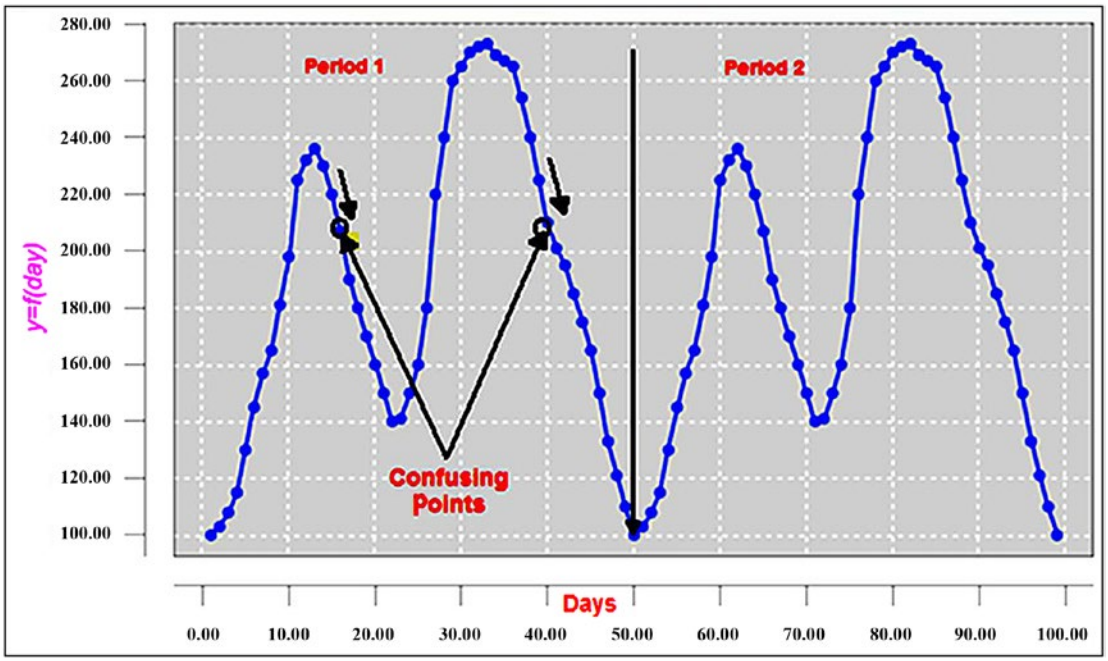


Figure 7-2. *Confusing points on the function chart*

Reflecting Function Topology in the Data

For this example, you need to use an additional trick. You will include the function topology in the data to help the network distinguish between confusing points. Specifically, your training file will use sliding windows as the input records.

Each sliding window record includes the input function value differences (between the current and previous days) of the ten previous records. The function values for the ten previous days are included in the sliding window because ten days is sufficient to make the confusing records distinguishable. The target function value of the sliding window is the target function value difference (between the next and current days) of the original record 11.

You are building the sliding window record that consists of ten fields that contain the function values for the ten previous days, because ten days is sufficient to distinguish confusing points on the chart. However, more days can be included in the record (say, 12 days).

Essentially, by using such a record format, you teach the network to learn the following conditions. If the difference in function values for the ten previous records is equal to a1, a2, a3, a4, a5, a6, a7, a8, a9, and a10, then the difference in the next day's function value and the current day's function value should be equal to the target function value of the next record (record 11). Figure 7-3 shows the visual example of constructing a sliding window record.

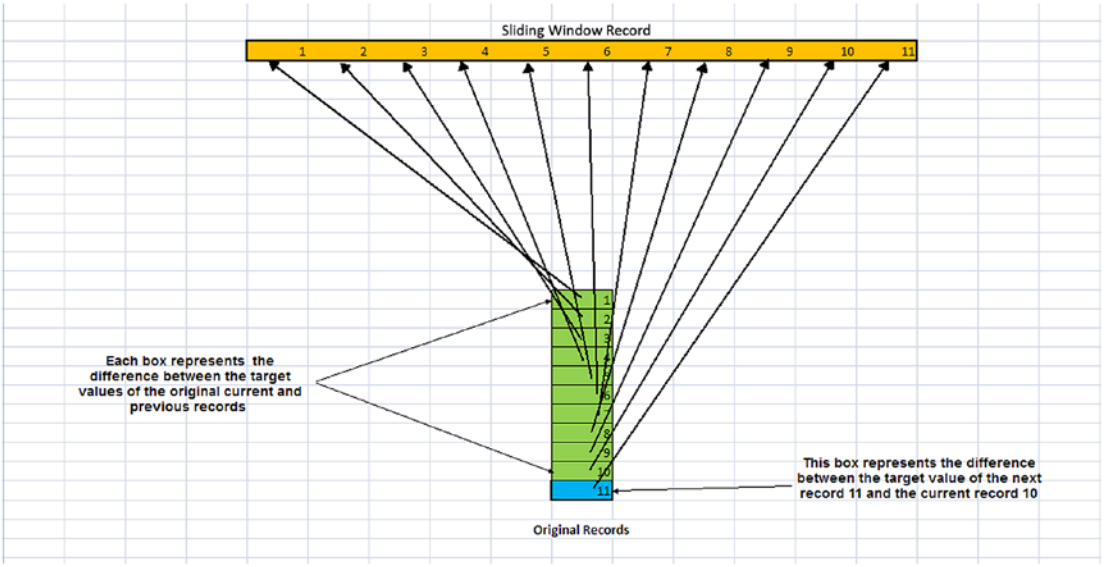


Figure 7-3. *Constructing the sliding window record*

Table 7-2 shows the sliding window training data set. Each sliding window (record) includes ten fields that come from the previous ten days plus one extra field with the expected value to predict.

Table 7-2. *Sliding Windows Data Set on the Interval [1, 50]*

Sliding Windows										
-9	-6	-10	-10	-10	-15	-17	-12	-11	-10	3
-6	-10	-10	-10	-15	-17	-12	-11	-10	3	5
-10	-10	-10	-15	-17	-12	-11	-10	3	3	7
-10	-10	-15	-17	-12	-11	-10	3	3	7	15
-10	-15	-17	-12	-11	-10	3	3	7	15	15
-15	-17	-12	-11	-10	3	3	7	15	15	12
-17	-12	-11	-10	3	3	7	15	15	12	8
-12	-11	-10	3	3	7	15	15	12	8	16
-11	-10	3	3	7	15	15	12	8	16	17
-10	3	3	7	15	15	12	8	16	17	27
3	3	7	15	15	12	8	16	17	27	7
3	7	15	15	12	8	16	17	27	7	4
7	15	15	12	8	16	17	27	7	4	-6
15	15	12	8	16	17	27	7	4	-6	-10
15	12	8	16	17	27	7	4	-6	-10	-13
12	8	16	17	27	7	4	-6	-10	-13	-17
8	16	17	27	7	4	-6	-10	-13	-17	-10
16	17	27	7	4	-6	-10	-13	-17	-10	-10
17	27	7	4	-6	-10	-13	-17	-10	-10	-10
27	7	4	-6	-10	-13	-17	-10	-10	-10	-10
7	4	-6	-10	-13	-17	-10	-10	-10	-10	-10
4	-6	-10	-13	-17	-10	-10	-10	-10	-10	1
-6	-10	-13	-17	-10	-10	-10	-10	-10	1	9
-10	-13	-17	-10	-10	-10	-10	-10	1	9	10
-13	-17	-10	-10	-10	-10	-10	1	9	10	20
-17	-10	-10	-10	-10	-10	1	9	10	20	40

(*continued*)

Table 7-2. (*continued*)

Sliding Windows										
-10	-10	-10	-10	-10	1	9	10	20	40	20
-10	-10	-10	-10	1	9	10	20	40	20	20
-10	-10	-10	1	9	10	20	40	20	20	5
-10	-10	1	9	10	20	40	20	20	5	5
-10	1	9	10	20	40	20	20	5	5	2
1	9	10	20	40	20	20	5	5	2	1
9	10	20	40	20	20	5	5	2	1	-4
10	20	40	20	20	5	5	2	1	-4	-2
20	40	20	20	5	5	2	1	-4	-2	-2
40	20	20	5	5	2	1	-4	-2	-2	-11
20	20	5	5	2	1	-4	-2	-2	-11	-14
20	5	5	2	1	-4	-2	-2	-11	-14	-15
5	5	2	1	-4	-2	-2	-11	-14	-15	-15
5	2	1	-4	-2	-2	-11	-14	-15	-15	-9
2	1	-4	-2	-2	-11	-14	-15	-15	-9	-6
1	-4	-2	-2	-11	-14	-15	-15	-9	-6	-10
-4	-2	-2	-11	-14	-15	-15	-9	-6	-10	-10
-2	-2	-11	-14	-15	-15	-9	-6	-10	-10	-10
-2	-11	-14	-15	-15	-9	-6	-10	-10	-10	-15
-11	-14	-15	-15	-9	-6	-10	-10	-10	-15	-17
-14	-15	-15	-9	-6	-10	-10	-10	-15	-17	-12
-15	-15	-9	-6	-10	-10	-10	-15	-17	-12	-11
-15	-9	-6	-10	-10	-10	-15	-17	-12	-11	-10
-9	-6	-10	-10	-10	-15	-17	-12	-11	-10	3

This data set needs to be normalized on the interval [-1.1]. Figure 7-4 shows the fragment of the normalized sliding windows data set. The 11th field of each record holds the prediction value. Table 7-3 shows the normalized training data set.

Table 7-3. *Normalized Training Data Set*

Normalized Sliding Windows										
-0.68571	-0.6	-0.71429	-0.71429	-0.71429	-0.85714	-0.91429	-0.77143	-0.74286	-0.71429	-0.34286
-0.6	-0.71429	-0.71429	-0.71429	-0.85714	-0.91429	-0.77143	-0.74286	-0.71429	-0.34286	-0.28571
-0.71429	-0.71429	-0.71429	-0.85714	-0.91429	-0.77143	-0.74286	-0.71429	-0.34286	-0.34286	-0.22857
-0.71429	-0.71429	-0.85714	-0.91429	-0.77143	-0.74286	-0.71429	-0.34286	-0.34286	-0.22857	0
-0.71429	-0.85714	-0.91429	-0.77143	-0.74286	-0.71429	-0.34286	-0.34286	-0.22857	0	0
-0.85714	-0.91429	-0.77143	-0.74286	-0.71429	-0.34286	-0.34286	-0.22857	0	0	-0.08571
-0.91429	-0.77143	-0.74286	-0.71429	-0.34286	-0.34286	-0.22857	0	0	-0.08571	-0.2
-0.77143	-0.74286	-0.71429	-0.34286	-0.34286	-0.22857	0	0	-0.08571	-0.2	0.028571
-0.74286	-0.71429	-0.34286	-0.34286	-0.22857	0	0	-0.08571	-0.2	0.028571	0.057143
-0.71429	-0.34286	-0.34286	-0.22857	0	0	-0.08571	-0.2	0.028571	0.057143	0.342857
-0.34286	-0.34286	-0.22857	0	0	-0.08571	-0.2	0.028571	0.057143	0.342857	-0.22857
-0.34286	-0.22857	0	0	-0.08571	-0.2	0.028571	0.057143	0.342857	-0.22857	-0.31429
-0.22857	0	0	-0.08571	-0.2	0.028571	0.057143	0.342857	-0.22857	-0.31429	-0.6
0	0	-0.08571	-0.2	0.028571	0.057143	0.342857	-0.22857	-0.31429	-0.6	-0.71429
0	-0.08571	-0.2	0.028571	0.057143	0.342857	-0.22857	-0.31429	-0.6	-0.71429	-0.8
-0.08571	-0.2	0.028571	0.057143	0.342857	-0.22857	-0.31429	-0.6	-0.71429	-0.8	-0.91429
-0.2	0.028571	0.057143	0.342857	-0.22857	-0.31429	-0.6	-0.71429	-0.8	-0.91429	-0.71429
0.028571	0.057143	0.342857	-0.22857	-0.31429	-0.6	-0.71429	-0.8	-0.91429	-0.71429	-0.71429

(*continued*)

Table 7-3. (*continued*)

Normalized Sliding Windows										
0.057143	0.342857	-0.22857	-0.31429	-0.6	-0.71429	-0.8	-0.91429	-0.71429	-0.71429	-0.71429
0.342857	-0.22857	-0.31429	-0.6	-0.71429	-0.8	-0.91429	-0.71429	-0.71429	-0.71429	-0.71429
-0.22857	-0.31429	-0.6	-0.71429	-0.8	-0.91429	-0.71429	-0.71429	-0.71429	-0.71429	-0.71429
-0.31429	-0.6	-0.71429	-0.8	-0.91429	-0.71429	-0.71429	-0.71429	-0.71429	-0.71429	-0.4
-0.6	-0.71429	-0.8	-0.91429	-0.71429	-0.71429	-0.71429	-0.71429	-0.71429	-0.4	-0.17143
-0.71429	-0.8	-0.91429	-0.71429	-0.71429	-0.71429	-0.71429	-0.71429	-0.4	-0.17143	-0.14286
-0.8	-0.91429	-0.71429	-0.71429	-0.71429	-0.71429	-0.71429	-0.4	-0.17143	-0.14286	0.142857
-0.91429	-0.71429	-0.71429	-0.71429	-0.71429	-0.71429	-0.4	-0.17143	-0.14286	0.142857	0.714286
-0.71429	-0.71429	-0.71429	-0.71429	-0.71429	-0.4	-0.17143	-0.14286	0.142857	0.714286	0.142857
-0.71429	-0.71429	-0.71429	-0.71429	-0.4	-0.17143	-0.14286	0.142857	0.714286	0.142857	0.142857
-0.71429	-0.71429	-0.71429	-0.4	-0.17143	-0.14286	0.142857	0.714286	0.142857	0.142857	-0.28571
-0.71429	-0.71429	-0.4	-0.17143	-0.14286	0.142857	0.714286	0.142857	0.142857	-0.28571	-0.28571
-0.71429	-0.4	-0.17143	-0.14286	0.142857	0.714286	0.142857	0.142857	-0.28571	-0.28571	-0.37143
-0.4	-0.17143	-0.14286	0.142857	0.714286	0.142857	0.142857	-0.28571	-0.28571	-0.37143	-0.4
-0.17143	-0.14286	0.142857	0.714286	0.142857	0.142857	-0.28571	-0.28571	-0.37143	-0.4	-0.54286
-0.14286	0.142857	0.714286	0.142857	0.142857	-0.28571	-0.28571	-0.37143	-0.4	-0.54286	-0.48571
0.142857	0.714286	0.142857	0.142857	-0.28571	-0.28571	-0.37143	-0.4	-0.54286	-0.48571	-0.48571
0.714286	0.142857	0.142857	-0.28571	-0.28571	-0.37143	-0.4	-0.54286	-0.48571	-0.48571	-0.74286

Normalized Sliding Windows										
0.142857	0.142857	-0.28571	-0.28571	-0.37143	-0.4	-0.54286	-0.48571	-0.48571	-0.74286	-0.82857
0.142857	-0.28571	-0.28571	-0.37143	-0.4	-0.54286	-0.48571	-0.48571	-0.74286	-0.82857	-0.85714
-0.28571	-0.28571	-0.37143	-0.4	-0.54286	-0.48571	-0.48571	-0.74286	-0.82857	-0.85714	-0.85714
-0.28571	-0.37143	-0.4	-0.54286	-0.48571	-0.48571	-0.74286	-0.82857	-0.85714	-0.85714	-0.68571
-0.37143	-0.4	-0.54286	-0.48571	-0.48571	-0.74286	-0.82857	-0.85714	-0.85714	-0.68571	-0.6
-0.4	-0.54286	-0.48571	-0.48571	-0.74286	-0.82857	-0.85714	-0.85714	-0.68571	-0.6	-0.71429
-0.54286	-0.48571	-0.48571	-0.74286	-0.82857	-0.85714	-0.85714	-0.68571	-0.6	-0.71429	-0.71429
-0.48571	-0.48571	-0.74286	-0.82857	-0.85714	-0.85714	-0.68571	-0.6	-0.71429	-0.71429	-0.71429
-0.48571	-0.74286	-0.82857	-0.85714	-0.85714	-0.68571	-0.6	-0.71429	-0.71429	-0.71429	-0.85714
-0.74286	-0.82857	-0.85714	-0.85714	-0.68571	-0.6	-0.71429	-0.71429	-0.71429	-0.85714	-0.91429
-0.82857	-0.85714	-0.85714	-0.68571	-0.6	-0.71429	-0.71429	-0.71429	-0.85714	-0.91429	-0.77143
-0.85714	-0.85714	-0.68571	-0.6	-0.71429	-0.71429	-0.71429	-0.85714	-0.91429	-0.77143	-0.74286
-0.85714	-0.68571	-0.6	-0.71429	-0.71429	-0.71429	-0.85714	-0.91429	-0.77143	-0.74286	-0.71429
-0.68571	-0.6	-0.71429	-0.71429	-0.71429	-0.85714	-0.91429	-0.77143	-0.74286	-0.71429	-0.34286

Network Architecture

You will use the network (shown in Figure 7-5) that consists of the input layer (holding ten neurons), four hidden layers (each holding 13 neurons), and the output layer (holding a single neuron). The reasoning for choosing this architecture was discussed earlier.

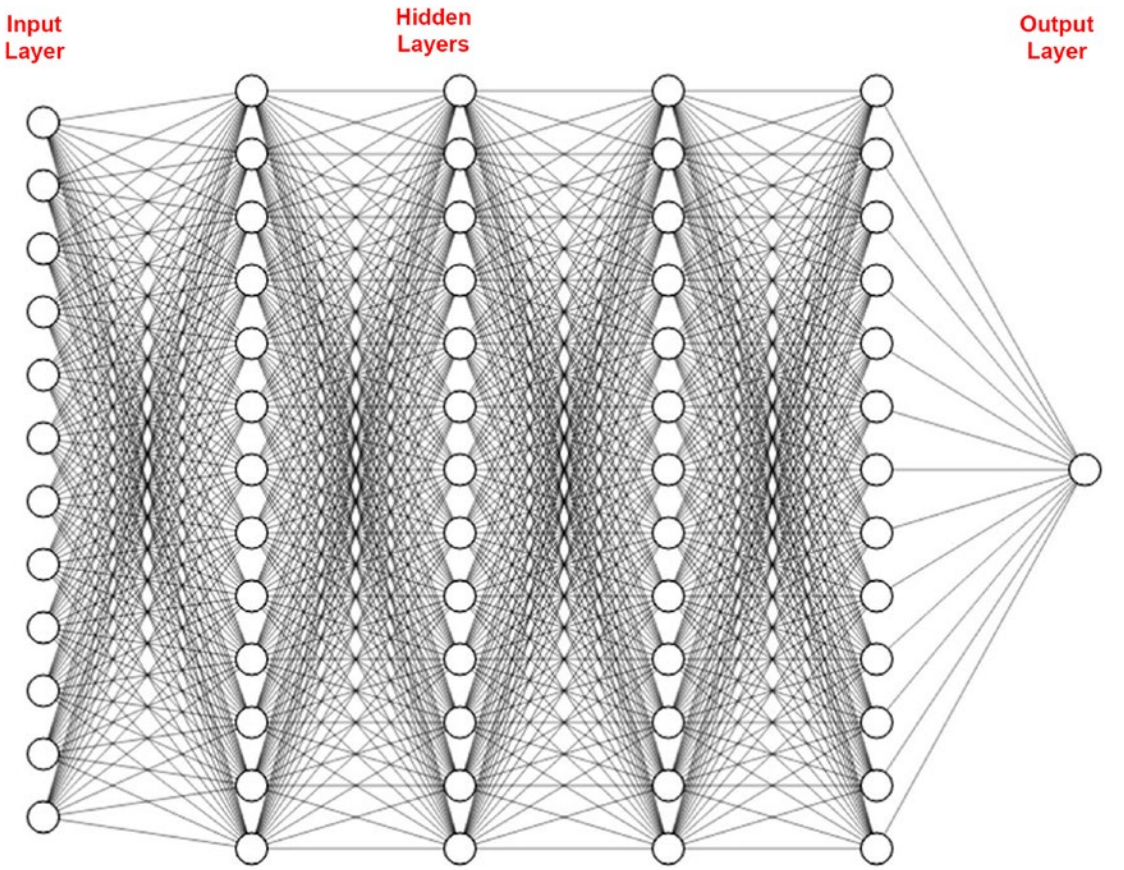


Figure 7-4. *Network architecture*

Each record in the sliding window training data set contains ten input fields and one output field. You are ready to develop the neural network processing program.

Program Code

Listing 7-1 shows the program code.

Listing 7-1. Program Code

```
//============================================================================
//  Approximation of the complex periodic function. The input is a training
//  or testing file with the records built as sliding windows. Each sliding
//  window record contains 11 fields.
//  The first 10 fields are the field1 values from the original 10 records plus
//  the field2 value from the next record, which is actually the difference
    between the target values of the next original record (record 11) and
//  record 10.
//============================================================================

package sample4;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.*;
import java.util.Properties;
import java.time.YearMonth;
import java.awt.Color;
import java.awt.Font;
import java.io.BufferedReader;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.LocalDate;
import java.time.Month;
import java.time.ZoneId;
import java.util.ArrayList;
import java.util.Calendar;
```

```
import java.util.Date;
import java.util.List;
import java.util.Locale;
import java.util.Properties;

import org.encog.Encog;
import org.encog.engine.network.activation.ActivationTANH;
import org.encog.engine.network.activation.ActivationReLU;
import org.encog.ml.data.MLData;
import org.encog.ml.data.MLDataPair;
import org.encog.ml.data.MLDataSet;
import org.encog.ml.data.buffer.MemoryDataLoader;
import org.encog.ml.data.buffer.codec.CSVDataCODEC;
import org.encog.ml.data.buffer.codec.DataSetCODEC;
import org.encog.neural.networks.BasicNetwork;
import org.encog.neural.networks.layers.BasicLayer;
import org.encog.neural.networks.training.propagation.resilient.
ResilientPropagation;
import org.encog.persist.EncogDirectoryPersistence;
import org.encog.util.csv.CSVFormat;

import org.knowm.xchart.SwingWrapper;
import org.knowm.xchart.XYChart;
import org.knowm.xchart.XYChartBuilder;
import org.knowm.xchart.XYSeries;
import org.knowm.xchart.demo.charts.ExampleChart;
import org.knowm.xchart.style.Styler.LegendPosition;
import org.knowm.xchart.style.colors.ChartColor;
import org.knowm.xchart.style.colors.XChartSeriesColors;
import org.knowm.xchart.style.lines.SeriesLines;
import org.knowm.xchart.style.markers.SeriesMarkers;
import org.knowm.xchart.BitmapEncoder;
import org.knowm.xchart.BitmapEncoder.BitmapFormat;
import org.knowm.xchart.QuickChart;
import org.knowm.xchart.SwingWrapper;
```

```
public class Sample4 implements ExampleChart<XYChart>
{

   static double doublePointNumber = 0.00;
   static int intPointNumber = 0;
   static InputStream input = null;
   static double[] arrFunctionValue = new double[500];
   static double inputDiffValue = 0.00;
   static double targetDiffValue = 0.00;
   static double predictDiffValue = 0.00;
   static double valueDifferencePerc = 0.00;
   static String strFunctionValuesFileName;
   static int returnCode  = 0;
   static int numberOfInputNeurons;
   static int numberOfOutputNeurons;
   static int numberOfRecordsInFile;
   static int intNumberOfRecordsInTestFile;
   static double realTargetDiffValue;
   static double realPredictDiffValue;
   static String functionValuesTrainFileName;
   static String functionValuesTestFileName;
   static String trainFileName;
   static String priceFileName;
   static String testFileName;
   static String chartTrainFileName;
   static String chartTestFileName;
   static String networkFileName;
   static int workingMode;
   static String cvsSplitBy = ",";

   // De-normalization parameters
   static double Nh =  1;
   static double Nl = -1;

   static double Dh = 50.00;
   static double Dl = -20.00;
```

```
  static String inputtargetFileName       ;
  static double lastFunctionValueForTraining = 0.00;
  static int tempIndexField;
  static double tempTargetField;
  static  int[] arrIndex = new int[100];
  static double[] arrTarget = new double[100];
  static List<Double> xData = new ArrayList<Double>();
  static List<Double> yData1 = new ArrayList<Double>();
  static List<Double> yData2 = new ArrayList<Double>();

  static XYChart Chart;

 @Override
 public XYChart getChart()
  {

   // Create Chart

   Chart = new  XYChartBuilder().width(900).height(500).title(getClass().
            getSimpleName()).xAxisTitle("Days").yAxisTitle("y= f(x)").build();

   // Customize Chart
   Chart.getStyler().setPlotBackgroundColor(ChartColor.
   getAWTColor(ChartColor.GREY));
   Chart.getStyler().setPlotGridLinesColor(new Color(255, 255, 255));
   Chart.getStyler().setChartBackgroundColor(Color.WHITE);
   Chart.getStyler().setLegendBackgroundColor(Color.PINK);
   Chart.getStyler().setChartFontColor(Color.MAGENTA);
   Chart.getStyler().setChartTitleBoxBackgroundColor(new Color(0, 222, 0));
   Chart.getStyler().setChartTitleBoxVisible(true);
   Chart.getStyler().setChartTitleBoxBorderColor(Color.BLACK);
   Chart.getStyler().setPlotGridLinesVisible(true);
   Chart.getStyler().setAxisTickPadding(20);
   Chart.getStyler().setAxisTickMarkLength(15);
   Chart.getStyler().setPlotMargin(20);
   Chart.getStyler().setChartTitleVisible(false);
   Chart.getStyler().setChartTitleFont(new Font(Font.MONOSPACED,
   Font.BOLD, 24));
```

```
Chart.getStyler().setLegendFont(new Font(Font.SERIF, Font.PLAIN, 18));
Chart.getStyler().setLegendPosition(LegendPosition.OutsideS);
Chart.getStyler().setLegendSeriesLineLength(12);
Chart.getStyler().setAxisTitleFont(new Font(Font.SANS_SERIF,
Font.ITALIC, 18));
Chart.getStyler().setAxisTickLabelsFont(new Font(Font.SERIF, Font.PLAIN, 11));
Chart.getStyler().setDatePattern("yyyy-MM");
Chart.getStyler().setDecimalPattern("#0.00");

// Interval to normalize
double Nh =  1;
double Nl = -1;

// Values in the sliding windows
double Dh = 50.00;
double Dl = -20.00;

try
   {

     // Configuration

     // Setting the mode of the program run
     workingMode = 1; // Set to run the program in the training mode

     if (workingMode == 1)
       {
         // Configure the program to run in the training mode

         trainFileName = "C:/Book_Examples/Sample4_Norm_Train_Sliding_
         Windows_File.csv";
         functionValuesTrainFileName = "C:/Book_Examples/Sample4_
         Function_values_Period_1.csv";
         chartTrainFileName = "XYLine_Sample4_Train_Chart";
         numberOfRecordsInFile = 51;
       }
```

```
else
  {
   // Configure the program to run in the testing mode
    trainFileName = "C:/Book_Examples/Sample4_Norm_Train_Sliding_
    Windows_File.csv";
    functionValuesTrainFileName = "C:/Book_Examples/Sample4_
    Function_values_Period_1.csv";
    chartTestFileName = "XYLine_Sample4_Test_Chart";
    numberOfRecordsInFile = 51;
    lastFunctionValueForTraining = 100.00;
 }

//-------------------------------------------------------
// Common configuration
//-------------------------------------------------------
networkFileName = "C:/Book_Examples/Example4_Saved_Network_File.csv";
inputtargetFileName      = "C:/Book_Examples/Sample4_Input_File.csv";
numberOfInputNeurons = 10;
numberOfOutputNeurons = 1;

// Check the working mode to run

// Training mode. Train, validate, and save the trained network file
if(workingMode == 1)
 {

   // Load function values for training file in memory
   loadFunctionValueTrainFileInMemory();

    File file1 = new File(chartTrainFileName);
    File file2 = new File(networkFileName);

    if(file1.exists())
      file1.delete();

    if(file2.exists())
      file2.delete();

    returnCode = 0;    // Clear the error Code
```

```
        do
         {
           returnCode = trainValidateSaveNetwork();

         } while (returnCode > 0);

     }   // End the train logic

    else
     {
       // Test mode. Test the approximation at the points where
       // neural network was not trained

       // Load function values for training file in memory
       loadInputTargetFileInMemory();

       //loadFunctionValueTrainFileInMemory();

       File file1 = new File(chartTestFileName);

       if(file1.exists())
         file1.delete();

       loadAndTestNetwork();

     }

 }
catch (NumberFormatException e)
 {
     System.err.println("Problem parsing workingMode.workingMode = " +
     workingMode);
     System.exit(1);
  }
catch (Throwable t)
  {
    t.printStackTrace();
    System.exit(1);
  }
```

```
        finally
         {
            Encog.getInstance().shutdown();
         }

     Encog.getInstance().shutdown();

     return Chart;

    }  // End of the method

    // ========================================================================
    // Load CSV to memory.
    // @return The loaded dataset.
    // ========================================================================
    public static MLDataSet loadCSV2Memory(String filename, int input,
    int ideal, boolean headers, CSVFormat format, boolean significance)
      {
         DataSetCODEC codec = new CSVDataCODEC(new File(filename), format,
         headers, input, ideal, significance);
         MemoryDataLoader load = new MemoryDataLoader(codec);
         MLDataSet dataset = load.external2Memory();
         return dataset;
      }

    // ========================================================================
    //  The main method.
    //  @param Command line arguments. No arguments are used.
    // ========================================================================
    public static void main(String[] args)
     {
       ExampleChart<XYChart> exampleChart = new Sample4();
       XYChart Chart = exampleChart.getChart();
       new SwingWrapper<XYChart>(Chart).displayChart();
     } // End of the main method
```

```
//=====================================================================
// This method trains, Validates, and saves the trained network file
//=====================================================================
static public int trainValidateSaveNetwork()
 {
   double functionValue = 0.00;
   double denormInputValueDiff = 0.00;
   double denormTargetValueDiff = 0.00;
   double denormTargetValueDiff_02 = 0.00;
   double denormPredictValueDiff = 0.00;
   double denormPredictValueDiff_02 = 0.00;

   // Load the training CSV file in memory
   MLDataSet trainingSet =
   loadCSV2Memory(trainFileName,numberOfInputNeurons,
   numberOfOutputNeurons,true,CSVFormat.ENGLISH,false);

   // create a neural network
   BasicNetwork network = new BasicNetwork();

   // Input layer
   network.addLayer(new BasicLayer(null,true,10));

   // Hidden layer
   network.addLayer(new BasicLayer(new ActivationTANH(),true,13));
   network.addLayer(new BasicLayer(new ActivationTANH(),true,13));
   network.addLayer(new BasicLayer(new ActivationTANH(),true,13));
   network.addLayer(new BasicLayer(new ActivationTANH(),true,13));

   // Output layer
   network.addLayer(new BasicLayer(new ActivationTANH(),false,1));

   network.getStructure().finalizeStructure();
   network.reset();

   // train the neural network
   final ResilientPropagation train = new ResilientPropagation(network,
   trainingSet);
```

```
int epoch = 1;
returnCode = 0;

do
 {
   train.iteration();
   System.out.println("Epoch #" + epoch + " Error:" + train.getError());

   epoch++;

   if (epoch >= 11000 && network.calculateError(trainingSet) > 0.00000119)
       {
        returnCode = 1;

        System.out.println("Error = " + network.calculateError
        (trainingSet));
        System.out.println("Try again");
        return returnCode;
       }

 } while(train.getError() > 0.000001187);

// Save the network file
EncogDirectoryPersistence.saveObject(new File(networkFileName),network);

double sumGlobalDifferencePerc = 0.00;
double sumGlobalDifferencePerc_02 = 0.00;

double averGlobalDifferencePerc = 0.00;
double maxGlobalDifferencePerc = 0.00;
double maxGlobalDifferencePerc_02 = 0.00;

int m = 0; // Record number in the input file
double xPoint_Initial = 1.00;
double xPoint_Increment = 1.00;
double xPoint = xPoint_Initial - xPoint_Increment;

realTargetDiffValue = 0.00;
realPredictDiffValue = 0.00;
```

```
for(MLDataPair pair: trainingSet)
  {
      m++;
      xPoint = xPoint + xPoint_Increment;

      if(xPoint >  50.00)
         break;

       final MLData output = network.compute(pair.getInput());

       MLData inputData = pair.getInput();
       MLData actualData = pair.getIdeal();
       MLData predictData = network.compute(inputData);

       inputDiffValue = inputData.getData(0);
       targetDiffValue = actualData.getData(0);
       predictDiffValue = predictData.getData(0);

       // De-normalize the values
       denormInputValueDiff     =((Dl - Dh)*inputDiffValue - Nh*Dl +
       Dh*Nl)/(Nl - Nh);
       denormTargetValueDiff = ((Dl - Dh)*targetDiffValue - Nh*Dl +
       Dh*Nl)/(Nl - Nh);
       denormPredictValueDiff =((Dl - Dh)*predictDiffValue - Nh*Dl +
       Dh*Nl)/(Nl - Nh);

       functionValue = arrFunctionValue[m-1];

       realTargetDiffValue = functionValue + denormTargetValueDiff;
       realPredictDiffValue = functionValue + denormPredictValueDiff;

       valueDifferencePerc =
         Math.abs(((realTargetDiffValue - realPredictDiffValue)/
         realPredictDiffValue)*100.00);

       System.out.println ("xPoint = " + xPoint +
       " realTargetDiffValue = " + realTargetDiffValue +
       " realPredictDiffValue = " + realPredictDiffValue);

       sumGlobalDifferencePerc = sumGlobalDifferencePerc +
       valueDifferencePerc;
```

```
            if (valueDifferencePerc > maxGlobalDifferencePerc)
              maxGlobalDifferencePerc = valueDifferencePerc;

            xData.add(xPoint);
            yData1.add(realTargetDiffValue);
            yData2.add(realPredictDiffValue);

        }   // End for pair loop

        XYSeries series1 = Chart.addSeries("Actual data", xData, yData1);
        XYSeries series2 = Chart.addSeries("Predict data", xData, yData2);

        series1.setLineColor(XChartSeriesColors.BLUE);
        series2.setMarkerColor(Color.ORANGE);
        series1.setLineStyle(SeriesLines.SOLID);
        series2.setLineStyle(SeriesLines.SOLID);

        try
         {
           //Save the chart image
           BitmapEncoder.saveBitmapWithDPI(Chart, chartTrainFileName,
           BitmapFormat.JPG, 100);
           System.out.println ("Train Chart file has been saved") ;
         }
       catch (IOException ex)
        {
         ex.printStackTrace();
         System.exit(3);
        }

        // Finally, save this trained network
        EncogDirectoryPersistence.saveObject(new File(networkFileName),network);
        System.out.println ("Train Network has been saved") ;

        averGlobalDifferencePerc = sumGlobalDifferencePerc/numberOfRecordsInFile;

        System.out.println(" ");
        System.out.println("maxGlobalDifferencePerc = " + maxGlobalDifferencePerc +
                "  averGlobalDifferencePerc = " + averGlobalDifferencePerc);
```

```
    returnCode = 0;

    return returnCode;

 }   // End of the method

//=========================================================================
// Testing Method
//=========================================================================
static public void loadAndTestNetwork()
 {
  System.out.println("Testing the networks results");

  List<Double> xData = new ArrayList<Double>();
  List<Double> yData1 = new ArrayList<Double>();
  List<Double> yData2 = new ArrayList<Double>();

  int intStartingPriceIndexForBatch = 0;
  int intStartingDatesIndexForBatch = 0;
  double sumGlobalDifferencePerc = 0.00;
  double maxGlobalDifferencePerc = 0.00;
  double averGlobalDifferencePerc = 0.00;
  double targetToPredictPercent = 0;
  double maxGlobalResultDiff = 0.00;
  double averGlobalResultDiff = 0.00;
  double sumGlobalResultDiff = 0.00;
  double maxGlobalInputPrice = 0.00;
  double sumGlobalInputPrice = 0.00;
  double averGlobalInputPrice = 0.00;
  double maxGlobalIndex = 0;
  double inputDiffValueFromRecord = 0.00;
  double targetDiffValueFromRecord = 0.00;
  double predictDiffValueFromRecord = 0.00;
  double denormInputValueDiff     = 0.00;
  double denormTargetValueDiff = 0.00;
  double denormTargetValueDiff_02 = 0.00;
  double denormPredictValueDiff = 0.00;
  double denormPredictValueDiff_02 = 0.00;
```

```
double normTargetPriceDiff;
double normPredictPriceDiff;
String tempLine;
String[] tempWorkFields;
double tempInputXPointValueFromRecord = 0.0;
double tempTargetXPointValueFromRecord = 0.00;
double tempValueDiffence = 0.00;
double functionValue;
double minXPointValue = 0.00;
double minTargetXPointValue = 0.00;
int tempMinIndex = 0;
double rTempTargetXPointValue = 0.00;
double rTempPriceDiffPercKey = 0.00;
double rTempPriceDiff = 0.00;
double rTempSumDiff = 0.00;
double r1 = 0.00;
double r2 = 0.00;

BufferedReader br4;
BasicNetwork network;

 int k1 = 0;

// Process testing records
maxGlobalDifferencePerc  = 0.00;
averGlobalDifferencePerc = 0.00;
sumGlobalDifferencePerc = 0.00;

realTargetDiffValue = 0.00;
realPredictDiffValue = 0.00;

// Load the training dataset into memory
MLDataSet trainingSet =
    loadCSV2Memory(trainFileName,numberOfInputNeurons,numberOfOutput
    Neurons,true,CSVFormat.ENGLISH,false);
```

```
// Load the saved trained network
network =
  (BasicNetwork)EncogDirectoryPersistence.loadObject(new
  File(networkFileName));

int m = 0;  // Index of the current record

// Record number in the input file
double xPoint_Initial = 51.00;
double xPoint_Increment = 1.00;
double xPoint = xPoint_Initial - xPoint_Increment;

for (MLDataPair pair:  trainingSet)
 {
   m++;
   xPoint = xPoint + xPoint_Increment;

   final MLData output = network.compute(pair.getInput());

   MLData inputData = pair.getInput();
   MLData actualData = pair.getIdeal();
   MLData predictData = network.compute(inputData);

   // Calculate and print the results
   inputDiffValue = inputData.getData(0);
   targetDiffValue = actualData.getData(0);
   predictDiffValue = predictData.getData(0);

   if(m == 1)
     functionValue = lastFunctionValueForTraining;
   else
    functionValue = realPredictDiffValue;

   // De-normalize the values
   denormInputValueDiff     =((Dl - Dh)*inputDiffValue - Nh*Dl +
   Dh*Nl)/(Nl - Nh);
   denormTargetValueDiff = ((Dl - Dh)*targetDiffValue - Nh*Dl +
   Dh*Nl)/(Nl - Nh);
```

```
    denormPredictValueDiff =((Dl - Dh)*predictDiffValue - Nh*Dl +
    Dh*Nl)/(Nl - Nh);

    realTargetDiffValue = functionValue +  denormTargetValueDiff;
    realPredictDiffValue = functionValue + denormPredictValueDiff;

    valueDifferencePerc =
           Math.abs(((realTargetDiffValue - realPredictDiffValue)/
           realPredictDiffValue)*100.00);

    System.out.println ("xPoint = " + xPoint + " realTargetDiffValue = " +
    realTargetDiffValue + " realPredictDiffValue = " +
    realPredictDiffValue);

    sumGlobalDifferencePerc = sumGlobalDifferencePerc + valueDifferencePerc;

    if (valueDifferencePerc > maxGlobalDifferencePerc)
      maxGlobalDifferencePerc = valueDifferencePerc;

    xData.add(xPoint);
    yData1.add(realTargetDiffValue);
    yData2.add(realPredictDiffValue);

   }  // End for pair loop

  // Print the max and average results

  System.out.println(" ");
  averGlobalDifferencePerc = sumGlobalDifferencePerc/numberOfRecordsInFile;

  System.out.println("maxGlobalResultDiff = " + maxGlobalDifferencePerc);
  System.out.println("averGlobalResultDiff = " + averGlobalDifferencePerc);

  // All testing batch files have been processed
  XYSeries series1 = Chart.addSeries("Actual", xData, yData1);
  XYSeries series2 = Chart.addSeries("Predicted", xData, yData2);

  series1.setLineColor(XChartSeriesColors.BLUE);
  series2.setMarkerColor(Color.ORANGE);
  series1.setLineStyle(SeriesLines.SOLID);
  series2.setLineStyle(SeriesLines.SOLID);
```

```
   // Save the chart image
   try
    {
      BitmapEncoder.saveBitmapWithDPI(Chart, chartTestFileName,
      BitmapFormat.JPG, 100);
    }
   catch (Exception bt)
    {
      bt.printStackTrace();
    }

   System.out.println ("The Chart has been saved");

   System.out.println("End of testing for test records");
  } // End of the method

//-----------------------------------------------------------------------
// Load Function values for training file in memory
//-----------------------------------------------------------------------
public static void loadFunctionValueTrainFileInMemory()
 {
   BufferedReader br1 = null;

   String line = "";
   String cvsSplitBy = ",";
   String tempXPointValue = "";
   double tempYFunctionValue = 0.00;

    try
      {
        br1 = new BufferedReader(new FileReader(functionValuesTrainFileName));

       int i = -1;
       int r = -2;

       while ((line = br1.readLine()) != null)
        {
          i++;
          r++;
```

```
            // Skip the header line
            if(i > 0)
              {
               // Break the line using comma as separator
               String[] workFields = line.split(cvsSplitBy);

               tempYFunctionValue = Double.parseDouble(workFields[0]);
               arrFunctionValue[r] = tempYFunctionValue;

               //System.out.println("arrFunctionValue[r] = " +
               arrFunctionValue[r]);
              }

         }  // end of the while loop

          br1.close();

        }
       catch (IOException ex)
        {
          ex.printStackTrace();
          System.err.println("Error opening files = " + ex);
          System.exit(1);
        }

    }

  //=========================================================================
  // Load Sample4_Input_File into 2 arrays in memory
  //=========================================================================
  public static void loadInputTargetFileInMemory()
  {
     BufferedReader br1 = null;

     String line = "";
     String cvsSplitBy = ",";
     String tempXPointValue = "";
     double tempYFunctionValue = 0.00;
```

```
        try
          {
           br1 = new BufferedReader(new FileReader(inputtargetFileName ));

           int i = -1;
           int r = -2;

           while ((line = br1.readLine()) != null)
            {
              i++;
              r++;

             // Skip the header line
             if(i > 0)
               {
                 // Break the line using comma as separator
                 String[] workFields = line.split(cvsSplitBy);

                 tempTargetField = Double.parseDouble(workFields[1]);

                 arrIndex[r] =  r;
                 arrTarget[r] = tempTargetField;
               }

          }  // end of the while loop

           br1.close();

         }
        catch (IOException ex)
         {
           ex.printStackTrace();
           System.err.println("Error opening files = " + ex);
           System.exit(1);
         }

      }

} // End of the class
```

As always, you'll load the training file into memory and build the network. The built network has the input layer with ten neurons, the four hidden layers (each with 13 neurons), and the output layer with a single neuron. Once the network is built, you'll train the network by looping over the epochs until the network error clears the error limit. Finally, you'll save the trained network on disk (it will be used later by the testing method).

Notice that you call the training method in a loop (as you did in the previous example). When after 11,000 iterations the network error is still not less than the error limit, you exit the training method with a return code of 1. That will trigger calling the training method again with the new set of weight/bias parameters (Listing 7-2).

Listing 7-2. Code Fragment at the Beginning of the Training Method

```
// Load the training CSV file in memory
MLDataSet trainingSet =
     loadCSV2Memory(trainFileName,numberOfInputNeurons,numberOfOutputNeurons,
        true,CSVFormat.ENGLISH,false);

// Create a neural network
 BasicNetwork network = new BasicNetwork();

// Input layer
network.addLayer(new BasicLayer(null,true,10));

// Hidden layer
network.addLayer(new BasicLayer(new ActivationTANH(),true,13));
network.addLayer(new BasicLayer(new ActivationTANH(),true,13));
network.addLayer(new BasicLayer(new ActivationTANH(),true,13));
network.addLayer(new BasicLayer(new ActivationTANH(),true,13));
// Output layer
network.addLayer(new BasicLayer(new ActivationTANH(),false,1));

network.getStructure().finalizeStructure();
network.reset();

// train the neural network
final ResilientPropagation train = new ResilientPropagation(network,
trainingSet);
```

```
int epoch = 1;
returnCode = 0;

do
    {
        train.iteration();
        System.out.println("Epoch #" + epoch + " Error:" + train.getError());

        epoch++;

        if (epoch >= 11000 && network.calculateError(trainingSet) > 0.00000119)
           {
                // Exit the training method with the return code = 1

                returnCode = 1;
                System.out.println("Try again");
                 return returnCode;
          }

       } while(train.getError() > 0.000001187);

      // Save the network file

      EncogDirectoryPersistence.saveObject(new File(networkFileName),network);
```

Next, you loop over the pair data set, getting from the network the input, actual, and predicted values for each record. The record values are normalized, so you denormalize their values. The following formula is used for denormalization.

$$f(x) = \left((D_L - D_H)^* x - N_H{}^* D_L + N_L{}^* D_H \right) / (N_L - N_H)$$

where:

x: Input data point

D_L: Minimum (lowest) value of x in the input data set

D_H: Maximum (highest) value of x in the input data set

N_L: The left part of the normalized interval [-1, 1] = -1

N_H: The right part of the normalized interval [-1, 1] = 1

After the denormalization, you calculate the `realTargetDiffValue` and `realPredictDiffValue` fields, print their values in the processing log, and populate the chart data for the current record. Finally, you save the chart file on disk and exit the training method with return code 0 (Listing 7-3).

Listing 7-3. Code Fragment at the End of the Training Method

```
int m = 0;
double xPoint_Initial = 1.00;
double xPoint_Increment = 1.00;
double xPoint = xPoint_Initial - xPoint_Increment;
realTargetDiffValue = 0.00;
realPredictDiffValue = 0.00;

for(MLDataPair pair: trainingSet)
    {
            m++;
            xPoint = xPoint + xPoint_Increment;

             final MLData output = network.compute(pair.getInput());

             MLData inputData = pair.getInput();
             MLData actualData = pair.getIdeal();
             MLData predictData = network.compute(inputData);

             // Calculate and print the results
             inputDiffValue = inputData.getData(0);
             targetDiffValue = actualData.getData(0);
             predictDiffValue = predictData.getData(0);

             // De-normalize the values
             denormInputValueDiff     =((Dl - Dh)*inputDiffValue - Nh*Dl +
             Dh*Nl)/(Nl - Nh);
             denormTargetValueDiff = ((Dl - Dh)*targetDiffValue - Nh*Dl +
             Dh*Nl)/(Nl - Nh);
             denormPredictValueDiff =((Dl - Dh)*predictDiffValue - Nh*Dl +
             Dh*Nl)/(Nl - Nh);
             functionValue = arrFunctionValue[m-1];
```

```
      realTargetDiffValue = functionValue + denormTargetValueDiff;
      realPredictDiffValue = functionValue + denormPredictValueDiff;

      valueDifferencePerc =
        Math.abs(((realTargetDiffValue - realPredictDiffValue)/
        realPredictDiffValue)*100.00);

      System.out.println ("xPoint = " + xPoint +
      "  realTargetDiffValue = " + realTargetDiffValue +
      "  realPredictDiffValue = " + realPredictDiffValue);

      sumDifferencePerc = sumDifferencePerc + valueDifferencePerc;

      if (valueDifferencePerc > maxDifferencePerc)
        maxDifferencePerc = valueDifferencePerc;

      xData.add(xPoint);
      yData1.add(realTargetDiffValue);
      yData2.add(realPredictDiffValue);

}   // End for pair loop

XYSeries series1 = Chart.addSeries("Actual data", xData, yData1);
XYSeries series2 = Chart.addSeries("Predict data", xData, yData2);

series1.setLineColor(XChartSeriesColors.BLUE);
series2.setMarkerColor(Color.ORANGE);
series1.setLineStyle(SeriesLines.SOLID);
series2.setLineStyle(SeriesLines.SOLID);
try
 {
   //Save the chart image
   BitmapEncoder.saveBitmapWithDPI(Chart, chartTrainFileName,
   BitmapFormat.JPG, 100);
   System.out.println ("Train Chart file has been saved") ;
 }
```

```
        catch (IOException ex)
         {
          ex.printStackTrace();
          System.exit(3);
         }

         // Finally, save this trained network
         EncogDirectoryPersistence.saveObject(new File(networkFileName),network);
         System.out.println ("Train Network has been saved") ;

         averNormDifferencePerc  = sumDifferencePerc/numberOfRecordsInFile;

         System.out.println(" ");
         System.out.println("maxDifferencePerc = " + maxDifferencePerc +
         "  averNormDifferencePerc = " + averNormDifferencePerc);

         returnCode = 0;
         return returnCode;

}   // End of the method
```

So far, the processing logic of the training method is about the same as in the previous examples, disregarding that the format of the training data set is different and includes sliding window records. You will see a substantial change in the logic of the testing method.

Training the Network

Listing 7-4 shows the training processing results.

Listing 7-4. Training Results

```
xPoint =  1.0  TargetDiff = 102.99999  PredictDiff = 102.98510
xPoint =  2.0  TargetDiff = 107.99999  PredictDiff = 107.99950
xPoint =  3.0  TargetDiff = 114.99999  PredictDiff = 114.99861
xPoint =  4.0  TargetDiff = 130.0      PredictDiff = 130.00147
xPoint =  5.0  TargetDiff = 145.0      PredictDiff = 144.99901
xPoint =  6.0  TargetDiff = 156.99999  PredictDiff = 157.00011
xPoint =  7.0  TargetDiff = 165.0      PredictDiff = 164.99849
xPoint =  8.0  TargetDiff = 181.00000  PredictDiff = 181.00009
xPoint =  9.0  TargetDiff = 197.99999  PredictDiff = 197.99984
```

```
xPoint = 10.0  TargetDiff = 225.00000  PredictDiff = 224.99914
xPoint = 11.0  TargetDiff = 231.99999  PredictDiff = 231.99987
xPoint = 12.0  TargetDiff = 236.00000  PredictDiff = 235.99949
xPoint = 13.0  TargetDiff = 230.0      PredictDiff = 230.00122
xPoint = 14.0  TargetDiff = 220.00000  PredictDiff = 219.99767
xPoint = 15.0  TargetDiff = 207.0      PredictDiff = 206.99951
xPoint = 16.0  TargetDiff = 190.00000  PredictDiff = 190.00221
xPoint = 17.0  TargetDiff = 180.00000  PredictDiff = 180.00009
xPoint = 18.0  TargetDiff = 170.00000  PredictDiff = 169.99977
xPoint = 19.0  TargetDiff = 160.00000  PredictDiff = 159.98978
xPoint = 20.0  TargetDiff = 150.00000  PredictDiff = 150.07543
xPoint = 21.0  TargetDiff = 140.00000  PredictDiff = 139.89404
xPoint = 22.0  TargetDiff = 141.0      PredictDiff = 140.99714
xPoint = 23.0  TargetDiff = 150.00000  PredictDiff = 149.99875
xPoint = 24.0  TargetDiff = 159.99999  PredictDiff = 159.99929
xPoint = 25.0  TargetDiff = 180.00000  PredictDiff = 179.99896
xPoint = 26.0  TargetDiff = 219.99999  PredictDiff = 219.99909
xPoint = 27.0  TargetDiff = 240.00000  PredictDiff = 240.00141
xPoint = 28.0  TargetDiff = 260.00000  PredictDiff = 259.99865
xPoint = 29.0  TargetDiff = 264.99999  PredictDiff = 264.99938
xPoint = 30.0  TargetDiff = 269.99999  PredictDiff = 270.00068
xPoint = 31.0  TargetDiff = 272.00000  PredictDiff = 271.99931
xPoint = 32.0  TargetDiff = 273.0      PredictDiff = 272.99969
xPoint = 33.0  TargetDiff = 268.99999  PredictDiff = 268.99975
xPoint = 34.0  TargetDiff = 266.99999  PredictDiff = 266.99994
xPoint = 35.0  TargetDiff = 264.99999  PredictDiff = 264.99742
xPoint = 36.0  TargetDiff = 253.99999  PredictDiff = 254.00076
xPoint = 37.0  TargetDiff = 239.99999  PredictDiff = 240.02203
xPoint = 38.0  TargetDiff = 225.00000  PredictDiff = 225.00479
xPoint = 39.0  TargetDiff = 210.00000  PredictDiff = 210.03944
xPoint = 40.0  TargetDiff = 200.99999  PredictDiff = 200.86493
xPoint = 41.0  TargetDiff = 195.0      PredictDiff = 195.11291
xPoint = 42.0  TargetDiff = 185.00000  PredictDiff = 184.91010
xPoint = 43.0  TargetDiff = 175.00000  PredictDiff = 175.02804
xPoint = 44.0  TargetDiff = 165.00000  PredictDiff = 165.07052
xPoint = 45.0  TargetDiff = 150.00000  PredictDiff = 150.01101
```

```
xPoint = 46.0  TargetDiff = 133.00000  PredictDiff = 132.91352
xPoint = 47.0  TargetDiff = 121.00000  PredictDiff = 121.00125
xPoint = 48.0  TargetDiff = 109.99999  PredictDiff = 110.02157
xPoint = 49.0  TargetDiff = 100.00000  PredictDiff = 100.01322
xPoint = 50.0  TargetDiff = 102.99999  PredictDiff = 102.98510

maxErrorPerc = 0.07574160995391013
averErrorPerc = 0.01071011328541703
```

Figure 7-5 shows the chart of the training results.

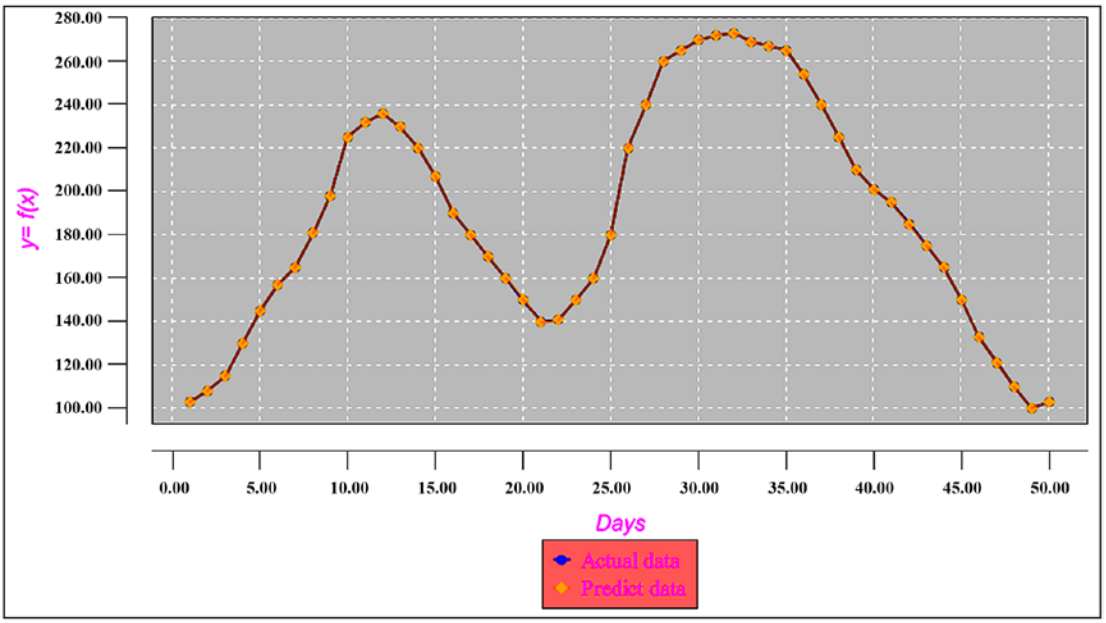


Figure 7-5. *Chart of the training/validation results*

Testing the Network

First, you change the configuration data to process the testing logic. You load the previously saved trained network. Notice here that you don't load the testing data set. You will be determining the current function value in the following way. On the first step of the loop, the current function value is equal to the `lastFunctionValueForTraining` variable, which was calculated during the training process. This variable holds the function value at the last point, which is 50. On all the following steps of the loop, you set the current record value to the function value calculated during the previous step of the loop.

Near the beginning of this example I explained how you will calculate the predicted values during the testing phase. I repeat this explanation here:

> *"During the test, you will calculate the function's next-day value in the following way. By being at point x = 50, you want to calculate the predicted function value at the next point, x = 51. Feeding the difference between the xPoint values at the current and previous points (field 1) to the trained network, you will get back the predicted difference between the function values at the next and current points. Therefore, the predicted function value at the next point is equal to the sum of the actual function value at the current point and the predicted value difference obtained from the trained network."*

Next, you loop over the pair data set starting from xPoint 51 (the first point of the testing interval). At each step of the loop, you obtain the input, actual, and predicted values for each record, denormalize their values, and calculate `realTargetDiffValue` and `realPredictDiffValue` for each record. You print their values as the testing log and populate the chart elements with data for each record. Finally, you save the generated chart file. Listing 7-5 shows the test processing results.

Listing 7-5. Testing Results

```
xPoint = 51.0  TargetDiff = 102.99999  PredictedDiff = 102.98510
xPoint = 52.0  TargetDiff = 107.98510  PredictedDiff = 107.98461
xPoint = 53.0  TargetDiff = 114.98461  PredictedDiff = 114.98322
xPoint = 54.0  TargetDiff = 129.98322  PredictedDiff = 129.98470
xPoint = 55.0  TargetDiff = 144.98469  PredictedDiff = 144.98371
xPoint = 56.0  TargetDiff = 156.98371  PredictedDiff = 156.98383
xPoint = 57.0  TargetDiff = 164.98383  PredictedDiff = 164.98232
xPoint = 58.0  TargetDiff = 180.98232  PredictedDiff = 180.98241
xPoint = 59.0  TargetDiff = 197.98241  PredictedDiff = 197.98225
xPoint = 60.0  TargetDiff = 224.98225  PredictedDiff = 224.98139
xPoint = 61.0  TargetDiff = 231.98139  PredictedDiff = 231.98127
xPoint = 62.0  TargetDiff = 235.98127  PredictedDiff = 235.98077
xPoint = 63.0  TargetDiff = 229.98077  PredictedDiff = 229.98199
xPoint = 64.0  TargetDiff = 219.98199  PredictedDiff = 219.97966
xPoint = 65.0  TargetDiff = 206.97966  PredictedDiff = 206.97917
xPoint = 66.0  TargetDiff = 189.97917  PredictedDiff = 189.98139
```

```
xPoint = 67.0  TargetDiff = 179.98139  PredictedDiff = 179.98147
xPoint = 68.0  TargetDiff = 169.98147  PredictedDiff = 169.98124
xPoint = 69.0  TargetDiff = 159.98124  PredictedDiff = 159.97102
xPoint = 70.0  TargetDiff = 149.97102  PredictedDiff = 150.04646
xPoint = 71.0  TargetDiff = 140.04646  PredictedDiff = 139.94050
xPoint = 72.0  TargetDiff = 140.94050  PredictedDiff = 140.93764
xPoint = 73.0  TargetDiff = 149.93764  PredictedDiff = 149.93640
xPoint = 74.0  TargetDiff = 159.93640  PredictedDiff = 159.93569
xPoint = 75.0  TargetDiff = 179.93573  PredictedDiff = 179.93465
xPoint = 76.0  TargetDiff = 219.93465  PredictedDiff = 219.93374
xPoint = 77.0  TargetDiff = 239.93374  PredictedDiff = 239.93515
xPoint = 78.0  TargetDiff = 259.93515  PredictedDiff = 259.93381
xPoint = 79.0  TargetDiff = 264.93381  PredictedDiff = 264.93318
xPoint = 80.0  TargetDiff = 269.93318  PredictedDiff = 269.93387
xPoint = 81.0  TargetDiff = 271.93387  PredictedDiff = 271.93319
xPoint = 82.0  TargetDiff = 272.93318  PredictedDiff = 272.93287
xPoint = 83.0  TargetDiff = 268.93287  PredictedDiff = 268.93262
xPoint = 84.0  TargetDiff = 266.93262  PredictedDiff = 266.93256
xPoint = 85.0  TargetDiff = 264.93256  PredictedDiff = 264.92998
xPoint = 86.0  TargetDiff = 253.92998  PredictedDiff = 253.93075
xPoint = 87.0  TargetDiff = 239.93075  PredictedDiff = 239.95277
xPoint = 88.0  TargetDiff = 224.95278  PredictedDiff = 224.95756
xPoint = 89.0  TargetDiff = 209.95756  PredictedDiff = 209.99701
xPoint = 90.0  TargetDiff = 200.99701  PredictedDiff = 200.86194
xPoint = 91.0  TargetDiff = 194.86194  PredictedDiff = 194.97485

maxGlobalResultDiff = 0.07571646804925916
averGlobalResultDiff = 0.01071236446121567
```

Figure 7-6 shows the chart of the testing results.

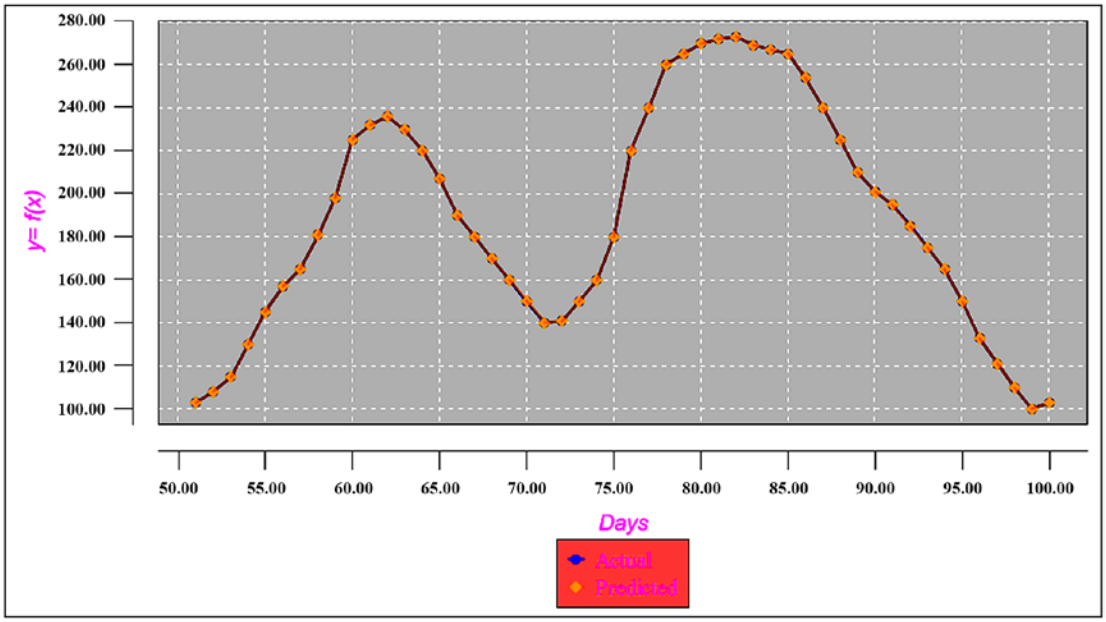


Figure 7-6. *The chart of the test result records*

Both charts practically overlap.

Digging Deeper

In this example, you learned that by using some special data preparation techniques you are able to calculate the function value at the first point of the next interval by knowing the function value at the last point of the training interval. Repeating this process for the rest of the points in the testing interval, you get the function values for all points of the test interval.

Why do I always mention that the function needs to be periodic? Because you determine the function values on the next interval based on the results calculated for the training interval. This technique can also be applied to nonperiodic functions. The only requirement is that the function values on the next interval can be determined in some way based on the values in the training interval. For example, consider a function where values on the next interval double the values on the training interval. Such a function is not periodic, but the techniques discussed in this chapter will work. Also, it is not necessary that each point in the next interval be determined by the corresponding point in the training interval. As long as some rule exists for determining the function

value at some point on the next interval based on the function value at some point on the training interval, this technique will work. That substantially increases the class of functions that the network can process outside the training interval.

Tip How do you obtain the error limit? At the start, just guess the error limit value and run the training process. If you see while looping over the epochs that the network error easily clears the error limit, then decrease the error limit and try again. Keep decreasing the error limit value until you see that the network error is able to clear the error limit; however, it must to work hard to do this.

When you find such an error limit, try to play with the network architecture by changing the number of hidden layers and the number of neurons within the hidden layers to see whether it is possible to decrease the error limit even more. Remember that for more complex function topologies, using more hidden layers will improve the results. If while increasing the number of hidden layers and the number of neurons you reach the point when the network error degrades, stop this process and go back to the previous number of layers and neurons.

Summary

In this chapter, you saw how to approximate a complex periodic function. The training and testing data sets were transformed to the format of sliding window records to add the function topology information to the data. In the next chapter, I will discuss an even more complex situation that involves the approximation of a noncontinuous function (which is currently a known problem for the neural network approximation).

CHAPTER 8

Approximating Noncontinuous Functions

This chapter will discuss the neural network approximation of noncontinuous functions. Currently, this is a problematic area for neural networks because network processing is based on calculating partial function derivatives (using the gradient descent algorithm), and calculating them for noncontinuous functions at the points where the function value suddenly jump or drop leads to questionable results. We will dig deeper into this issue in this chapter. The chapter also includes a method I developed that solves this issue.

Example 5: Approximating Noncontinuous Functions

You will first attempt to approximate a noncontinuous function (shown in Figure 8-1) by using conventional neural network processing so you can see that the results are of very low quality. I will explain why this happens and then introduce the method that allows you to approximate such functions with good precision.

I. Livshin, *Artificial Neural Networks with Java*, https://doi.org/10.1007/978-1-4842-4421-0_8

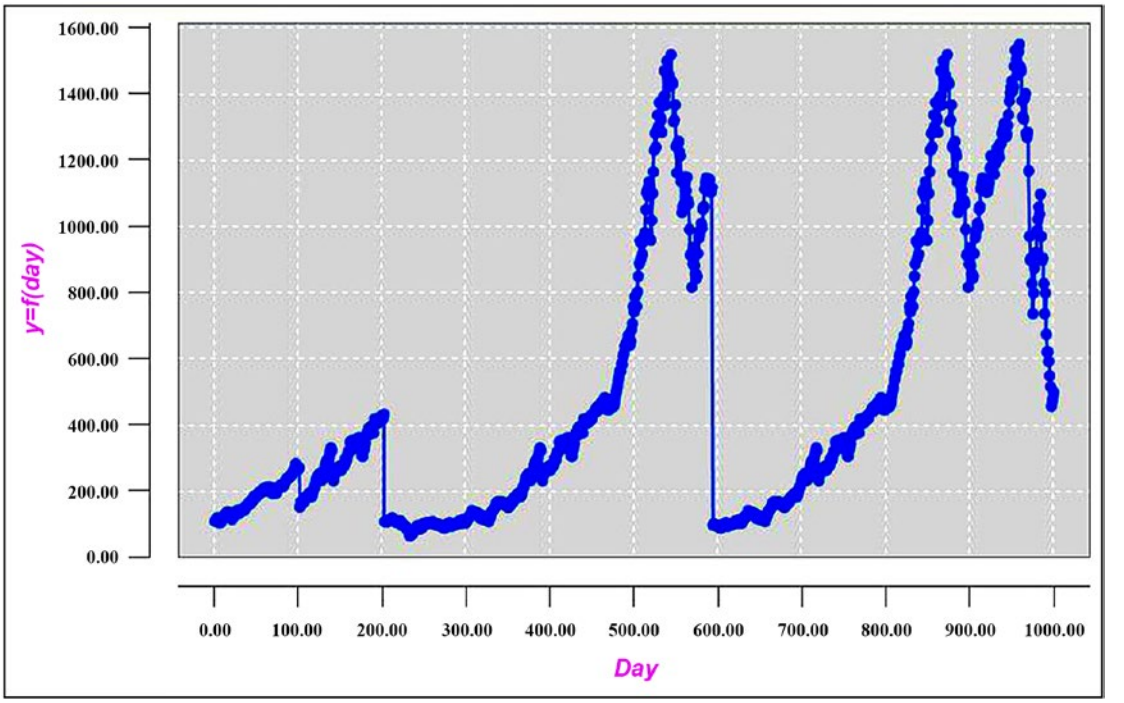


Figure 8-1. *Chart of noncontinuous function*

As you remember from the preceding chapters, neural network backpropagation uses partial derivatives of the network error function to redistribute the error calculated from the output layer to all hidden-layer neurons. It repeats this iterative process by moving in the direction opposite to the divergent function to find one of the local (possibly global) error function minimums. Because of the problem of calculating divergent/derivatives for noncontinuous functions, approximating such functions is problematic.

The function for this example is given by its values at 1,000 points. You are attempting to approximate this function using the traditional neural network backpropagation process. Table 8-1 shows a fragment of the input data set.

Table 8-1. *Fragment of the Input Data Set*

xPoint	yValue	xPoint	yValue	xPoint	yValue
1	107.387	31	137.932	61	199.499
2	110.449	32	140.658	62	210.45
3	116.943	33	144.067	63	206.789
4	118.669	34	141.216	64	208.551
5	108.941	35	141.618	65	210.739
6	103.071	36	142.619	66	206.311
7	110.16	37	149.811	67	210.384
8	104.933	38	151.468	68	197.218
9	114.12	39	156.919	69	192.003
10	118.326	40	159.757	70	207.936
11	118.055	41	163.074	71	208.041
12	125.764	42	160.628	72	204.394
13	128.612	43	168.573	73	194.024
14	132.722	44	163.297	74	193.223
15	132.583	45	168.155	75	205.974
16	136.361	46	175.654	76	206.53
17	134.52	47	180.581	77	209.696
18	132.064	48	184.836	78	209.886
19	129.228	49	178.259	79	217.36
20	121.889	50	185.945	80	217.095
21	113.142	51	187.234	81	216.827
22	125.33	52	188.395	82	212.615
23	124.696	53	192.357	83	219.881
24	125.76	54	196.023	84	223.883
25	131.241	55	193.067	85	227.887

(*continued*)

Table 8-1. (*continued*)

xPoint	yValue	xPoint	yValue	xPoint	yValue
26	136.568	56	200.337	86	236.364
27	140.847	57	197.229	87	236.272
28	139.791	58	201.805	88	238.42
29	131.033	59	206.756	89	241.18
30	136.216	60	205.89	90	242.341

This data set needs to be normalized on the interval [-1,1]. Table 8-2 shows a fragment of the normalized input data set.

Table 8-2. *Fragment of the Normalized Input Data Set*

xPoint	yValue	xPoint	yValue	xPoint	yValue
-1	-0.93846	-0.93994	-0.89879	-0.87988	-0.81883
-0.998	-0.93448	-0.93794	-0.89525	-0.87788	-0.80461
-0.996	-0.92605	-0.93594	-0.89082	-0.87588	-0.80936
-0.99399	-0.92381	-0.93393	-0.89452	-0.87387	-0.80708
-0.99199	-0.93644	-0.93193	-0.894	-0.87187	-0.80424
-0.98999	-0.94406	-0.92993	-0.8927	-0.86987	-0.80999
-0.98799	-0.93486	-0.92793	-0.88336	-0.86787	-0.8047
-0.98599	-0.94165	-0.92593	-0.88121	-0.86587	-0.82179
-0.98398	-0.92971	-0.92392	-0.87413	-0.86386	-0.82857
-0.98198	-0.92425	-0.92192	-0.87045	-0.86186	-0.80788
-0.97998	-0.9246	-0.91992	-0.86614	-0.85986	-0.80774
-0.97798	-0.91459	-0.91792	-0.86931	-0.85786	-0.81248
-0.97598	-0.91089	-0.91592	-0.859	-0.85586	-0.82594
-0.97397	-0.90556	-0.91391	-0.86585	-0.85385	-0.82698

(*continued*)

Table 8-2. (*continued*)

xPoint	yValue	xPoint	yValue	xPoint	yValue
-0.97197	-0.90574	-0.91191	-0.85954	-0.85185	-0.81042
-0.96997	-0.90083	-0.90991	-0.8498	-0.84985	-0.8097
-0.96797	-0.90322	-0.90791	-0.8434	-0.84785	-0.80559
-0.96597	-0.90641	-0.90591	-0.83788	-0.84585	-0.80534
-0.96396	-0.91009	-0.9039	-0.84642	-0.84384	-0.79564
-0.96196	-0.91962	-0.9019	-0.83644	-0.84184	-0.79598
-0.95996	-0.93098	-0.8999	-0.83476	-0.83984	-0.79633
-0.95796	-0.91516	-0.8979	-0.83325	-0.83784	-0.8018
-0.95596	-0.91598	-0.8959	-0.82811	-0.83584	-0.79236
-0.95395	-0.9146	-0.89389	-0.82335	-0.83383	-0.78716
-0.95195	-0.90748	-0.89189	-0.82719	-0.83183	-0.78196
-0.94995	-0.90056	-0.88989	-0.81774	-0.82983	-0.77096
-0.94795	-0.895	-0.88789	-0.82178	-0.82783	-0.77108
-0.94595	-0.89638	-0.88589	-0.81584	-0.82583	-0.76829
-0.94394	-0.90775	-0.88388	-0.80941	-0.82382	-0.7647
-0.94194	-0.90102	-0.88188	-0.81053	-0.82182	-0.76319

Network Architecture

The network for this example consists of the input layer with a single neuron, seven hidden layers (each with five neurons), and an output layer with a single neuron. See Figure 8-2.

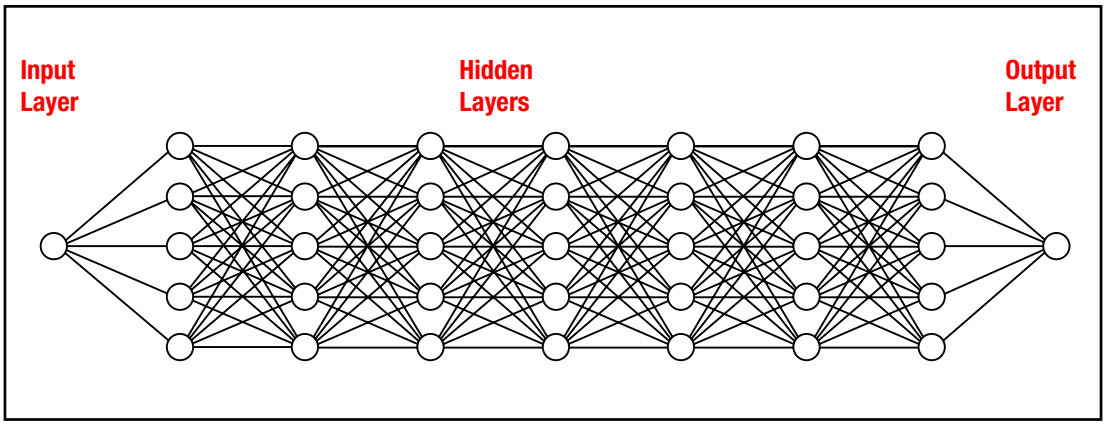


Figure 8-2. *Network architecture*

Program Code

Listing 8-1 shows the program code.

Listing 8-1. Program Code

```
// ============================================================
// Approximation non-continuous function whose values are given
// at 999 points. The input file is normalized.
// ============================================================

package sample5;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.*;
import java.util.Properties;
```

```
import java.time.YearMonth;
import java.awt.Color;
import java.awt.Font;
import java.io.BufferedReader;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.LocalDate;
import java.time.Month;
import java.time.ZoneId;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.List;
import java.util.Locale;
import java.util.Properties;

import org.encog.Encog;
import org.encog.engine.network.activation.ActivationTANH;
import org.encog.engine.network.activation.ActivationReLU;
import org.encog.ml.data.MLData;
import org.encog.ml.data.MLDataPair;
import org.encog.ml.data.MLDataSet;
import org.encog.ml.data.buffer.MemoryDataLoader;
import org.encog.ml.data.buffer.codec.CSVDataCODEC;
import org.encog.ml.data.buffer.codec.DataSetCODEC;
import org.encog.neural.networks.BasicNetwork;
import org.encog.neural.networks.layers.BasicLayer;
import org.encog.neural.networks.training.propagation.resilient.
ResilientPropagation;
import org.encog.persist.EncogDirectoryPersistence;
import org.encog.util.csv.CSVFormat;

import org.knowm.xchart.SwingWrapper;
import org.knowm.xchart.XYChart;
import org.knowm.xchart.XYChartBuilder;
import org.knowm.xchart.XYSeries;
```

```
import org.knowm.xchart.demo.charts.ExampleChart;
import org.knowm.xchart.style.Styler.LegendPosition;
import org.knowm.xchart.style.colors.ChartColor;
import org.knowm.xchart.style.colors.XChartSeriesColors;
import org.knowm.xchart.style.lines.SeriesLines;
import org.knowm.xchart.style.markers.SeriesMarkers;
import org.knowm.xchart.BitmapEncoder;
import org.knowm.xchart.BitmapEncoder.BitmapFormat;
import org.knowm.xchart.QuickChart;
import org.knowm.xchart.SwingWrapper;

public class Sample5 implements ExampleChart<XYChart>
 {
    // Interval to normalize
   static double Nh =  1;
   static double Nl = -1;

   // First column
   static double minXPointDl = 1.00;
   static double maxXPointDh = 1000.00;

   // Second column - target data
   static double minTargetValueDl = 60.00;
   static double maxTargetValueDh = 1600.00;

   static double doublePointNumber = 0.00;
   static int intPointNumber = 0;
   static InputStream input = null;
   static double[] arrPrices = new double[2500];
   static double normInputXPointValue = 0.00;
   static double normPredictXPointValue = 0.00;
   static double normTargetXPointValue = 0.00;
   static double normDifferencePerc = 0.00;
   static double returnCode = 0.00;
   static double denormInputXPointValue = 0.00;
   static double denormPredictXPointValue = 0.00;
   static double denormTargetXPointValue = 0.00;
```

```
 static double valueDifference = 0.00;
 static int numberOfInputNeurons;
 static int numberOfOutputNeurons;
  static int intNumberOfRecordsInTestFile;
 static String trainFileName;
 static String priceFileName;
 static String testFileName;
 static String chartTrainFileName;
 static String chartTestFileName;
 static String networkFileName;
 static int workingMode;
 static String cvsSplitBy = ",";

 static List<Double> xData = new ArrayList<Double>();
 static List<Double> yData1 = new ArrayList<Double>();
 static List<Double> yData2 = new ArrayList<Double>();

 static XYChart Chart;

@Override
public XYChart getChart()
 {
  // Create Chart

  Chart = new  XYChartBuilder().width(900).height(500).title(getClass().
               getSimpleName()).xAxisTitle("x").yAxisTitle("y= f(x)").
               build();

  // Customize Chart
  Chart.getStyler().setPlotBackgroundColor(ChartColor.getAWTColor
  (ChartColor.GREY));
  Chart.getStyler().setPlotGridLinesColor(new Color(255, 255, 255));
  Chart.getStyler().setChartBackgroundColor(Color.WHITE);
  Chart.getStyler().setLegendBackgroundColor(Color.PINK);
  Chart.getStyler().setChartFontColor(Color.MAGENTA);
  Chart.getStyler().setChartTitleBoxBackgroundColor(new Color(0, 222, 0));
  Chart.getStyler().setChartTitleBoxVisible(true);
```

```
    Chart.getStyler().setChartTitleBoxBorderColor(Color.BLACK);
    Chart.getStyler().setPlotGridLinesVisible(true);
    Chart.getStyler().setAxisTickPadding(20);
    Chart.getStyler().setAxisTickMarkLength(15);
    Chart.getStyler().setPlotMargin(20);
    Chart.getStyler().setChartTitleVisible(false);
    Chart.getStyler().setChartTitleFont(new Font(Font.MONOSPACED,
    Font.BOLD, 24));
    Chart.getStyler().setLegendFont(new Font(Font.SERIF, Font.PLAIN, 18));
    Chart.getStyler().setLegendPosition(LegendPosition.InsideSE);
    Chart.getStyler().setLegendSeriesLineLength(12);
    Chart.getStyler().setAxisTitleFont(new Font(Font.SANS_SERIF,
    Font.ITALIC, 18));
    Chart.getStyler().setAxisTickLabelsFont(new Font(Font.SERIF,
    Font.PLAIN, 11));
    Chart.getStyler().setDatePattern("yyyy-MM");
    Chart.getStyler().setDecimalPattern("#0.00");
    //Chart.getStyler().setLocale(Locale.GERMAN);

    try
       {
         // Configuration

          // Set the mode to run the program

        workingModee = 1;   // Training mode

       if( workingMode == 1)
          {
            // Training mode
                        trainFileName = "C:/Book_Examples/Sample5_Train_
                        Norm.csv";
            chartTrainFileName = "XYLine_Sample5_Train_Chart_Results";
          }
```

```
else
  {
    // Testing mode
                intNumberOfRecordsInTestFile = 3;
    testFileName = "C:/Book_Examples/Sample2_Norm.csv";
    chartTestFileName = "XYLine_Test_Results_Chart";
  }

 // Common part of config data
 networkFileName = "C:/Book_Examples/Sample5_Saved_Network_File.csv";
 numberOfInputNeurons = 1;
 numberOfOutputNeurons = 1;

 // Check the working mode to run

 if(workingMode == 1)
  {
    // Training mode
    File file1 = new File(chartTrainFileName);
    File file2 = new File(networkFileName);

    if(file1.exists())
      file1.delete();

    if(file2.exists())
      file2.delete();

    returnCode = 0;    // Clear the error Code

    do
     {
       returnCode = trainValidateSaveNetwork();
     }  while (returnCode > 0);
  }
```

```
        else
         {
            // Test mode
            loadAndTestNetwork();
         }
       }
      catch (Throwable t)
        {
          t.printStackTrace();
          System.exit(1);
        }
       finally
        {
          Encog.getInstance().shutdown();
        }

   Encog.getInstance().shutdown();
   return Chart;

}  // End of the method

// =======================================================
// Load CSV to memory.
// @return The loaded dataset.
// =======================================================
public static MLDataSet loadCSV2Memory(String filename, int input,
int ideal, boolean headers, CSVFormat format, boolean significance)
  {
     DataSetCODEC codec = new CSVDataCODEC(new File(filename), format,
     headers, input, ideal, significance);
     MemoryDataLoader load = new MemoryDataLoader(codec);
     MLDataSet dataset = load.external2Memory();
     return dataset;
  }
```

```
// =======================================================
//  The main method.
//  @param Command line arguments. No arguments are used.
// =======================================================
public static void main(String[] args)
 {
   ExampleChart<XYChart> exampleChart = new Sample5();
   XYChart Chart = exampleChart.getChart();
   new SwingWrapper<XYChart>(Chart).displayChart();
 } // End of the main method

//=====================================================================
// This method trains, Validates, and saves the trained network file
//=====================================================================
static public double trainValidateSaveNetwork()
 {
   // Load the training CSV file in memory
   MLDataSet trainingSet =
     loadCSV2Memory(trainFileName,numberOfInputNeurons,numberOfOutput
     Neurons,true,CSVFormat.ENGLISH,false);

   // create a neural network
   BasicNetwork network = new BasicNetwork();

   // Input layer
   network.addLayer(new BasicLayer(null,true,1));

   // Hidden layer
   network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
   network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
   network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
   network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
   network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
   network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
   network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
```

```
// Output layer
network.addLayer(new BasicLayer(new ActivationTANH(),false,1));

network.getStructure().finalizeStructure();
network.reset();

// train the neural network
final ResilientPropagation train = new ResilientPropagation
(network, trainingSet);

int epoch = 1;

do
 {
    train.iteration();
   System.out.println("Epoch #" + epoch + " Error:" + train.getError());

   epoch++;

   if (epoch >= 11000 && network.calculateError(trainingSet) > 0.00225)
       {
        returnCode = 1;

        System.out.println("Try again");
        return returnCode;
       }
 } while(train.getError() > 0.0022);

// Save the network file
EncogDirectoryPersistence.saveObject(new File(networkFileName),network);

System.out.println("Neural Network Results:");
double sumNormDifferencePerc = 0.00;
double averNormDifferencePerc = 0.00;
double maxNormDifferencePerc = 0.00;
int m = 0;
double xPointer = 0.00;
```

```
for(MLDataPair pair: trainingSet)
  {
      m++;
      xPointer++;

       final MLData output = network.compute(pair.getInput());

       MLData inputData = pair.getInput();
       MLData actualData = pair.getIdeal();
       MLData predictData = network.compute(inputData);

       // Calculate and print the results
       normInputXPointValue = inputData.getData(0);
       normTargetXPointValue = actualData.getData(0);
       normPredictXPointValue = predictData.getData(0);

       denormInputXPointValue = ((minXPointDl - maxXPointDh)*
       normInputXPointValue - Nh*minXPointDl + maxXPointDh *Nl)/
       (Nl - Nh);

       denormTargetXPointValue =((minTargetValueDl - maxTargetValueDh)*
       normTargetXPointValue - Nh*minTargetValueDl + maxTargetValueDh*Nl)/
       (Nl - Nh);
       denormPredictXPointValue =((minTargetValueDl - maxTargetValueDh)*
       normPredictXPointValue - Nh*minTargetValueDl + maxTargetValue
       Dh*Nl)/(Nl - Nh);

       valueDifference =
         Math.abs(((denormTargetXPointValue - denormPredictXPointValue)/
                 denormTargetXPointValue)*100.00);

       System.out.println ("RecordNumber = " + m + "  denormTargetX
       PointValue = " + denormTargetXPointValue + "  denormPredictX
       PointValue = " + denormPredictXPointValue + "  value
       Difference = " + valueDifference);

       sumNormDifferencePerc = sumNormDifferencePerc + valueDifference;
```

```
            if (valueDifference > maxNormDifferencePerc)
              maxNormDifferencePerc = valueDifference;

            xData.add(xPointer);
            yData1.add(denormTargetXPointValue);
            yData2.add(denormPredictXPointValue);

        }   // End for pair loop

        XYSeries series1 = Chart.addSeries("Actual data", xData, yData1);
        XYSeries series2 = Chart.addSeries("Predict data", xData, yData2);

        series1.setLineColor(XChartSeriesColors.BLUE);
        series2.setMarkerColor(Color.ORANGE);
        series1.setLineStyle(SeriesLines.SOLID);
        series2.setLineStyle(SeriesLines.SOLID);

        try
         {
           //Save the chart image
           BitmapEncoder.saveBitmapWithDPI(Chart, chartTrainFileName,
           BitmapFormat.JPG, 100);
           System.out.println ("Train Chart file has been saved") ;
         }
       catch (IOException ex)
        {
         ex.printStackTrace();
         System.exit(3);
        }

        // Finally, save this trained network
        EncogDirectoryPersistence.saveObject(new File(networkFileName),
        network);
        System.out.println ("Train Network has been saved") ;

        averNormDifferencePerc  = sumNormDifferencePerc/1000.00;

        System.out.println(" ");
        System.out.println("maxNormDifferencePerc = " + maxNormDifference
        Perc + "  averNormDifferencePerc = " + averNormDifferencePerc);
```

```
    returnCode = 0.00;
    return returnCode;

 }   // End of the method

//================================================
// This method load and test the trained network
//================================================
static public void loadAndTestNetwork()
 {
  System.out.println("Testing the networks results");

  List<Double> xData = new ArrayList<Double>();
  List<Double> yData1 = new ArrayList<Double>();
  List<Double> yData2 = new ArrayList<Double>();

  double targetToPredictPercent = 0;
  double maxGlobalResultDiff = 0.00;
  double averGlobalResultDiff = 0.00;
  double sumGlobalResultDiff = 0.00;
  double maxGlobalIndex = 0;
  double normInputXPointValueFromRecord = 0.00;
  double normTargetXPointValueFromRecord = 0.00;
  double normPredictXPointValueFromRecord = 0.00;

  BasicNetwork network;

  maxGlobalResultDiff = 0.00;
  averGlobalResultDiff = 0.00;
  sumGlobalResultDiff = 0.00;

  // Load the test dataset into memory
  MLDataSet testingSet = loadCSV2Memory(testFileName, numberOfInput
  Neurons,numberOfOutputNeurons,true,CSVFormat.ENGLISH,false);
```

```
// Load the saved trained network
network = (BasicNetwork)EncogDirectoryPersistence.loadObject(new File
(networkFileName));

int i = - 1; // Index of the current record
double xPoint = -0.00;

for (MLDataPair pair:  testingSet)
 {
      i++;
      xPoint = xPoint + 2.00;

      MLData inputData = pair.getInput();
      MLData actualData = pair.getIdeal();
      MLData predictData = network.compute(inputData);

      normInputXPointValueFromRecord = inputData.getData(0);
      normTargetXPointValueFromRecord = actualData.getData(0);
      normPredictXPointValueFromRecord = predictData.getData(0);

      // De-normalize them
      denormInputXPointValue = ((minXPointDl - maxXPointDh)*
      normInputXPointValueFromRecord - Nh*minXPointDl +
      maxXPointDh*Nl)/(Nl - Nh);

      denormTargetXPointValue = ((minTargetValueDl - maxTargetValueDh)*
      normTargetXPointValueFromRecord - Nh*minTargetValueDl +
      maxTargetValueDh*Nl)/(Nl - Nh);

      denormPredictXPointValue =((minTargetValueDl - maxTargetValueDh)*
      normPredictXPointValueFromRecord - Nh*minTargetValueDl +
      maxTargetValueDh*Nl)/(Nl - Nh);

      targetToPredictPercent = Math.abs((denormTargetXPointValue -
      denormPredictXPointValue)/denormTargetXPointValue*100);

      System.out.println("xPoint = " + xPoint +  "  denormTargetX
      PointValue = " + denormTargetXPointValue + "  denormPredictX
      PointValue = " + denormPredictXPointValue + "   targetToPredict
      Percent = " + targetToPredictPercent);
```

```
      if (targetToPredictPercent > maxGlobalResultDiff)
      maxGlobalResultDiff = targetToPredictPercent;

      sumGlobalResultDiff = sumGlobalResultDiff + targetToPredictPercent;

      // Populate chart elements
      xData.add(xPoint);
      yData1.add(denormTargetXPointValue);
      yData2.add(denormPredictXPointValue);

 }  // End for pair loop

// Print the max and average results
System.out.println(" ");
averGlobalResultDiff = sumGlobalResultDiff/intNumberOfRecordsInTestFile;

System.out.println("maxGlobalResultDiff = " + maxGlobalResultDiff +
"  i = " + maxGlobalIndex);
System.out.println("averGlobalResultDiff = " + averGlobalResultDiff);

// All testing batch files have been processed
XYSeries series1 = Chart.addSeries("Actual", xData, yData1);
XYSeries series2 = Chart.addSeries("Predicted", xData, yData2);

series1.setLineColor(XChartSeriesColors.BLUE);
series2.setMarkerColor(Color.ORANGE);
series1.setLineStyle(SeriesLines.SOLID);
series2.setLineStyle(SeriesLines.SOLID);

// Save the chart image
try
 {
    BitmapEncoder.saveBitmapWithDPI(Chart, chartTestFileName,
    BitmapFormat.JPG, 100);
 }
catch (Exception bt)
 {
   bt.printStackTrace();
 }
```

```
      System.out.println ("The Chart has been saved");
      System.out.println("End of testing for test records");

   } // End of the method

} // End of the class
```

Code Fragments for the Training Process

The training method is called in a loop until it successfully clears the error limit. You load the normalized training file and then create the network with one input layer (one neuron), seven hidden layers (each with five neurons), and the output layer (one neuron). Next, you train the network by looping over the epochs until the network error clears the error limit. At that point, you exit the loop. The network is trained, and you save it on disk (it will be used by the testing method). Listing 8-2 shows the fragment of the training method.

Listing 8-2. Fragment of the Code of the Training Method

```
// Load the training CSV file in memory
MLDataSet trainingSet =
loadCSV2Memory(trainFileName,numberOfInputNeurons,numberOfOutputNeurons,
     true,CSVFormat.ENGLISH,false);

// create a neural network
BasicNetwork network = new BasicNetwork();

// Input layer
network.addLayer(new BasicLayer(null,true,1));

// Hidden layer
network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
```

```
// Output layer
network.addLayer(new BasicLayer(new ActivationTANH(),false,1));

network.getStructure().finalizeStructure();
network.reset();

// train the neural network
final ResilientPropagation train = new ResilientPropagation(network,
trainingSet);

int epoch = 1;

do
  {
     train.iteration();
     System.out.println("Epoch #" + epoch + " Error:" + train.getError());

     epoch++;

     if (epoch >= 11000 && network.calculateError(trainingSet) > 0.00225)
     // 0.0221   0.00008
       {
          returnCode = 1;
          System.out.println("Try again");
          return returnCode;
        }
 } while(train.getError() > 0.0022);

// Save the network file
EncogDirectoryPersistence.saveObject(new File(networkFileName),network);
```

Next you loop over the pair data set and retrieve from the network the input, actual, and predicted values for each record. You then denormalize the retrieved values, put them in the log, and populate the chart data.

```
int m = 0;
double xPointer = 0.00;

for(MLDataPair pair: trainingSet)
  {
      m++;
      xPointer++;
```

```
        final MLData output = network.compute(pair.getInput());

        MLData inputData = pair.getInput();
        MLData actualData = pair.getIdeal();
        MLData predictData = network.compute(inputData);

         // Calculate and print the results
         normInputXPointValue = inputData.getData(0);
         normTargetXPointValue = actualData.getData(0);
         normPredictXPointValue = predictData.getData(0);

         denormInputXPointValue = ((minXPointDl - maxXPointDh)*normInputX
         PointValue - Nh*minXPointDl + maxXPointDh *Nl)/(Nl - Nh);

         denormTargetXPointValue =((minTargetValueDl - maxTargetValueDh)*
         normTargetXPointValue - Nh*minTargetValueDl + maxTargetValueDh*Nl)/
         (Nl - Nh);
         denormPredictXPointValue =((minTargetValueDl - maxTargetValueDh)*
         normPredictXPointValue - Nh*minTargetValueDl + maxTargetValueDh*Nl)/
         (Nl - Nh);

         valueDifference = Math.abs(((denormTargetXPointValue - denormPredictX
         PointValue)/denormTargetXPointValue)*100.00);

         System.out.println ("RecordNumber = " + m + "  denormTargetX
         PointValue = " + denormTargetXPointValue + "  denormPredictXPoint
         Value = " + denormPredictXPointValue + "  valueDifference = " +
         valueDifference);

        sumNormDifferencePerc = sumNormDifferencePerc + valueDifference;

        if (valueDifference > maxNormDifferencePerc)
        maxNormDifferencePerc = valueDifference;

        xData.add(xPointer);
        yData1.add(denormTargetXPointValue);
        yData2.add(denormPredictXPointValue);

    }   // End for pair loop
```

Finally, you calculate the average and maximum values of the results and save the chart file.

```
XYSeries series1 = Chart.addSeries("Actual data", xData, yData1);
XYSeries series2 = Chart.addSeries("Predict data", xData, yData2);

   series1.setLineColor(XChartSeriesColors.BLUE);
   series2.setMarkerColor(Color.ORANGE);
   series1.setLineStyle(SeriesLines.SOLID);
   series2.setLineStyle(SeriesLines.SOLID);

   try
    {
       //Save the chart image
       BitmapEncoder.saveBitmapWithDPI(Chart, chartTrainFileName,
       BitmapFormat.JPG, 100);
       System.out.println ("Train Chart file has been saved") ;
     }
   catch (IOException ex)
    {
       ex.printStackTrace();
       System.exit(3);
    }

    // Save this trained network
    EncogDirectoryPersistence.saveObject(new File(networkFileName),network);
    System.out.println ("Train Network has been saved") ;

    averNormDifferencePerc  = sumNormDifferencePerc/1000.00;

    System.out.println(" ");
    System.out.println("maxNormDifferencePerc = " + maxNormDifferencePerc + "
    averNormDifferencePerc = " + averNormDifferencePerc);

    returnCode = 0.00;
    return returnCode;

}   // End of the method
```

Unsatisfactory Training Results

Listing 8-3 shows the ending fragment of the training results.

Listing 8-3. Ending Fragment of the Training Results

```
RecordNumber =  983  TargetValue = 1036.19  PredictedValue = 930.03102
DiffPerc = 10.24513
RecordNumber =  984  TargetValue = 1095.63  PredictedValue = 915.36958
DiffPerc = 16.45267
RecordNumber =  985  TargetValue = 968.75   PredictedValue = 892.96942
DiffPerc = 7.822511
RecordNumber =  986  TargetValue = 896.24   PredictedValue = 863.64775
DiffPerc = 3.636554
RecordNumber =  987  TargetValue = 903.25   PredictedValue = 829.19287
DiffPerc = 8.198962
RecordNumber =  988  TargetValue = 825.88   PredictedValue = 791.96691
DiffPerc = 4.106298
RecordNumber =  989  TargetValue = 735.09   PredictedValue = 754.34279
DiffPerc = 2.619107
RecordNumber =  990  TargetValue = 797.87   PredictedValue = 718.23458
DiffPerc = 9.981002
RecordNumber =  991  TargetValue = 672.81   PredictedValue = 684.88576
DiffPerc = 1.794825
RecordNumber =  992  TargetValue = 619.14   PredictedValue = 654.90309
DiffPerc = 5.776254
RecordNumber =  993  TargetValue = 619.32   PredictedValue = 628.42044
DiffPerc = 1.469424
RecordNumber =  994  TargetValue = 590.47   PredictedValue = 605.28210
DiffPerc = 2.508528
RecordNumber =  995  TargetValue = 547.28   PredictedValue = 585.18808
DiffPerc = 6.926634
RecordNumber =  996  TargetValue = 514.62   PredictedValue = 567.78844
DiffPerc = 10.33159
RecordNumber =  997  TargetValue = 455.4    PredictedValue = 552.73603
DiffPerc = 21.37374
```

```
RecordNumber =  998  TargetValue = 470.43   PredictedValue = 539.71156
DiffPerc = 14.72728
RecordNumber =  999  TargetValue = 480.28   PredictedValue = 528.43269
DiffPerc = 10.02596
RecordNumber = 1000  TargetValue = 496.77   PredictedValue = 518.65485
DiffPerc = 4.405429

maxNormDifferencePerc = 97.69386964911284
averNormDifferencePerc = 7.232624870097155
```

This approximation is low quality. Even with the network being well optimized, the average approximation error for all records is more than 8 percent, and the maximum approximation error (the worst approximated record) is more than 97 percent. Such function approximation is certainly not usable. Figure 8-3 shows the chart of the approximation results.

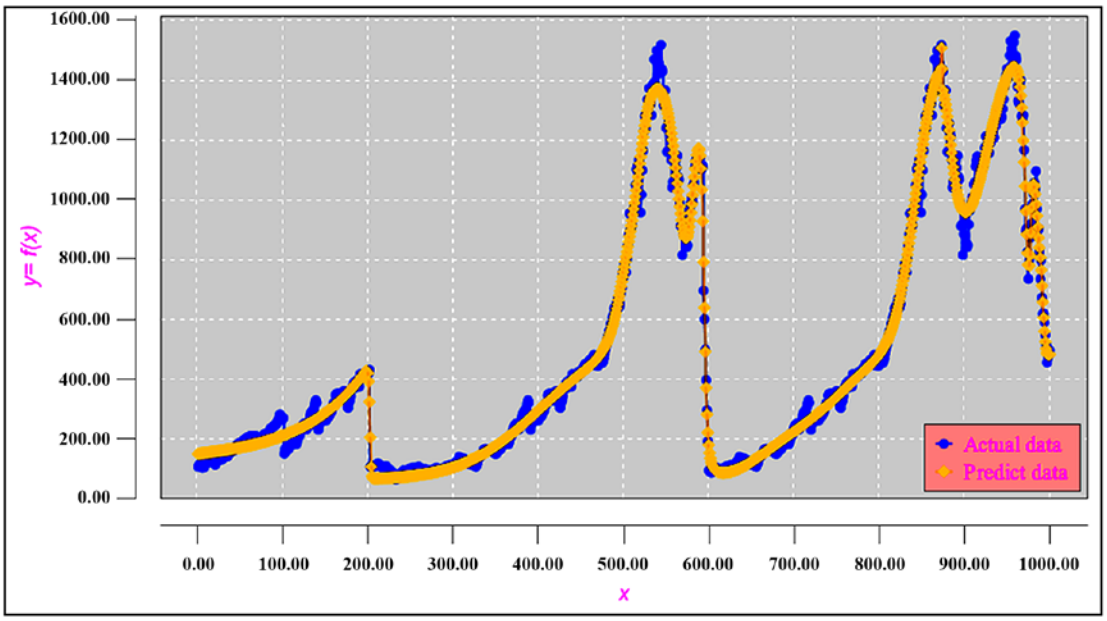


Figure 8-3. *Low-quality function approximation*

I knew that this approximation would not work and stated this at the beginning of the example. However, it was deliberately done to demonstrate the point. Now, I will show how this noncontinuous function can be successfully approximated using a neural network.

The problem with this function approximation is the function topology (that has sudden jumps or drops of the function values at certain points). So, you will break the input file into a series of one-record input files that are called *micro-batches*. This is similar to the batch training, but here you actively control the batch size. By doing this, you eliminate the negative impact of the difficult function topology. After breaking up the data set, every record will be isolated and not linked to the previous or next function value. Breaking the input file into micro-batches creates 1,000 input files, which the network processes individually. You link each trained network with the record it represents. During the validation and testing processes, the logic finds the trained network that best matches the first field of the corresponding testing or validation record.

Approximating the Noncontinuous Function Using the Micro-Bach Method

Let's break the normalized training data set into micro-batches. Each micro-batch data set should contain the label record and one record from the original file to be processed. Table 8-3 shows how a micro-batch data set looks.

Table 8-3. *Micro-Batch File*

xPoint	Function Value
-1	-0.938458442

Here you write a simple program to break the normalized training data set into micro-batches. As a result of executing this program, you created 999 micro-batch data sets (numbered from 0 to 998). Figure 8-4 shows a fragment of the list of the micro-batch data sets.

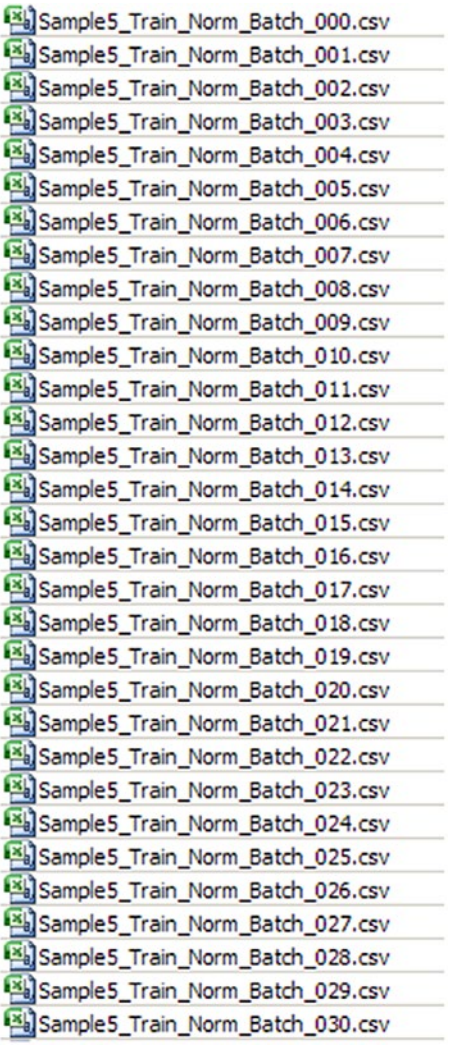


Figure 8-4. *Fragment of list of normalized training micro-batch data sets*

This set of the micro-batch data sets is now the input for training the network.

Program Code for Micro-Batch Processing

Listing 8-4 shows the program code.

Listing 8-4. Program Code

```
// ============================================================
// Approximation of non-continuous function using the micro-batch method.
// The input is the normalized set of micro-batch files  (each micro-batch
// includes a single day record).
// Each record consists of:
// - normDayValue
// - normTargetValue
//
// The number of inputLayer neurons is 12
// The number of outputLayer neurons is 1
//
// The difference of this program is that it independently trains many
// single-day networks. That allows training each daily network using the
// best value of weights/biases parameters, therefore achieving the best
// optimization results for each year.
//
// Each network is saved on disk and a map is created to link each saved
   trained
// network with the corresponding training micro-batch file.
// ============================================================
package sample5_microbatches;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.*;
import java.util.Properties;
import java.time.YearMonth;
import java.awt.Color;
```

```
import java.awt.Font;
import java.io.BufferedReader;
import java.time.Month;
import java.time.ZoneId;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;
import java.util.Locale;
import java.util.Properties;
import org.encog.Encog;
import org.encog.engine.network.activation.ActivationTANH;
import org.encog.engine.network.activation.ActivationReLU;
import org.encog.ml.data.MLData;
import org.encog.ml.data.MLDataPair;
import org.encog.ml.data.MLDataSet;
import org.encog.ml.data.buffer.MemoryDataLoader;
import org.encog.ml.data.buffer.codec.CSVDataCODEC;
import org.encog.ml.data.buffer.codec.DataSetCODEC;
import org.encog.neural.networks.BasicNetwork;
import org.encog.neural.networks.layers.BasicLayer;
import org.encog.neural.networks.training.propagation.
resilient.  ResilientPropagation;
import org.encog.persist.EncogDirectoryPersistence;
import org.encog.util.csv.CSVFormat;

import org.knowm.xchart.SwingWrapper;
import org.knowm.xchart.XYChart;
import org.knowm.xchart.XYChartBuilder;
import org.knowm.xchart.XYSeries;
import org.knowm.xchart.demo.charts.ExampleChart;
import org.knowm.xchart.style.Styler.LegendPosition;
import org.knowm.xchart.style.colors.ChartColor;
import org.knowm.xchart.style.colors.XChartSeriesColors;
import org.knowm.xchart.style.lines.SeriesLines;
import org.knowm.xchart.style.markers.SeriesMarkers;
import org.knowm.xchart.BitmapEncoder;
```

```
import org.knowm.xchart.BitmapEncoder.BitmapFormat;
import org.knowm.xchart.QuickChart;
import org.knowm.xchart.SwingWrapper;

public class Sample5_Microbatches implements ExampleChart<XYChart>
{
   // Normalization parameters

   // Normalizing interval
   static double Nh =  1;
   static double Nl = -1;

  static double inputDayDh = 1000.00;
  static double inputDayDl = 1.00;
  static double targetFunctValueDiffPercDh = 1600.00;
  static double targetFunctValueDiffPercDl = 60.00;
   static String cvsSplitBy = ",";
   static Properties prop = null;
   static String strWorkingMode;
   static String strNumberOfBatchesToProcess;
   static String strTrainFileNameBase;
   static String strTestFileNameBase;
   static String strSaveTrainNetworkFileBase;
   static String strSaveTestNetworkFileBase;
   static String strValidateFileName;
   static String strTrainChartFileName;
   static String strTestChartFileName;
   static String strFunctValueTrainFile;
   static String strFunctValueTestFile;
   static int intDayNumber;
   static double doubleDayNumber;
   static int intWorkingMode;
   static int numberOfTrainBatchesToProcess;
   static int numberOfTestBatchesToProcess;
   static int intNumberOfRecordsInTrainFile;
   static int intNumberOfRecordsInTestFile;
   static int intNumberOfRowsInBatches;
```

```
 static int intInputNeuronNumber;
 static int intOutputNeuronNumber;
 static String strOutputFileName;
 static String strSaveNetworkFileName;
 static String strDaysTrainFileName;
 static XYChart Chart;
 static String iString;
 static double inputFunctValueFromFile;
 static double targetToPredictFunctValueDiff;
 static int[] returnCodes  = new int[3];
 static List<Double> xData = new ArrayList<Double>();
 static List<Double> yData1 = new ArrayList<Double>();
 static List<Double> yData2 = new ArrayList<Double>();
 static double[] DaysyearDayTraining = new double[1200];
 static String[] strTrainingFileNames = new String[1200];
 static String[] strTestingFileNames = new String[1200];
 static String[] strSaveTrainNetworkFileNames = new String[1200];
 static double[] linkToSaveNetworkDayKeys = new double[1200];
 static double[] linkToSaveNetworkTargetFunctValueKeys = new double[1200];
 static double[] arrTrainFunctValues = new double[1200];
 static double[] arrTestFunctValues = new double[1200];

@Override
public XYChart getChart()
 {
   // Create Chart

  Chart = new XYChartBuilder().width(900).height(500).title(getClass().
  getSimpleName()).xAxisTitle("day").yAxisTitle("y=f(day)").build();
  // Customize Chart
  Chart.getStyler().setPlotBackgroundColor(ChartColor.getAWTColor(Chart
  Color.GREY));
  Chart.getStyler().setPlotGridLinesColor(new Color(255, 255, 255));
  Chart.getStyler().setChartBackgroundColor(Color.WHITE);
  Chart.getStyler().setLegendBackgroundColor(Color.PINK);
  Chart.getStyler().setChartFontColor(Color.MAGENTA);
  Chart.getStyler().setChartTitleBoxBackgroundColor(new Color(0, 222, 0));
```

```
    Chart.getStyler().setChartTitleBoxVisible(true);
    Chart.getStyler().setChartTitleBoxBorderColor(Color.BLACK);
    Chart.getStyler().setPlotGridLinesVisible(true);
    Chart.getStyler().setAxisTickPadding(20);
    Chart.getStyler().setAxisTickMarkLength(15);
    Chart.getStyler().setPlotMargin(20);
    Chart.getStyler().setChartTitleVisible(false);
    Chart.getStyler().setChartTitleFont(new Font(Font.MONOSPACED,
    Font.BOLD, 24));
    Chart.getStyler().setLegendFont(new Font(Font.SERIF, Font.PLAIN, 18));
    Chart.getStyler().setLegendPosition(LegendPosition.OutsideE);
    Chart.getStyler().setLegendSeriesLineLength(12);
    Chart.getStyler().setAxisTitleFont(new Font(Font.SANS_SERIF,
    Font.ITALIC, 18));
    Chart.getStyler().setAxisTickLabelsFont(new Font(Font.SERIF,
    Font.PLAIN, 11));
    //Chart.getStyler().setDayPattern("yyyy-MM");
    Chart.getStyler().setDecimalPattern("#0.00");

    // Config data
    // Training mode
    intWorkingMode = 0;

    // Testing mode
    numberOfTrainBatchesToProcess = 1000;
    numberOfTestBatchesToProcess = 999;
    intNumberOfRowsInBatches = 1;
    intInputNeuronNumber = 1;
    intOutputNeuronNumber = 1;
    strTrainFileNameBase = "C:/My_Neural_Network_Book/Temp_Files/Sample5_
    Train_Norm_Batch_";
    strTestFileNameBase = "C:/My_Neural_Network_Book/Temp_Files/Sample5_
    Test_Norm_Batch_";
    strSaveTrainNetworkFileBase = "C:/Book_Examples/Sample5_Save_Network_
    Batch_";
```

```
strTrainChartFileName = "C:/Book_Examples/Sample5_Chart_Train_File_
Microbatch.jpg";
strTestChartFileName = "C:/Book_Examples/Sample5_Chart_Test_File_
Microbatch.jpg";

// Generate training batch file names and the corresponding saveNetwork
   file names
intDayNumber = -1;  // Day number for the chart

for (int i = 0; i < numberOfTrainBatchesToProcess; i++)
 {
   intDayNumber++;
    iString = Integer.toString(intDayNumber);

   if (intDayNumber >= 10 & intDayNumber < 100  )
    {
      strOutputFileName = strTrainFileNameBase + "0" + iString + ".csv";
      strSaveNetworkFileName = strSaveTrainNetworkFileBase + "0" +
        iString + ".csv";
    }
   else
    {
      if (intDayNumber < 10)
       {
         strOutputFileName = strTrainFileNameBase + "00" + iString +
           ".csv";
         strSaveNetworkFileName = strSaveTrainNetworkFileBase + "00" +
           iString + ".csv";
       }
      else
       {
         strOutputFileName = strTrainFileNameBase + iString + ".csv";

         strSaveNetworkFileName = strSaveTrainNetworkFileBase +
           iString + ".csv";
       }
    }
```

```
            strTrainingFileNames[intDayNumber] = strOutputFileName;
            strSaveTrainNetworkFileNames[intDayNumber] = strSaveNetwork
            FileName;

        }  // End the FOR loop

        // Build the array linkToSaveNetworkFunctValueDiffKeys
        String tempLine;
        double tempNormFunctValueDiff = 0.00;
        double tempNormFunctValueDiffPerc = 0.00;
        double tempNormTargetFunctValueDiffPerc = 0.00;
        String[] tempWorkFields;
        try
         {
            intDayNumber = -1;  // Day number for the chart

            for (int m = 0; m < numberOfTrainBatchesToProcess; m++)
              {
                 intDayNumber++;
                 BufferedReader br3 = new BufferedReader(newFileReader
                         (strTrainingFileNames[intDayNumber]));
                 tempLine = br3.readLine();

                 // Skip the label record and zero batch record
                 tempLine = br3.readLine();

                 // Break the line using comma as separator
                 tempWorkFields = tempLine.split(cvsSplitBy);
                 tempNormFunctValueDiffPerc = Double.parseDouble
                 (tempWorkFields[0]);
                 tempNormTargetFunctValueDiffPerc = Double.parseDouble
                 (tempWorkFields[1]);
                 linkToSaveNetworkDayKeys[intDayNumber] = tempNormFunctValue
                 DiffPerc;
                 linkToSaveNetworkTargetFunctValueKeys[intDayNumber] =
                    tempNormTargetFunctValueDiffPerc;
        }  // End the FOR loop
```

```
  // Generate testing batche file names
  if(intWorkingMode == 1)
   {
     intDayNumber = -1;

     for (int i = 0; i < numberOfTestBatchesToProcess; i++)
      {
        intDayNumber++;
        iString = Integer.toString(intDayNumber);

        // Construct the testing batch names
        if (intDayNumber >= 10 & intDayNumber < 100  )
         {
           strOutputFileName = strTestFileNameBase + "0" +
             iString + ".csv";
         }
        else
         {
           if (intDayNumber < 10)
            {
              strOutputFileName = strTestFileNameBase + "00" +
                iString + ".csv";
            }
           else
            {
              strOutputFileName = strTestFileNameBase +
                iString + ".csv";
            }
         }

     strTestingFileNames[intDayNumber] = strOutputFileName;

   }  // End the FOR loop

  }   // End of IF

}     // End for try
```

```
catch (IOException io1)
 {
   io1.printStackTrace();
   System.exit(1);
 }

 // Load, train, and test Function Values file in memory
 //loadTrainFunctValueFileInMemory();

if(intWorkingMode == 0)
   {
     // Train mode
     int paramErrorCode;
     int paramBatchNumber;
     int paramR;
     int paramDayNumber;
     int paramS;

     File file1 = new File(strTrainChartFileName);
     if(file1.exists())
        file1.delete();
     returnCodes[0] = 0;    // Clear the error Code
     returnCodes[1] = 0;    // Set the initial batch Number to 0;
     returnCodes[2] = 0;    // Day number;
    do
     {
       paramErrorCode = returnCodes[0];
       paramBatchNumber = returnCodes[1];
       paramDayNumber = returnCodes[2];
       returnCodes =  trainBatches(paramErrorCode,paramBatchNumber,
       paramDayNumber);
     } while (returnCodes[0] > 0);

   }   // End the train logic
  else
   {
     // Testing mode
     File file2 = new File(strTestChartFileName);
```

```
        if(file2.exists())
          file2.delete();

        loadAndTestNetwork();

       // End the test logic
     }

     Encog.getInstance().shutdown();
     //System.exit(0);
     return Chart;

 }  // End of method

// =======================================================
// Load CSV to memory.
// @return The loaded dataset.
// =======================================================
public static MLDataSet loadCSV2Memory(String filename, int input, int
ideal, boolean headers, CSVFormat format, boolean significance)
  {
     DataSetCODEC codec = new CSVDataCODEC(new File(filename), format,
     headers, input, ideal, significance);
     MemoryDataLoader load = new MemoryDataLoader(codec);
     MLDataSet dataset = load.external2Memory();
     return dataset;
  }

// =======================================================
//  The main method.
//  @param Command line arguments. No arguments are used.
// =======================================================
public static void main(String[] args)
 {
  ExampleChart<XYChart> exampleChart = new Sample5_Microbatches();
   XYChart Chart = exampleChart.getChart();
   new SwingWrapper<XYChart>(Chart).displayChart();
 } // End of the main method
```

```
//=====================================================================
// This method trains batches as individual network1s
// saving them in separate trained datasets
//=====================================================================
static public int[] trainBatches(int paramErrorCode,int paramBatch
Number,int paramDayNumber)
  {
    int rBatchNumber;
    double targetToPredictFunctValueDiff = 0;
    double maxGlobalResultDiff = 0.00;
    double averGlobalResultDiff = 0.00;
    double sumGlobalResultDiff = 0.00;
    double normInputFunctValueDiffPercFromRecord = 0.00;
    double normTargetFunctValue1 = 0.00;
    double normPredictFunctValue1 = 0.00;
    double denormInputDayFromRecord1;
    double denormInputFunctValueDiffPercFromRecord;
    double denormTargetFunctValue1 = 0.00;
    double denormAverPredictFunctValue11 = 0.00;
    BasicNetwork network1 = new BasicNetwork();

    // Input layer
    network1.addLayer(new BasicLayer(null,true,intInputNeuronNumber));

    // Hidden layer.
    network1.addLayer(new BasicLayer(new ActivationTANH(),true,7));
    network1.addLayer(new BasicLayer(new ActivationTANH(),true,7));
    network1.addLayer(new BasicLayer(new ActivationTANH(),true,7));
    network1.addLayer(new BasicLayer(new ActivationTANH(),true,7));
    network1.addLayer(new BasicLayer(new ActivationTANH(),true,7));
    network1.addLayer(new BasicLayer(new ActivationTANH(),true,7));
    network1.addLayer(new BasicLayer(new ActivationTANH(),true,7));

   // Output layer
   network1.addLayer(new BasicLayer(new ActivationTANH(),false,
   intOutputNeuronNumber));
   network1.getStructure().finalizeStructure();
   network1.reset();
```

```
maxGlobalResultDiff = 0.00;
averGlobalResultDiff = 0.00;
sumGlobalResultDiff = 0.00;

// Loop over batches
intDayNumber = paramDayNumber;  // Day number for the chart

for (rBatchNumber = paramBatchNumber; rBatchNumber < numberOfTrain
BatchesToProcess; rBatchNumber++)
 {
   intDayNumber++;

  // Load the training file in memory
  MLDataSet trainingSet = loadCSV2Memory(strTrainingFileNames
  [rBatchNumber],intInputNeuronNumber,intOutputNeuronNumber, true,
  CSVFormat.ENGLISH,false);
 // train the neural network1
  ResilientPropagation train = new ResilientPropagation(network1,
  trainingSet);
  int epoch = 1;
 do
    {
 train.iteration();
 epoch++;

       for (MLDataPair pair11:  trainingSet)
       {
         MLData inputData1 = pair11.getInput();
         MLData actualData1 = pair11.getIdeal();
         MLData predictData1 = network1.compute(inputData1);

         // These values are Normalized as the whole input is
         normInputFunctValueDiffPercFromRecord = inputData1.getData(0);
         normTargetFunctValue1 = actualData1.getData(0);
         normPredictFunctValue1 = predictData1.getData(0);
```

```
            denormInputFunctValueDiffPercFromRecord =((inputDayDl -
            inputDayDh)*normInputFunctValueDiffPercFromRecord -
            Nh*inputDayDl + inputDayDh*Nl)/(Nl - Nh);
            denormTargetFunctValue1 = ((targetFunctValueDiffPercDl -
            targetFunctValueDiffPercDh)*normTargetFunctValue1 - Nh*target
            FunctValueDiffPercDl + targetFunctValueDiffPercDh*Nl)/(Nl - Nh);
            denormAverPredictFunctValue11 =((targetFunctValueDiffPercDl -
            targetFunctValueDiffPercDh)*normPredictFunctValue1 - Nh*
            targetFunctValueDiffPercDl + targetFunctValueDiffPercDh*Nl)/
            (Nl - Nh);

            targetToPredictFunctValueDiff = (Math.abs(denormTarget
            FunctValue1 - denormAverPredictFunctValue11)/denormTarget
            FunctValue1)*100;
          }

         if (epoch >= 1000 && targetToPredictFunctValueDiff > 0.0000071)
          {
            returnCodes[0] = 1;
            returnCodes[1] = rBatchNumber;
            returnCodes[2] = intDayNumber-1;

            return returnCodes;
           }

        } while(targetToPredictFunctValueDiff > 0.000007);

    // This batch is optimized

    // Save the network1 for the current batch
    EncogDirectoryPersistence.saveObject(newFile(strSaveTrainNetwork
    FileNames[rBatchNumber]),network1);

    // Get the results after the network1 optimization
    int i = - 1;

    for (MLDataPair pair1:  trainingSet)
     {
      i++;
      MLData inputData1 = pair1.getInput();
```

```
MLData actualData1 = pair1.getIdeal();
MLData predictData1 = network1.compute(inputData1);

// These values are Normalized as the whole input is
normInputFunctValueDiffPercFromRecord = inputData1.getData(0);
normTargetFunctValue1 = actualData1.getData(0);
normPredictFunctValue1 = predictData1.getData(0);

// De-normalize the obtained values
denormInputFunctValueDiffPercFromRecord =((inputDayDl - inputDayDh)*
normInputFunctValueDiffPercFromRecord - Nh*inputDayDl +
inputDayDh*Nl)/(Nl - Nh);

denormTargetFunctValue1 = ((targetFunctValueDiffPercDl - target
FunctValueDiffPercDh)*normTargetFunctValue1 - Nh*targetFunctValue
DiffPercDl + targetFunctValueDiffPercDh*Nl)/(Nl - Nh);

denormAverPredictFunctValue11 =((targetFunctValueDiffPercDl - target
FunctValueDiffPercDh)*normPredictFunctValue1 - Nh*targetFunctValue
DiffPercDl + targetFunctValueDiffPercDh*Nl)/(Nl - Nh);

//inputFunctValueFromFile = arrTrainFunctValues[rBatchNumber];

targetToPredictFunctValueDiff = (Math.abs(denormTargetFunctValue1 -
denormAverPredictFunctValue11)/denormTargetFunctValue1)*100;

  System.out.println("intDayNumber = " + intDayNumber +  "  target
  FunctionValue = " + denormTargetFunctValue1 + "  predictFunction
  Value = " + denormAverPredictFunctValue11 + "  valurDiff = " +
  targetToPredictFunctValueDiff);

if (targetToPredictFunctValueDiff > maxGlobalResultDiff)maxGlobal
ResultDiff =targetToPredictFunctValueDiff;

sumGlobalResultDiff = sumGlobalResultDiff +targetToPredictFunct
ValueDiff;

// Populate chart elements
doubleDayNumber = (double) rBatchNumber+1;
xData.add(doubleDayNumber);
```

```
        yData1.add(denormTargetFunctValue1);
        yData2.add(denormAverPredictFunctValue11);

       }  // End for FunctValue pair1 loop

    }  // End of the loop over batches

    sumGlobalResultDiff = sumGlobalResultDiff +targetToPredictFunctValue
    Diff;
    averGlobalResultDiff = sumGlobalResultDiff/numberOfTrainBatchesTo
    Process;

    // Print the max and average results

    System.out.println(" ");
    System.out.println(" ");
    System.out.println("maxGlobalResultDiff = " + maxGlobalResultDiff);
    System.out.println("averGlobalResultDiff = " + averGlobalResultDiff);

    XYSeries series1 = Chart.addSeries("Actual", xData, yData1);
    XYSeries series2 = Chart.addSeries("Predicted", xData, yData2);

    series1.setLineColor(XChartSeriesColors.BLUE);
    series2.setMarkerColor(Color.ORANGE);
    series1.setLineStyle(SeriesLines.SOLID);
    series2.setLineStyle(SeriesLines.SOLID);

    // Save the chart image
    try
      {
        BitmapEncoder.saveBitmapWithDPI(Chart, strTrainChartFileName,
        BitmapFormat.JPG, 100);
      }
     catch (Exception bt)
      {
        bt.printStackTrace();
      }

    System.out.println ("The Chart has been saved");
```

```
  returnCodes[0] = 0;
  returnCodes[1] = 0;
  returnCodes[2] = 0;
  return returnCodes;

 }  // End of method

//=========================================================================
// Load the previously saved trained network1 and tests it by
// processing the Test record
//=========================================================================
static public void loadAndTestNetwork()
 {
   System.out.println("Testing the network1s results");

   List<Double> xData = new ArrayList<Double>();
   List<Double> yData1 = new ArrayList<Double>();
   List<Double> yData2 = new ArrayList<Double>();

   double targetToPredictFunctValueDiff = 0;
   double maxGlobalResultDiff = 0.00;
   double averGlobalResultDiff = 0.00;
   double sumGlobalResultDiff = 0.00;
   double maxGlobalIndex = 0;
   double normInputDayFromRecord1 = 0.00;
   double normTargetFunctValue1 = 0.00;
   double normPredictFunctValue1 = 0.00;
   double denormInputDayFromRecord1 = 0.00;
   double denormTargetFunctValue1 = 0.00;
   double denormAverPredictFunctValue1 = 0.00;
   double normInputDayFromRecord2 = 0.00;
   double normTargetFunctValue2 = 0.00;
   double normPredictFunctValue2 = 0.00;
   double denormInputDayFromRecord2 = 0.00;
   double denormTargetFunctValue2 = 0.00;
   double denormAverPredictFunctValue2 = 0.00;
   double normInputDayFromTestRecord = 0.00;
   double denormInputDayFromTestRecord = 0.00;
```

```
double denormAverPredictFunctValue = 0.00;
double denormTargetFunctValueFromTestRecord = 0.00;
String tempLine;
String[] tempWorkFields;
double dayKeyFromTestRecord = 0.00;
double targetFunctValueFromTestRecord = 0.00;
double r1 = 0.00;
double r2 = 0.00;
BufferedReader br4;
BasicNetwork network1;
BasicNetwork network2;
int k1 = 0;
int k3 = 0;

try
 {
    // Process testing records
    maxGlobalResultDiff = 0.00;
    averGlobalResultDiff = 0.00;
    sumGlobalResultDiff = 0.00;

    for (k1 = 0; k1 < numberOfTestBatchesToProcess; k1++)
     {
        // Read the corresponding test micro-batch file.
        br4 = new BufferedReader(new FileReader(strTestingFileNames[k1]));
        tempLine = br4.readLine();

        // Skip the label record
        tempLine = br4.readLine();

        // Break the line using comma as separator
        tempWorkFields = tempLine.split(cvsSplitBy);

        dayKeyFromTestRecord = Double.parseDouble(tempWorkFields[0]);
        targetFunctValueFromTestRecord = Double.parseDouble(tempWork
        Fields[1]);
```

```
// De-normalize the dayKeyFromTestRecord
denormInputDayFromTestRecord = ((inputDayDl - inputDayDh)*day
KeyFromTestRecord - Nh*inputDayDl + inputDayDh*Nl)/(Nl - Nh);

// De-normalize the targetFunctValueFromTestRecord
denormTargetFunctValueFromTestRecord = ((targetFunctValue
DiffPercDl - targetFunctValueDiffPercDh)*targetFunctValueFrom
TestRecord - Nh*targetFunctValueDiffPercDl + targetFunctValue
DiffPercDh*Nl)/(Nl - Nh);

// Load the corresponding training micro-batch dataset in memory
MLDataSet trainingSet1 = loadCSV2Memory(strTrainingFile
Names[k1],intInputNeuronNumber,intOutputNeuronNumber,true,
CSVFormat.ENGLISH,false);
  network1 = (BasicNetwork)EncogDirectoryPersistence.
  loadObject(new File(strSaveTrainNetworkFileNames[k1]));

 // Get the results after the network1 optimization
 int iMax = 0;
 int i = - 1; // Index of the array to get results

 for (MLDataPair pair1:  trainingSet1)
  {
     i++;
     iMax = i+1;

     MLData inputData1 = pair1.getInput();
     MLData actualData1 = pair1.getIdeal();
     MLData predictData1 = network1.compute(inputData1);

     // These values are Normalized
     normInputDayFromRecord1 = inputData1.getData(0);
     normTargetFunctValue1 = actualData1.getData(0);
     normPredictFunctValue1 = predictData1.getData(0);

     // De-normalize the obtained values
     denormInputDayFromRecord1 = ((inputDayDl - inputDayDh)*
     normInputDayFromRecord1 - Nh*inputDayDl + inputDayDh*Nl)/
     (Nl - Nh);
```

```
        denormTargetFunctValue1 = ((targetFunctValueDiffPercDl -
        targetFunctValueDiffPercDh)*normTargetFunctValue1 - Nh*
        targetFunctValueDiffPercDl + targetFunctValueDiffPercDh*Nl)/
        (Nl - Nh);

        denormAverPredictFunctValue1 =((targetFunctValueDiffPercDl -
        targetFunctValueDiffPercDh)*normPredictFunctValue1 - Nh*
        targetFunctValueDiffPercDl + targetFunctValueDiffPercDh*Nl)/
        (Nl - Nh);

     }  // End for pair1

     // Now calculate everything again for the SaveNetwork (which
     // key is greater than dayKeyFromTestRecord value)in memory

    MLDataSet trainingSet2 = loadCSV2Memory(strTrainingFile
    Names[k1+1],intInputNeuronNumber,intOutputNeuronNumber,true,
    CSVFormat.ENGLISH,false);
    network2 = (BasicNetwork)EncogDirectoryPersistence.loadObject
    (new File(strSaveTrainNetworkFileNames[k1+1]));

     // Get the results after the network1 optimization
     iMax = 0;
     i = - 1;

    for (MLDataPair pair2:  trainingSet2)
     {
        i++;
        iMax = i+1;

        MLData inputData2 = pair2.getInput();
        MLData actualData2 = pair2.getIdeal();
        MLData predictData2 = network2.compute(inputData2);

        // These values are Normalized
        normInputDayFromRecord2 = inputData2.getData(0);
        normTargetFunctValue2 = actualData2.getData(0);
        normPredictFunctValue2 = predictData2.getData(0);
```

```
    // De-normalize the obtained values
    denormInputDayFromRecord2 = ((inputDayDl - inputDayDh)*
    normInputDayFromRecord2 - Nh*inputDayDl + inputDayDh*Nl)/
    (Nl - Nh);

    denormTargetFunctValue2 = ((targetFunctValueDiffPercDl -
    targetFunctValueDiffPercDh)*normTargetFunctValue2 - Nh*target
    FunctValueDiffPercDl + targetFunctValueDiffPercDh*Nl)/
    (Nl - Nh);

    denormAverPredictFunctValue2 =((targetFunctValueDiffPercDl -
    targetFunctValueDiffPercDh)*normPredictFunctValue2 -
    Nh*targetFunctValueDiffPercDl + targetFunctValueDiffPercDh
    *Nl)/(Nl - Nh);
}  // End for pair1 loop

// Get the average of the denormAverPredictFunctValue1 and
denormAverPredictFunctValue2 denormAverPredictFunctValue =
(denormAverPredictFunctValue1 + denormAverPredictFunctValue2)/2;

targetToPredictFunctValueDiff =
  (Math.abs(denormTargetFunctValueFromTestRecord - denormAver
  PredictFunctValue)/denormTargetFunctValueFromTestRecord)*100;

 System.out.println("Record Number = " + k1 + "  DayNumber =
 " + denormInputDayFromTestRecord + "  denormTargetFunctValue
 FromTestRecord = " + denormTargetFunctValueFromTestRecord +
 "  denormAverPredictFunctValue = " + denormAverPredict
 FunctValue + "  valurDiff = " + targetToPredictFunctValueDiff);
 if (targetToPredictFunctValueDiff > maxGlobalResultDiff)
  {
    maxGlobalIndex = iMax;
    maxGlobalResultDiff =targetToPredictFunctValueDiff;
  }

 sumGlobalResultDiff = sumGlobalResultDiff + targetToPredict
 FunctValueDiff;
```

```
            // Populate chart elements

            xData.add(denormInputDayFromTestRecord);
            yData1.add(denormTargetFunctValueFromTestRecord);
            yData2.add(denormAverPredictFunctValue);

        }   // End of loop using k1

       // Print the max and average results

       System.out.println(" ");

       averGlobalResultDiff = sumGlobalResultDiff/numberOfTestBatchesToProcess;

       System.out.println("maxGlobalResultDiff = " + maxGlobalResultDiff +
         "  i = " + maxGlobalIndex);
       System.out.println("averGlobalResultDiff = " + averGlobalResultDiff);

    }     // End of TRY
   catch (IOException e1)
    {
          e1.printStackTrace();
    }

    // All testing batch files have been processed
    XYSeries series1 = Chart.addSeries("Actual", xData, yData1);
    XYSeries series2 = Chart.addSeries("Forecasted", xData, yData2);

    series1.setLineColor(XChartSeriesColors.BLUE);
    series2.setMarkerColor(Color.ORANGE);
    series1.setLineStyle(SeriesLines.SOLID);
    series2.setLineStyle(SeriesLines.SOLID);

    // Save the chart image
    try
     {
       BitmapEncoder.saveBitmapWithDPI(Chart, strTrainChartFileName,
       BitmapFormat.JPG, 100);
     }
```

```
        catch (Exception bt)
         {
           bt.printStackTrace();
         }

        System.out.println ("The Chart has been saved");
        System.out.println("End of testing for mini-batches training");

    } // End of the method

} // End of the  Encog class
```

The processing logic is quite different in this program. Let's start from the `getChart()` method. Apart from the usual statements needed by the XChart package, you generate here the names for the training micro-batches and save-network files. The generated file names for micro-batches must match the micro-batch file names being prepared on disk when you broke the normalized training file into micro-batches.

The names for saved-network files have a corresponding structure. These generated names will be used by the training method to save trained networks corresponding to micro-batches on disk. The generated names are saved in two arrays called `strTrainingFileNames[]` and `strSaveTrainNetworkFileNames[]`.

Figure 8-5 shows the fragment of the generated saved network.

Figure 8-5. *Fragment of the generated save-network files*

Next, you generate and populate two arrays called `linkToSaveNetworkDayKeys[]` and `linkToSaveNetworkTargetFunctValueKeys[]`. For each consecutive day, you populate the `linkToSaveNetworkDayKeys[]` array with the `field1` value from the training micro-batch records. You populate the `linkToSaveNetworkTargetFunctValueKeys[]` array with the names of the corresponding saveNetworkFiles on disk. Therefore, those two arrays hold the link between the micro-batch data set and the corresponding save-network data set.

The program also generates the names of the testing micro-batch files, similar to the generated names for the training micro-batch files. When all this is done, you call the `loadTrainFunctValueFileInMemory` method that loads the training file values in memory.

Program Code for the getChart( ) Method

Listing 8-5 shows the program code for the `getChart()` method.

Listing 8-5. Code of the getChart Method

```
public XYChart getChart()
 {

  // Create the Chart

  Chart = new XYChartBuilder().width(900).height(500).title(getClass().
    getSimpleName()).xAxisTitle("day").yAxisTitle("y=f(day)").build();

  // Customize Chart
  Chart.getStyler().setPlotBackgroundColor(ChartColor.getAWTColor
  (ChartColor.GREY));
  Chart.getStyler().setPlotGridLinesColor(new Color(255, 255, 255));
  Chart.getStyler().setChartBackgroundColor(Color.WHITE);
  Chart.getStyler().setLegendBackgroundColor(Color.PINK);
  Chart.getStyler().setChartFontColor(Color.MAGENTA);
  Chart.getStyler().setChartTitleBoxBackgroundColor(new Color(0, 222, 0));
  Chart.getStyler().setChartTitleBoxVisible(true);
  Chart.getStyler().setChartTitleBoxBorderColor(Color.BLACK);
  Chart.getStyler().setPlotGridLinesVisible(true);
  Chart.getStyler().setAxisTickPadding(20);
  Chart.getStyler().setAxisTickMarkLength(15);
  Chart.getStyler().setPlotMargin(20);
  Chart.getStyler().setChartTitleVisible(false);
  Chart.getStyler().setChartTitleFont(new Font(Font.MONOSPACED,
  Font.BOLD, 24));
  Chart.getStyler().setLegendFont(new Font(Font.SERIF, Font.PLAIN, 18));
  Chart.getStyler().setLegendPosition(LegendPosition.OutsideE);
  Chart.getStyler().setLegendSeriesLineLength(12);
  Chart.getStyler().setAxisTitleFont(new Font(Font.SANS_SERIF,
  Font.ITALIC, 18));
  Chart.getStyler().setAxisTickLabelsFont(new Font(Font.SERIF,
  Font.PLAIN, 11));
```

```
//Chart.getStyler().setDayPattern("yyyy-MM");
Chart.getStyler().setDecimalPattern("#0.00");

// Config data

// For training
//intWorkingMode = 0;

// For testing
intWorkingMode = 1;
// common config data

intNumberOfTrainBatchesToProcess = 1000;
intNumberOfTestBatchesToProcess = 1000;
intNumberOfRecordsInTestFile = 999;
intNumberOfRowsInBatches = 1;
intInputNeuronNumber = 1;
intOutputNeuronNumber = 1;
strTrainFileNameBase = "C:/Book_Examples/Sample5_Train_Norm_Batch_";
strTestFileNameBase = "C:/Book_Examples/Sample5_Test_Norm_Batch_";
strSaveTrainNetworkFileBase = "C:/Book_Examples/Sample5_Save_Network_
Batch_";
strTrainChartFileName = "C:/Book_Examples/Sample5_Chart_Train_File_
Microbatch.jpg";
strTestChartFileName = "C:/Book_Examples/Sample5_Chart_Test_File_
Microbatch.jpg";
strFunctValueTrainFile = "C:/Book_Examples/Sample5_Train_Real.csv";
strFunctValueTestFile = "C:/Book_Examples/Sample5_Test_Real.csv";

// Generate training micro-batch file names and the corresponding Save
   Network file names

intDayNumber = -1;  // Day number for the chart

for (int i = 0; i < intNumberOfTrainBatchesToProcess; i++)
 {
   intDayNumber++;

   iString = Integer.toString(intDayNumber);
```

```
  if (intDayNumber >= 10 & intDayNumber < 100  )
   {
     strOutputFileName = strTrainFileNameBase + "0" + iString + ".csv";
     strSaveNetworkFileName = strSaveTrainNetworkFileBase + "0" +
     iString + ".csv";
   }
  else
   {
     if (intDayNumber < 10)
      {
        strOutputFileName = strTrainFileNameBase + "00" + iString +
        ".csv";
        strSaveNetworkFileName = strSaveTrainNetworkFileBase + "00" +
        iString + ".csv";
      }
     else
      {
        strOutputFileName = strTrainFileNameBase + iString + ".csv";

        strSaveNetworkFileName = strSaveTrainNetworkFileBase +
        iString + ".csv";
      }
   }

     strTrainingFileNames[intDayNumber] = strOutputFileName;
     strSaveTrainNetworkFileNames[intDayNumber] = strSaveNetwork
     FileName;

}  // End the FOR loop

 // Build the array linkToSaveNetworkFunctValueDiffKeys

 String tempLine;
 double tempNormFunctValueDiff = 0.00;
 double tempNormFunctValueDiffPerc = 0.00;
 double tempNormTargetFunctValueDiffPerc = 0.00;

 String[] tempWorkFields;
```

```
try
 {
    intDayNumber = -1;  // Day number for the chart

    for (int m = 0; m < intNumberOfTrainBatchesToProcess; m++)
      {
         intDayNumber++;

         BufferedReader br3 = new BufferedReader(new
         FileReader(strTrainingFileNames[intDayNumber]));
         tempLine = br3.readLine();

         // Skip the label record and zero batch record
         tempLine = br3.readLine();

         // Break the line using comma as separator
         tempWorkFields = tempLine.split(cvsSplitBy);

         tempNormFunctValueDiffPerc = Double.parseDouble(tempWork
         Fields[0]);
         tempNormTargetFunctValueDiffPerc = Double.parseDouble
         (tempWorkFields[1]);

         linkToSaveNetworkDayKeys[intDayNumber] = tempNormFunctValue
         DiffPerc;
         linkToSaveNetworkTargetFunctValueKeys[intDayNumber] =
         tempNormTargetFunctValueDiffPerc;

    }  // End the FOR loop

   // Generate testing micro-batch file names

   if(intWorkingMode == 1)
    {
      intDayNumber = -1;

      for (int i = 0; i < intNumberOfTestBatchesToProcess; i++)
       {
         intDayNumber++;
         iString = Integer.toString(intDayNumber);
```

```
          // Construct the testing batch names
          if (intDayNumber >= 10 & intDayNumber < 100)
           {
             strOutputFileName = strTestFileNameBase + "0" + iString +
              ".csv";
           }
          else
           {
             if (intDayNumber < 10)
              {
                strOutputFileName = strTestFileNameBase + "00" +
                iString + ".csv";
              }
             else
              {
                strOutputFileName = strTrainFileNameBase + iString +
                ".csv";
              }
           }

      strTestingFileNames[intDayNumber] = strOutputFileName;

    }  // End the FOR loop

   }   // End of IF

 }     // End for try
catch (IOException io1)
 {
   io1.printStackTrace();
   System.exit(1);
 }

loadTrainFunctValueFileInMemory();
```

When that part is done, the logic checks whether to run the training or testing method. When the workingMode field is equal to 1, it calls the training method in a loop (the way you did it previously). However, because you now have many micro-batch training files (instead of the single data set), you need to expand the errorCode array to hold one more value: the micro-batch number.

Code Fragment 1 of the Training Method

Inside the training file, if after many iterations the network error is unable to clear the error limit, you exit the training method with a returnCode value of 1. The control is returned to the logic inside the getChart() method that calls the training method in a loop. At that point, you need to return the parameters that the micro-batch method is being called with. Listing 8-6 shows code fragment 1 of the training method.

Listing 8-6. Code Fragment 1 of the Training Method

```
if(intWorkingMode == 0)
   {

      // Train batches and save the trained networks

      int paramErrorCode;
      int paramBatchNumber;
      int paramR;
      int paramDayNumber;
      int paramS;

      File file1 = new File(strTrainChartFileName);

      if(file1.exists())
       file1.delete();

     returnCodes[0] = 0;    // Clear the error Code
     returnCodes[1] = 0;    // Set the initial batch Number to 0;
     returnCodes[2] = 0;    // Set the initial day number to 0;
```

```
    do
      {
          paramErrorCode = returnCodes[0];
          paramBatchNumber = returnCodes[1];
          paramDayNumber = returnCodes[2];

          returnCodes =
          trainBatches(paramErrorCode,paramBatchNumber,paramDayNumber);
       } while (returnCodes[0] > 0);

  }   // End of the train logic
else
  {
     // Load and test the network logic

     File file2 = new File(strTestChartFileName);

     if(file2.exists())
      file2.delete();

     loadAndTestNetwork();

     // End of the test logic
  }

  Encog.getInstance().shutdown();
  return Chart;

}  // End of method
```

Code Fragment 2 of the Training Method

Here, most of the code should be familiar to you, except the logic involved in processing the micro-batches. You build the network. Next, you loop over the micro-batches (remember, there are many training micro-batch files instead of a single training data set you processed before). Inside the loop, you load the training micro-batch file in memory and then train the network using the current micro-batch file.

When the network is trained, you save it on disk, using the name from the `linkToSaveNetworkDayKeys` array that corresponds to the currently processed micro-batch file. Looping over the pair data set, you retrieve the input, actual, and predicted values for each micro-batch, denormalize them, and print the results as the training log.

Within the network train loop, when after many iterations the network error is unable to clear the error limit, you set the `returnCode` value to 1 and exit the training method. The control is returned to the logic that calls the training method in a loop. When you exit the training method, you now set three `returnCode` values: the `returnCode` value, the micro-batch number, and the day number. That helps the logic that calls the training method in a loop to stay within the same micro-batch and day of processing. You also populate the results for the chart elements. Finally, you add the chart series data, calculate the average and maximum errors for all micro-batches, print the results as the log file, and save the chart file. Listing 8-7 shows code fragment 2 of the training method.

Listing 8-7. Code Fragment 2 of the Training Method

```
// Build the network
BasicNetwork network = new BasicNetwork();

// Input layer
network.addLayer(new BasicLayer(null,true,intInputNeuronNumber));

// Hidden layer.
network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
network.addLayer(new BasicLayer(new ActivationTANH(),true,5));
network.addLayer(new BasicLayer(new ActivationTANH(),true,5));

// Output layer
network.addLayer(new BasicLayer(new ActivationTANH(),false,
intOutputNeuronNumber));
```

```
network.getStructure().finalizeStructure();
network.reset();

maxGlobalResultDiff = 0.00;
averGlobalResultDiff = 0.00;
sumGlobalResultDiff = 0.00;

// Loop over micro-batches

intDayNumber = paramDayNumber;  // Day number for the chart

for (rBatchNumber = paramBatchNumber; rBatchNumber < intNumberOfTrain
BatchesToProcess; rBatchNumber++)
  {
      intDayNumber++;  // Day number for the chart

      // Load the training CVS file for the current batch in memory
      MLDataSet trainingSet =
         loadCSV2Memory(strTrainingFileNames[rBatchNumber],intInput
         NeuronNumber,intOutputNeuronNumber,true,CSVFormat.ENGLISH,false);

    // train the neural network
    ResilientPropagation train = new ResilientPropagation(network,
    trainingSet);

   int epoch = 1;
  double tempLastErrorPerc = 0.00;

  do
    {
        train.iteration();

       epoch++;

      for (MLDataPair pair1:  trainingSet)
        {
            MLData inputData = pair1.getInput();
            MLData actualData = pair1.getIdeal();
            MLData predictData = network.compute(inputData);
```

```
        // These values are Normalized as the whole input is
        normInputFunctValueDiffPercFromRecord = inputData.getData(0);

      normTargetFunctValue = actualData.getData(0);
      normPredictFunctValue = predictData.getData(0);

      denormInputFunctValueDiffPercFromRecord =((inputDayDl -
      inputDayDh)*normInputFunctValueDiffPercFromRecord - Nh*inputDayDl +
      inputDayDh*Nl)/(Nl - Nh);

      denormTargetFunctValue = ((targetFunctValueDiffPercDl -
      targetFunctValueDiffPercDh)*normTargetFunctValue -
      Nh*targetFunctValueDiffPercDl + targetFunctValueDiffPercDh*Nl)/
      (Nl - Nh);
      denormPredictFunctValue = ((targetFunctValueDiffPercDl -
      targetFunctValueDiffPercDh)*normPredictFunctValue - Nh*target
      FunctValueDiffPercDl + targetFunctValueDiffPercDh*Nl)/(Nl - Nh);

      inputFunctValueFromFile = arrTrainFunctValues[rBatchNumber];

      targetToPredictFunctValueDiff = (Math.abs(denormTargetFunctValue -
      denormPredictFunctValue)/denormTargetFunctValue)*100;

   }

   if (epoch >= 500 &&targetToPredictFunctValueDiff > 0.0002)
      {
          returnCodes[0] = 1;
          returnCodes[1] = rBatchNumber;
          returnCodes[2] = intDayNumber-1;
          return returnCodes;
       }

   } while(targetToPredictFunctValueDiff >  0.0002);  // 0.00002

  // Save the network for the current batch
  EncogDirectoryPersistence.saveObject(newFile(strSaveTrainNetwork
  FileNames[rBatchNumber]),network);
```

```
 // Get the results after the network optimization
 int i = - 1;

for (MLDataPair pair:  trainingSet)
 {
   i++;

   MLData inputData = pair.getInput();
   MLData actualData = pair.getIdeal();
   MLData predictData = network.compute(inputData);

   // These values are Normalized as the whole input is
   normInputFunctValueDiffPercFromRecord = inputData.getData(0);

   normTargetFunctValue = actualData.getData(0);
   normPredictFunctValue = predictData.getData(0);

   denormInputFunctValueDiffPercFromRecord = ((inputDayDl - inputDayDh)*
   normInputFunctValueDiffPercFromRecord - Nh*inputDayDl +
   inputDayDh*Nl)/(Nl - Nh);

   denormTargetFunctValue = ((targetFunctValueDiffPercDl - targetFunct
   ValueDiffPercDh)*normTargetFunctValue – Nh*targetFunctValueDiffPercDl +
   targetFunctValueDiffPercDh*Nl)/(Nl - Nh);

   denormPredictFunctValue = ((targetFunctValueDiffPercDl - target
   FunctValueDiffPercDh)*normPredictFunctValue – Nh*targetFunctValue
   DiffPercDl + targetFunctValueDiffPercDh*Nl)/(Nl - Nh);

   inputFunctValueFromFile = arrTrainFunctValues[rBatchNumber];

  targetToPredictFunctValueDiff = (Math.abs(denormTargetFunctValue -
  denormPredictFunctValue)/denormTargetFunctValue)*100;

   System.out.println("intDayNumber = " + intDayNumber +  "  target
   FunctionValue = " + denormTargetFunctValue + "  predictFunction
   Value = " + denormPredictFunctValue + "  valurDiff = " + targetTo
   PredictFunctValueDiff);
```

```
        if (targetToPredictFunctValueDiff > maxGlobalResultDiff)
        maxGlobalResultDiff =targetToPredictFunctValueDiff;

        sumGlobalResultDiff = sumGlobalResultDiff +targetToPredictFunct
        ValueDiff;

        // Populate chart elements
        doubleDayNumber = (double) rBatchNumber+1;
        xData.add(doubleDayNumber);
        yData1.add(denormTargetFunctValue);
        yData2.add(denormPredictFunctValue);

       }  // End for the pair loop

    }  // End of the loop over batches

    sumGlobalResultDiff = sumGlobalResultDiff +targetToPredictFunctValueDiff;
    averGlobalResultDiff = sumGlobalResultDiff/intNumberOfTrainBatches
    ToProcess;

    // Print the max and average results

    System.out.println(" ");
    System.out.println(" ");
    System.out.println("maxGlobalResultDiff = " + maxGlobalResultDiff);
    System.out.println("averGlobalResultDiff = " + averGlobalResultDiff);

    XYSeries series1 = Chart.addSeries("Actual", xData, yData1);
    XYSeries series2 = Chart.addSeries("Predicted", xData, yData2);

    series1.setLineColor(XChartSeriesColors.BLUE);
    series2.setMarkerColor(Color.ORANGE);
    series1.setLineStyle(SeriesLines.SOLID);
    series2.setLineStyle(SeriesLines.SOLID);

    // Save the chart image
    try
      {
        BitmapEncoder.saveBitmapWithDPI(Chart, strTrainChartFileName,
        BitmapFormat.JPG, 100);
      }
```

```
     catch (Exception bt)
      {
         bt.printStackTrace();
      }

      System.out.println ("The Chart has been saved");

    returnCodes[0] = 0;
    returnCodes[1] = 0;
    returnCodes[2] = 0;

    return returnCodes;

   }  // End of method
```

Training Results for the Micro-Batch Method

Listing 8-8 shows the ending fragment of the training results.

Listing 8-8. Training Results

```
DayNumber =  989  TargeValue = 735.09  PredictedValue = 735.09005
DiffPercf = 6.99834E-6
DayNumber =  990  TargeValue = 797.87  PredictedValue = 797.86995
DiffPercf = 6.13569E-6
DayNumber =  991  TargeValue = 672.81  PredictedValue = 672.80996
DiffPercf = 5.94874E-6
DayNumber =  992  TargeValue = 619.14  PredictedValue = 619.14003
DiffPercf = 5.53621E-6
DayNumber =  993  TargeValue = 619.32  PredictedValue = 619.32004
DiffPercf = 5.65663E-6
DayNumber =  994  TargeValue = 590.47  PredictedValue = 590.47004
DiffPercf = 6.40373E-6
DayNumber =  995  TargeValue = 547.28  PredictedValue = 547.27996
DiffPercf = 6.49734E-6
DayNumber =  996  TargeValue = 514.62  PredictedValue = 514.62002
DiffPercf = 3.39624E-6
```

```
DayNumber =  997  TargeValue = 455.4   PredictedValue = 455.40000
DiffPercf = 2.73780E-7
DayNumber =  998  TargeValue = 470.43  PredictedValue = 470.42999
DiffPercf = 4.35234E-7
DayNumber =  999  TargeValue = 480.28  PredictedValue = 480.28002
DiffPercf = 3.52857E-6
DayNumber = 1000 TargeValue = 496.77  PredictedValue = 496.76999
DiffPercf = 9.81900E-7

maxGlobalResultDiff = 9.819000149262707E-7
averGlobalResultDiff = 1.9638000298525415E-9
```

Now, the training processing results are quite good, especially for approximating the noncontinuous function. The average error is 0.0000000019638000298525415, the maximum error (the worst optimized record) is 0.000000981900014926270 7, and the chart looks great. Figure 8-6 shows the chart of the training processing results using micro-batches.

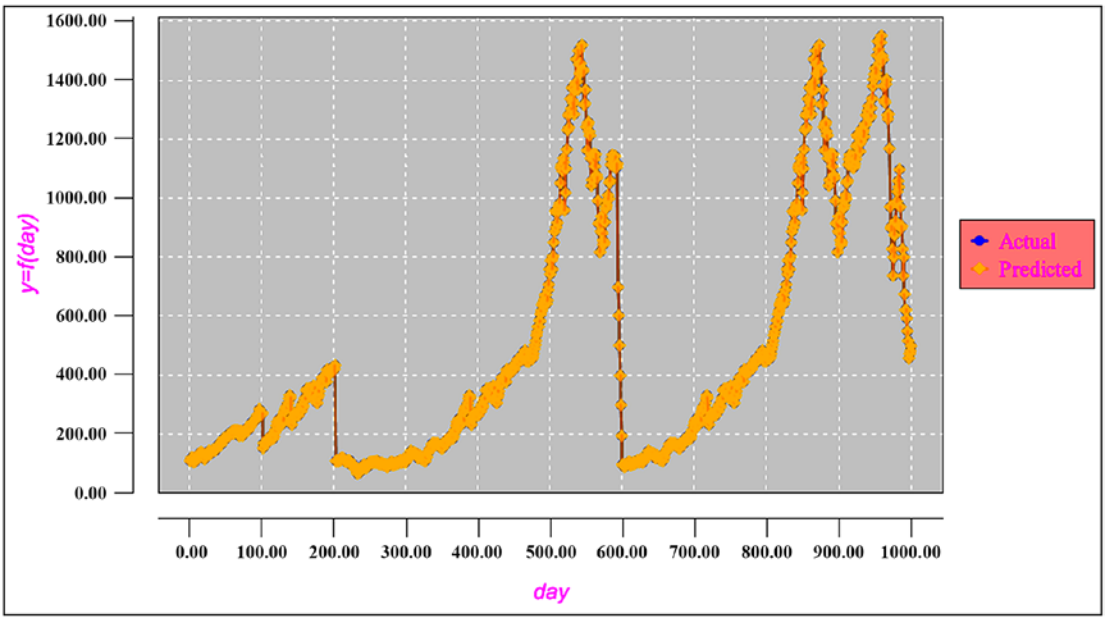


Figure 8-6. *Chart of training results using micro-batches*

For testing, you will build a file with values between the training points. For example, for the two training records 1 and 2, you will calculate a new day as the average of the two training days. For the record's function value, you will calculate the average of the two training function values. This way, two consecutive training records will create a single test record with values averaging the training records. Table 8-4 shows how the test record looks.

Table 8-4. *Test Record*

1.5	108.918

It averages two training records, as shown in Table 8-5.

Table 8-5. *Two Training Records*

1	107.387
2	110.449

The test data set has 998 records. Table 8-6 shows a fragment of the testing data set.

Table 8-6. *Fragment of the Testing Data Set*

xPoint	yValue	xPoint	yValue	xPoint	yValue
1.5	108.918	31.5	139.295	61.5	204.9745
2.5	113.696	32.5	142.3625	62.5	208.6195
3.5	117.806	33.5	142.6415	63.5	207.67
4.5	113.805	34.5	141.417	64.5	209.645
5.5	106.006	35.5	142.1185	65.5	208.525
6.5	106.6155	36.5	146.215	66.5	208.3475
7.5	107.5465	37.5	150.6395	67.5	203.801
8.5	109.5265	38.5	154.1935	68.5	194.6105
9.5	116.223	39.5	158.338	69.5	199.9695
10.5	118.1905	40.5	161.4155	70.5	207.9885

(*continued*)

Table 8-6. (*continued*)

xPoint	yValue	xPoint	yValue	xPoint	yValue
11.5	121.9095	41.5	161.851	71.5	206.2175
12.5	127.188	42.5	164.6005	72.5	199.209
13.5	130.667	43.5	165.935	73.5	193.6235
14.5	132.6525	44.5	165.726	74.5	199.5985
15.5	134.472	45.5	171.9045	75.5	206.252
16.5	135.4405	46.5	178.1175	76.5	208.113
17.5	133.292	47.5	182.7085	77.5	209.791
18.5	130.646	48.5	181.5475	78.5	213.623
19.5	125.5585	49.5	182.102	79.5	217.2275
20.5	117.5155	50.5	186.5895	80.5	216.961
21.5	119.236	51.5	187.8145	81.5	214.721
22.5	125.013	52.5	190.376	82.5	216.248
23.5	125.228	53.5	194.19	83.5	221.882
24.5	128.5005	54.5	194.545	84.5	225.885
25.5	133.9045	55.5	196.702	85.5	232.1255
26.5	138.7075	56.5	198.783	86.5	236.318
27.5	140.319	57.5	199.517	87.5	237.346
28.5	135.412	58.5	204.2805	88.5	239.8
29.5	133.6245	59.5	206.323	89.5	241.7605
30.5	137.074	60.5	202.6945	90.5	244.6855

Table 8-7 shows a fragment of the normalized testing data set.

Table 8-7. *Fragment of the Normalized Testing Data Set*

xPoint	y	xPoint	y	xPoint	y
-0.9990	-0.9365	-0.9389	-0.8970	-0.8789	-0.8117
-0.9970	-0.9303	-0.9369	-0.8930	-0.8769	-0.8070
-0.9950	-0.9249	-0.9349	-0.8927	-0.8749	-0.8082
-0.9930	-0.9301	-0.9329	-0.8943	-0.8729	-0.8057
-0.9910	-0.9403	-0.9309	-0.8934	-0.8709	-0.8071
-0.9890	-0.9395	-0.9289	-0.8880	-0.8689	-0.8073
-0.9870	-0.9383	-0.9269	-0.8823	-0.8669	-0.8132
-0.9850	-0.9357	-0.9249	-0.8777	-0.8649	-0.8252
-0.9830	-0.9270	-0.9229	-0.8723	-0.8629	-0.8182
-0.9810	-0.9244	-0.9209	-0.8683	-0.8609	-0.8078
-0.9790	-0.9196	-0.9189	-0.8677	-0.8589	-0.8101
-0.9770	-0.9127	-0.9169	-0.8642	-0.8569	-0.8192
-0.9750	-0.9082	-0.9149	-0.8624	-0.8549	-0.8265
-0.9730	-0.9056	-0.9129	-0.8627	-0.8529	-0.8187
-0.9710	-0.9033	-0.9109	-0.8547	-0.8509	-0.8101
-0.9690	-0.9020	-0.9089	-0.8466	-0.8488	-0.8076
-0.9670	-0.9048	-0.9069	-0.8406	-0.8468	-0.8055
-0.9650	-0.9083	-0.9049	-0.8421	-0.8448	-0.8005
-0.9630	-0.9149	-0.9029	-0.8414	-0.8428	-0.7958
-0.9610	-0.9253	-0.9009	-0.8356	-0.8408	-0.7962
-0.9590	-0.9231	-0.8989	-0.8340	-0.8388	-0.7991
-0.9570	-0.9156	-0.8969	-0.8307	-0.8368	-0.7971
-0.9550	-0.9153	-0.8949	-0.8257	-0.8348	-0.7898

(*continued*)

Table 8-7. (*continued*)

xPoint	y	xPoint	y	xPoint	y
-0.9530	-0.9110	-0.8929	-0.8253	-0.8328	-0.7846
-0.9510	-0.9040	-0.8909	-0.8225	-0.8308	-0.7765
-0.9489	-0.8978	-0.8889	-0.8198	-0.8288	-0.7710
-0.9469	-0.8957	-0.8869	-0.8188	-0.8268	-0.7697
-0.9449	-0.9021	-0.8849	-0.8126	-0.8248	-0.7665
-0.9429	-0.9044	-0.8829	-0.8100	-0.8228	-0.7639
-0.9409	-0.8999	-0.8809	-0.8147	-0.8208	-0.7601

Like with the normalized training data set, you break the normalized testing data set into micro-batches. Each micro-batch data set should contain the label record and the record from the original file to be processed. As a result, you will get 998 micro-batch data sets (numbered from 0 to 997). Figure 8-7 shows a fragment of the list of normalized testing micro-batch files.

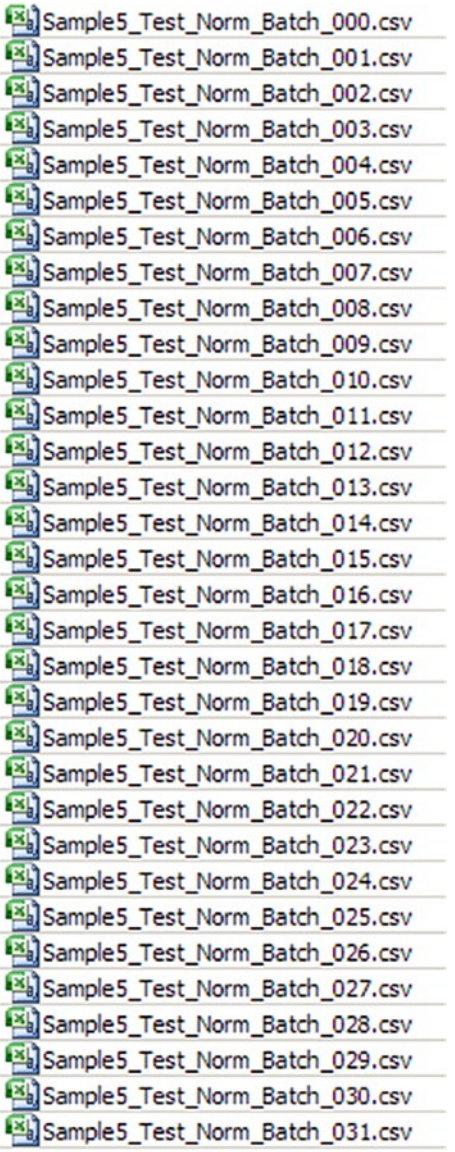


Figure 8-7. *Fragment of the normalized micro-batch test data set*

This set of files is now the input to the neural network testing process.

Test Processing Logic

For the test processing logic, you loop over micro-batches. For each test micro-batch, you read its record, retrieve the record values, and denormalize them. Next, you load the micro-batch data set for point 1 (which is the closest point to the testing record but

less than it) in memory. You also load the corresponding save-network file in memory. Looping over the pairs data set, you retrieve the input, active, and predicted values for the micro-batch and denormalize them.

You also load the micro-batch data set for point 2 (which is the closest point to the testing record but greater than it) in memory, and you load the corresponding save-network file in memory. Looping over the pairs data set, you retrieve the input, active, and predicted values for the micro-batch and denormalize them.

Next, you calculate the average predicted function values for point 1 and point 2. Finally, you calculate the error percent and print the results as the processing log. The rest is just the miscellaneous staff. Listing 8-9 shows the program code for the testing method.

Listing 8-9. Code of the Testing Method

```
for (k1 = 0; k1 < intNumberOfRecordsInTestFile; k1++)
    {
            // Read the corresponding test micro-batch file.
            br4 = new BufferedReader(new FileReader(strTestingFile
            Names[k1]));
            tempLine = br4.readLine();

            // Skip the label record
            tempLine = br4.readLine();

            // Break the line using comma as separator
            tempWorkFields = tempLine.split(cvsSplitBy);

            dayKeyFromRecord = Double.parseDouble(tempWorkFields[0]);
            targetFunctValueFromRecord = Double.parseDouble(tempWork
            Fields[1]);

            // Load the corresponding test micro-batch dataset in memory
            MLDataSet testingSet =
               loadCSV2Memory(strTestingFileNames[k1],intInputNeuronNumber,
               intOutputNeuronNumber,true,CSVFormat.ENGLISH,false);

            // Load the corresponding save network for the currently
               processed micro-batch
            r1 = linkToSaveNetworkDayKeys[k1];
```

```
network =
  (BasicNetwork)EncogDirectoryPersistence.loadObject(new
  File(strSaveTrainNetworkFileNames[k1]));

 // Get the results after the network optimization
 int iMax = 0;
 int i = - 1; // Index of the array to get results

 for (MLDataPair pair:  testingSet)
  {
     i++;
     iMax = i+1;

     MLData inputData = pair.getInput();
     MLData actualData = pair.getIdeal();
     MLData predictData = network.compute(inputData);

     // These values are Normalized as the whole input is
     normInputDayFromRecord = inputData.getData(0);
     normTargetFunctValue = actualData.getData(0);
     normPredictFunctValue = predictData.getData(0);

     denormInputDayFromRecord = ((inputDayDl - inputDayDh)*
     normInputDayFromRecord - Nh*inputDayDl + inputDayDh*Nl)/
     (Nl - Nh);

     denormTargetFunctValue = ((targetFunctValueDiffPercDl -
     targetFunctValueDiffPercDh)*normTargetFunctValue -
     Nh*targetFunctValueDiffPercDl + targetFunctValue
     DiffPercDh*Nl)/(Nl - Nh);

     denormPredictFunctValue =((targetFunctValueDiffPercDl -
     targetFunctValueDiffPercDh)*normPredictFunctValue - Nh*
     targetFunctValueDiffPercDl + targetFunctValueDiff
     PercDh*Nl)/(Nl - Nh);

     targetToPredictFunctValueDiff = (Math.abs(denormTarget
     FunctValue - denormPredictFunctValue)/denormTargetFunct
     Value)*100;
```

```
                System.out.println("DayNumber = " + denormInputDayFrom
                Record + "  targetFunctionValue = " + denormTarget
                FunctValue + "  predictFunctionValue = " + denormPredict
                FunctValue + "  valurDiff = " + targetToPredictFunct
                ValueDiff);

                if (targetToPredictFunctValueDiff > maxGlobalResultDiff)
                 {
                   maxGlobalIndex = iMax;
                   maxGlobalResultDiff =targetToPredictFunctValueDiff;
                 }

                sumGlobalResultDiff = sumGlobalResultDiff + targetToPredict
                FunctValueDiff;

                // Populate chart elements

                xData.add(denormInputDayFromRecord);
                yData1.add(denormTargetFunctValue);
                yData2.add(denormPredictFunctValue);

              }  // End for pair loop

         }   // End of loop using k1

        // Print the max and average results

        System.out.println(" ");

        averGlobalResultDiff = sumGlobalResultDiff/intNumberOfRecords
        InTestFile;

        System.out.println("maxErrorPerc = " + maxGlobalResultDiff);
        System.out.println("averErroPerc = " + averGlobalResultDiff);

      }
     catch (IOException e1)
      {
            e1.printStackTrace();
      }
```

```
    // All testing batch files have been processed
      XYSeries series1 = Chart.addSeries("Actual", xData, yData1);
      XYSeries series2 = Chart.addSeries("Forecasted", xData, yData2);

      series1.setLineColor(XChartSeriesColors.BLUE);
      series2.setMarkerColor(Color.ORANGE);
      series1.setLineStyle(SeriesLines.SOLID);
      series2.setLineStyle(SeriesLines.SOLID);

      // Save the chart image
      try
       {
         BitmapEncoder.saveBitmapWithDPI(Chart, strTrainChartFileName,
         BitmapFormat.JPG, 100);
       }
      catch (Exception bt)
       {
         bt.printStackTrace();
       }

      System.out.println ("The Chart has been saved");

      System.out.println("End of testing for mini-batches training");

   } // End of the method
```

Testing Results for the Micro-Batch Method

Listing 8-10 shows the end fragment of the testing results.

Listing 8-10. End Fragment of the Testing Results

```
DayNumber = 986.5  TargetValue = 899.745  AverPredictedValue = 899.74503
DiffPerc = 3.47964E-6
DayNumber = 987.5  TargetValue = 864.565  AverPredictedValue = 864.56503
DiffPerc = 3.58910E-6
DayNumber = 988.5  TargetValue = 780.485  AverPredictedValue = 780.48505
DiffPerc = 6.14256E-6
```

```
DayNumber = 989.5  TargetValue = 766.48    AverPredictedValue = 766.48000
DiffPerc = 1.62870E-7
DayNumber = 990.5  TargetValue = 735.34    AverPredictedValue = 735.33996
DiffPerc = 6.05935E-6
DayNumber = 991.5  TargetValue = 645.975  AverPredictedValue = 645.97500
DiffPerc = 4.53557E-7
DayNumber = 992.5  TargetValue = 619.23    AverPredictedValue = 619.23003
DiffPerc = 5.59670E-6
DayNumber = 993.5  TargetValue = 604.895  AverPredictedValue = 604.89504
DiffPerc = 6.02795E-6
DayNumber = 994.5  TargetValue = 568.875  AverPredictedValue = 568.87500
DiffPerc = 2.02687E-7
DayNumber = 995.5  TargetValue = 530.95    AverPredictedValue = 530.94999
DiffPerc = 1.71056E-6
DayNumber = 996.5  TargetValue = 485.01    AverPredictedValue = 485.01001
DiffPerc = 1.92301E-6
DayNumber = 997.5  TargetValue = 462.915  AverPredictedValue = 462.91499
DiffPerc = 7.96248E-8
DayNumber = 998.5  TargetValue = 475.355  AverPredictedValue = 475.35501
DiffPerc = 1.57186E-6
DayNumber = 999.5  TargetValue = 488.525  AverPredictedValue = 488.52501
DiffPerc = 1.23894E-6

maxErrorPerc = 6.840306081962611E-6
averErrorPerc  = 2.349685401959033E-6kim
```

Now, the testing results are also pretty good, considering that the function is noncontinuous. The `maxErrorPerc` field, which is the worst error among all records, is less than 0.0000068 percent, and the `averErrorPerc` field is less than 0.0000023 percent. If the day of the current micro-batch test file is not in the middle of two save-network keys, calculate the values proportionally or use interpolation for that purpose. Figure 8-8 shows the chart of the testing results. Both the actual and predicted charts overlap.

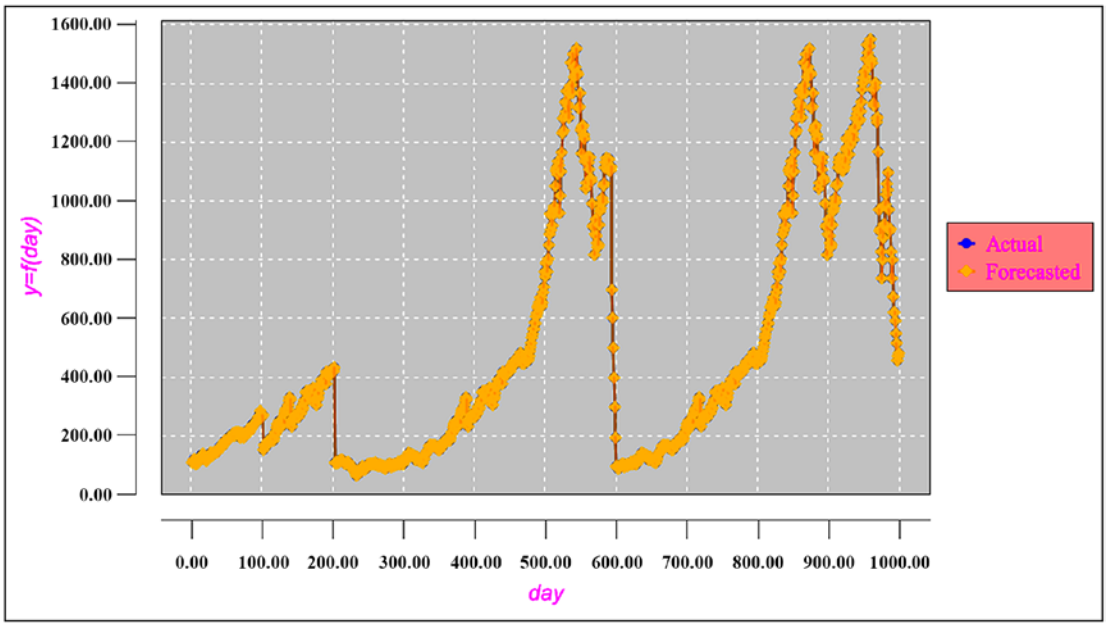


Figure 8-8. *Chart of the test results*

Both charts are practically identical and overlap each other.

Digging Deeper

Neural network backpropagation is considered a universal function approximation mechanism. However, there is a strict limitation for the type of functions that neural networks are able to approximate: the functions must be continuous (the universal approximation theorem).

Let's discuss what happens when the network attempts to approximate a noncontinuous function. To research this question, you use a small noncontinuous function that is given by its values at 20 points. These points surround the point of a rapidly changing function pattern. Figure 8-9 shows the chart.

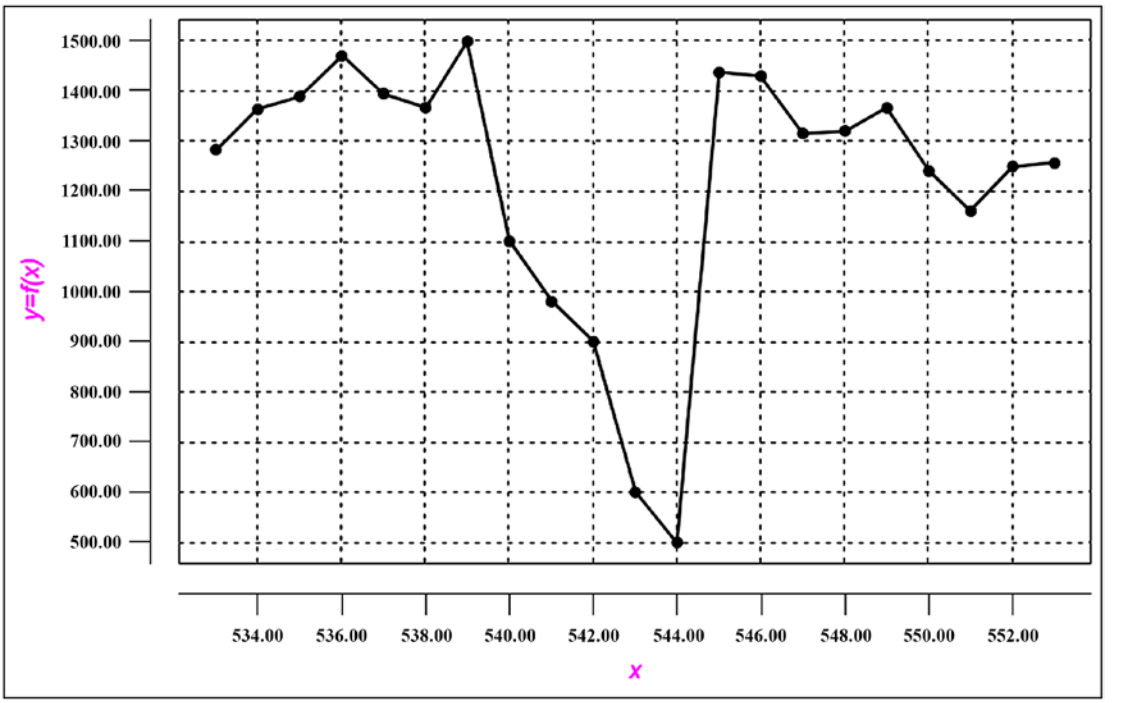


Figure 8-9. *Chart of the function with the rapidly changing pattern*

Table 8-8 shows the function values at 20 points.

Table 8-8. *Function Values*

X Point	Function Value
533	1282.71
534	1362.93
535	1388.91
536	1469.25
537	1394.46
538	1366.42
539	1498.58
540	1100

(*continued*)

Table 8-8. (*continued*)

X Point	Function Value
541	980
542	900
543	600
544	500
545	1436.51
546	1429.4
547	1314.95
548	1320.28
549	1366.01
550	1239.94
551	1160.33
552	1249.46
553	1255.82

This file is normalized before being processed. Figure 8-10 shows the network architecture used for approximating this function.

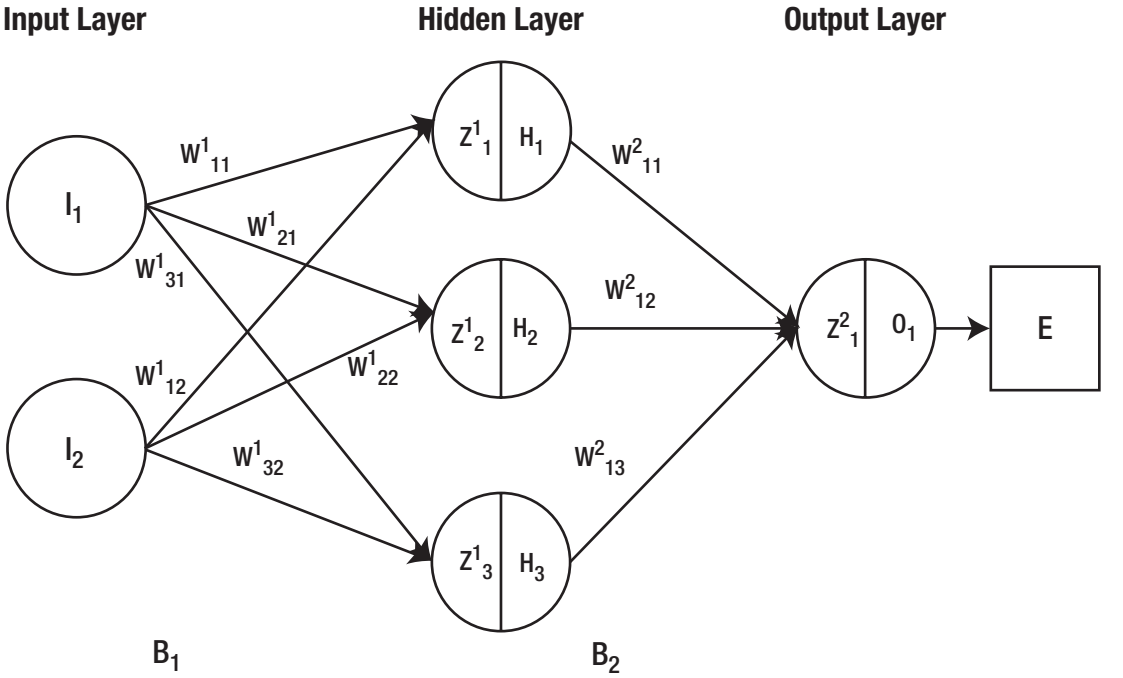


Figure 8-10. *Network architecture*

Executing the training process shows the following results:

– The maximum error percent (the maximum percent of the difference between the actual and predicted function values) is greater than 130.06 percent.

– The average error percent (the average percent of the difference between the actual and predicted function values) is greater than 16.25 percent.

Figure 8-11 shows the chart of the processing results.

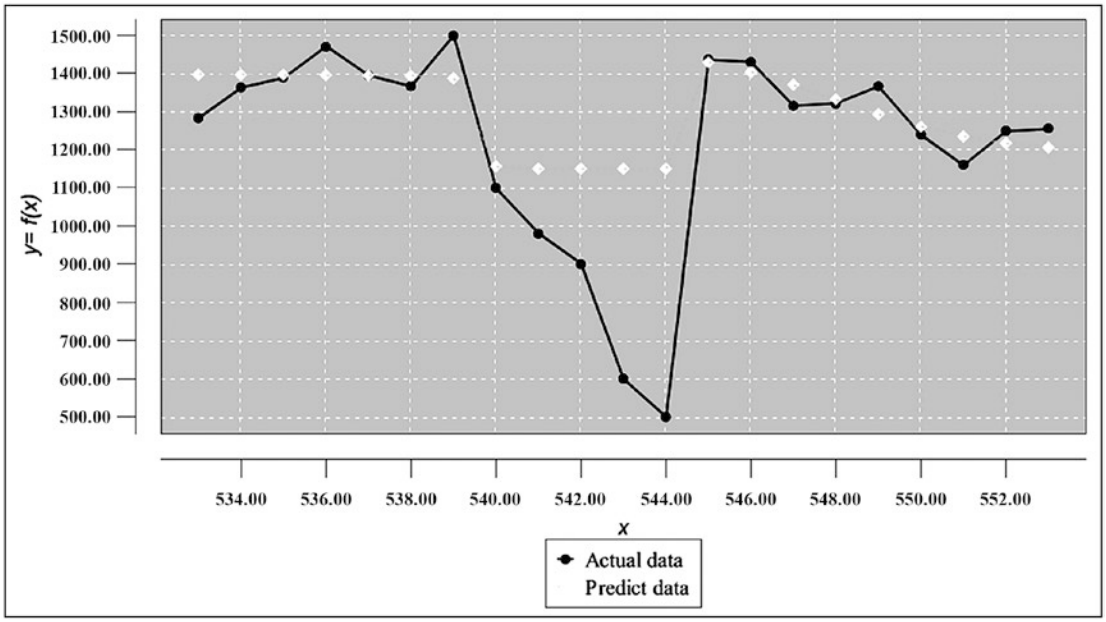


Figure 8-11. *Chart of the processing results*

The goal is to understand what happens during this noncontinuous function approximation process that leads to such poor results. To research this, you will calculate the forward pass result (the error) for each record. The calculation for the forward pass is done using Equations 8-1 through 8-5.

Neuron H_1

$$\begin{aligned} Z_1^1 &= W_{11}^1 * I_1 + B_1 * 1 \\ H_1 &= \sigma\left(Z_1^1\right) \end{aligned} \tag{8-1}$$

Neuron H_2

$$\begin{aligned} Z_2^1 &= W_{21}^1 * I_1 + B_1 * 1 \\ H_2 &= \sigma\left(Z_2^1\right) \end{aligned} \tag{8-2}$$

Neuron H_3

$$\begin{aligned} Z_3^1 &= W_{31}^1 * I_1 + B_1 * 1 \\ H_3 &= \sigma\left(Z_3^1\right) \end{aligned} \tag{8-3}$$

These calculations give you output from neurons H1, H2, and H3. Those values are used when processing neurons in the next layer (in this case, the output layer).

See Equation 8-4 for neuron $O_{1.}$

$$Z_1^2 = W_{11}^{2*} H_1 + W_{12}^{2*} H_2 + W_{13}^{2*} H_3 + B_2^* 1$$
$$O_1 = \sigma\left(Z_1^2\right) \tag{8-4}$$

Equation 8-5 shows the error function.

$$E = 0.5^* \left(\text{Actual Value for Record} \quad O_1\right)^2 \tag{8-5}$$

In Equations 8-1 through 8-3, σ is the activation function, W is the weight, and B is the bias.

Table 8-9 shows the calculated error for each record for the first forward pass.

Table 8-9. *Record Errors for the First Pass*

Day	Function Value		
-0.76	-0.410177778		
-0.68	-0.053644444		
-0.6	0.061822222		
-0.52	0.418888889		
-0.44	0.086488889	**Max**	**0.202629155**
-0.36	-0.038133333	**Min**	**0.156038965**
-0.28	0.549244444		
-0.2	-1.222222222	**Difference Percent**	**29.86**
-0.12	-1.755555556		
-0.04	-2.111111111		
0.04	-3.444444444		
0.12	-3.888888889		
0.2	0.273377778		

(*continued*)

Table 8-9. (*continued*)

Day	Function Value
0.28	0.241777778
0.36	-0.266888889
0.44	-0.2432
0.52	-0.039955556
0.6	-0.600266667
0.68	-0.954088889
0.76	-0.557955556
0.84	-0.529688889

The difference between the maximum and minimum error values for all records is very large and is about 30 percent. That's where the problem exists. When all records are processed, this point is the epoch. At that point, the network calculates the average error (for all processed errors in the epoch) and then processes the backpropagation step to redistribute the average error among all neurons in the output and hidden layers, adjusting their weights and bias values.

The calculated errors for all records depend on the initial (randomly assigned) weight/bias parameters set for this first pass. When the function contains continuous (monotone) function values that are gradually changed in an orderly way, the errors calculated for each record based on the initial weight/bias values are close enough, and the average error is close to the error calculated for each record. However, when the function is noncontinuous, its pattern rapidly changes at some points. That leads to the situation when the randomly selected initial weight/bias values are not good for all records, leading to wide differences between record errors.

Next, the backpropagation adjusts the initial weight/bias values of the neurons, but the problem continues to exist: those adjusted values are not good for all records that belong to a different function pattern (topology).

Tip The micro-batch method requires more calculations than the conventional way of network processing, so it should be used only when the conventional method is unable to deliver good approximation results.

Summary

Neural network approximation of noncontinuous functions is a difficult task for neural networks. It is practically impossible to obtain a good-quality approximation for such functions. This chapter introduced the micro-batch method, which is able to approximate any noncontinuous function with high-precision results. The next chapter shows how the micro-batch method substantially improves the approximation results for continuous functions with complex topologies.

CHAPTER 9

Approximating Continuous Functions with Complex Topology

This chapter shows that the micro-batch method substantially improves the approximation results of continuous functions with complex topologies.

Example 5a: Approximation of a Continuous Function with Complex Topology Using the Conventional Network Process

Figure 9-1 shows one such function. The function has the following formula: $y = \sqrt{e^{-(\sin(e^x))}}$. However, let's pretend that the function formula is unknown and that the function is given to you by its values at certain points.

I. Livshin, *Artificial Neural Networks with Java*, https://doi.org/10.1007/978-1-4842-4421-0_9

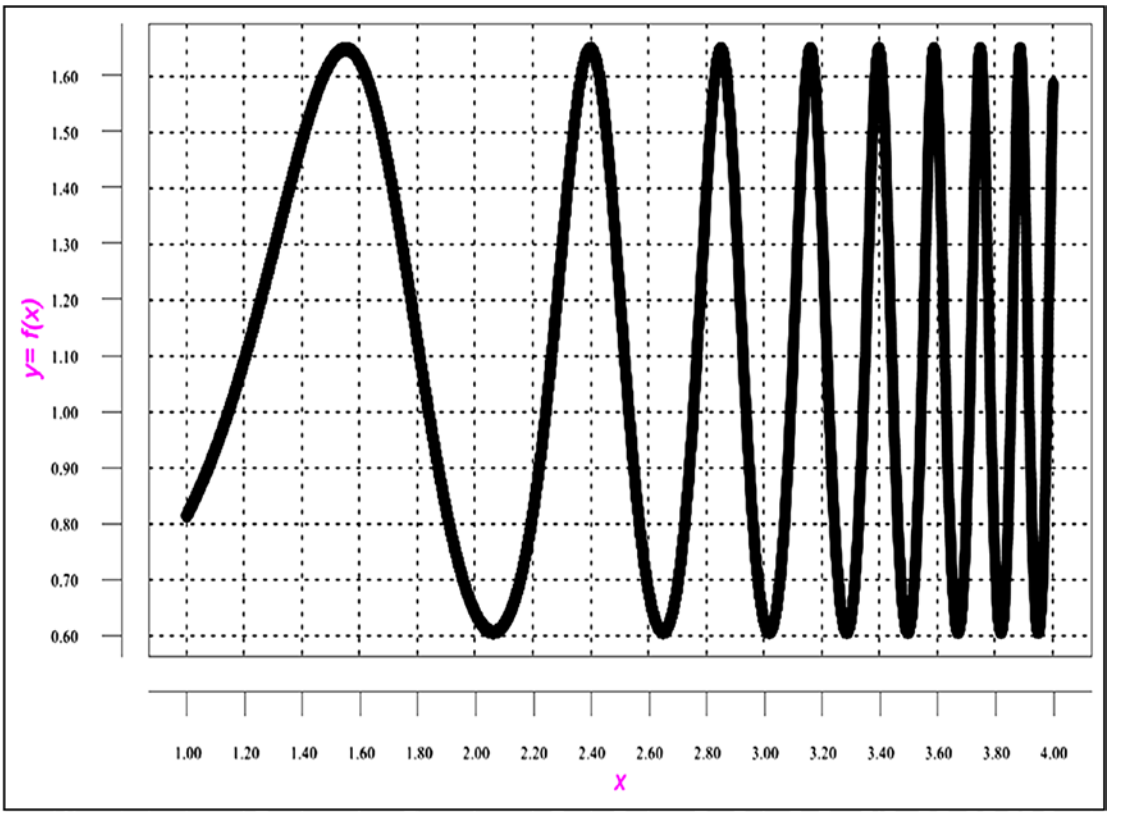


Figure 9-1. *Chart of the continuous function with a complicated topology*

Again, let's first attempt to approximate this function using the conventional neural network process. Table 9-1 shows a fragment of the training data set.

Table 9-1. *Fragment of the Training Data Set*

Point x	Function Value
1	0.81432914
1.0003	0.814632027
1.0006	0.814935228
1.0009	0.815238744
1.0012	0.815542575

(*continued*)

Table 9-1. (*continued*)

Point x	Function Value
1.0015	0.815846721
1.0018	0.816151183
1.0021	0.816455961
1.0024	0.816761055
1.0027	0.817066464
1.003	0.817372191
1.0033	0.817678233
1.0036	0.817984593
1.0039	0.818291269
1.0042	0.818598262

Table 9-2 shows a fragment of the testing data set.

Table 9-2. *Fragment of the Testing Data Set*

Point x	Point y
1.000015	0.814344277
1.000315	0.814647179
1.000615	0.814950396
1.000915	0.815253928
1.001215	0.815557774
1.001515	0.815861937
1.001815	0.816166415
1.002115	0.816471208
1.002415	0.816776318
1.002715	0.817081743

(*continued*)

Table 9-2. (*continued*)

Point x	Point y
1.003015	0.817387485
1.003315	0.817693544
1.003615	0.817999919
1.003915	0.818306611
1.004215	0.81861362

Both the training and testing data sets have been normalized before processing.

Network Architecture for Example 5a

Figure 9-2 shows the network architecture.

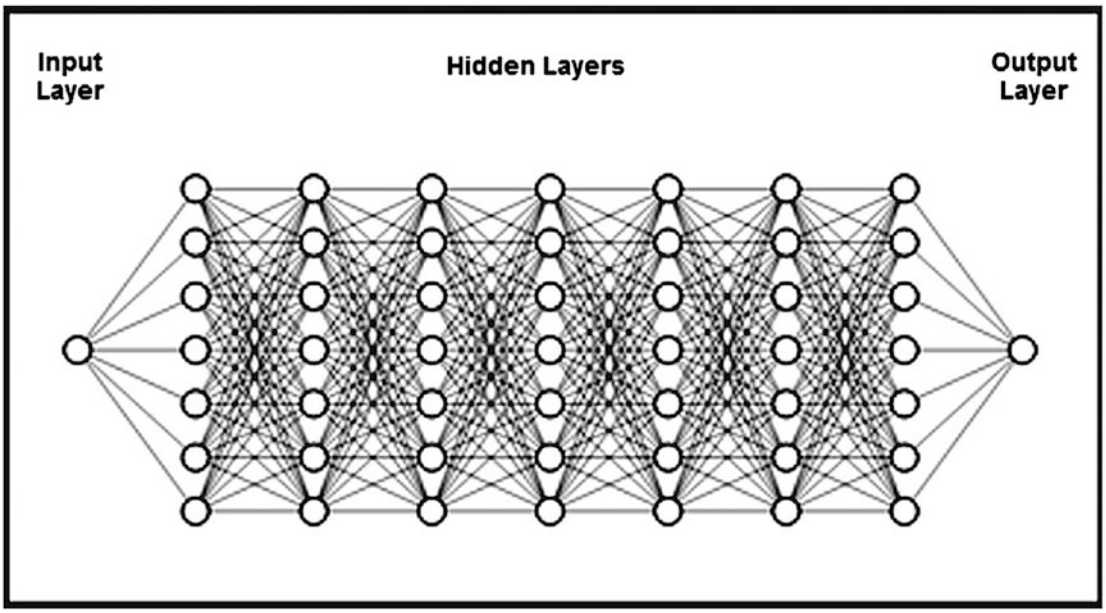


Figure 9-2. *Network architecture*

Both the training and testing data sets have been normalized.

Program Code for Example 5a

Listing 9-1 shows the program code. The code shows the conventional method for neural network processing, which approximates the function with complex topology (shown in Figure 9-1). Like with the conventional neural network processing shown in previous chapters, you first train the network.

That includes normalizing the input data on the interval [-1, 1] and then approximating the function at the training points. Next, in the testing mode, you calculate (predict) the function values at the points not used during the network training. Finally, you calculate the difference between the actual function values (known to you) and the predicted values. I will show the difference between the charts of the actual values and the predicted values.

Listing 9-1. Program Code

```
// ============================================================
// Approximation of the complex function using the conventional approach.
// The complex function values are given at 1000 points.
//
// The input file consists of records with two fields:
// Field1 - xPoint value
// Field2 - Function value at the xPoint
//
// The input file is normalized.
// ============================================================

package articleidi_complexformula_traditional;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.*;
```

```
import java.util.Properties;
import java.time.YearMonth;
import java.awt.Color;
import java.awt.Font;
import java.io.BufferedReader;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.LocalDate;
import java.time.Month;
import java.time.ZoneId;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.List;
import java.util.Locale;
import java.util.Properties;

import org.encog.Encog;
import org.encog.engine.network.activation.ActivationTANH;
import org.encog.engine.network.activation.ActivationReLU;
import org.encog.ml.data.MLData;
import org.encog.ml.data.MLDataPair;
import org.encog.ml.data.MLDataSet;
import org.encog.ml.data.buffer.MemoryDataLoader;
import org.encog.ml.data.buffer.codec.CSVDataCODEC;
import org.encog.ml.data.buffer.codec.DataSetCODEC;
import org.encog.neural.networks.BasicNetwork;
import org.encog.neural.networks.layers.BasicLayer;
import org.encog.neural.networks.training.propagation.resilient.
ResilientPropagation;
import org.encog.persist.EncogDirectoryPersistence;
import org.encog.util.csv.CSVFormat;

import org.knowm.xchart.SwingWrapper;
import org.knowm.xchart.XYChart;
import org.knowm.xchart.XYChartBuilder;
```

```
import org.knowm.xchart.XYSeries;
import org.knowm.xchart.demo.charts.ExampleChart;
import org.knowm.xchart.style.Styler.LegendPosition;
import org.knowm.xchart.style.colors.ChartColor;
import org.knowm.xchart.style.colors.XChartSeriesColors;
import org.knowm.xchart.style.lines.SeriesLines;
import org.knowm.xchart.style.markers.SeriesMarkers;
import org.knowm.xchart.BitmapEncoder;
import org.knowm.xchart.BitmapEncoder.BitmapFormat;
import org.knowm.xchart.QuickChart;
import org.knowm.xchart.SwingWrapper;

public class ArticleIDI_ComplexFormula_Traditional implements ExampleChart
<XYChart>
{
    // Interval to normalize
   static double Nh =  1;
   static double Nl = -1;

   // First column
   static double minXPointDl = 0.95;
   static double maxXPointDh = 4.05;

   // Second column - target data
   static double minTargetValueDl = 0.60;
   static double maxTargetValueDh = 1.65;

   static double doublePointNumber = 0.00;
   static int intPointNumber = 0;
   static InputStream input = null;
   static double[] arrPrices = new double[2500];
   static double normInputXPointValue = 0.00;
   static double normPredictXPointValue = 0.00;
   static double normTargetXPointValue = 0.00;
   static double normDifferencePerc = 0.00;
   static double returnCode = 0.00;
   static double denormInputXPointValue = 0.00;
```

```
  static double denormPredictXPointValue = 0.00;
  static double denormTargetXPointValue = 0.00;
  static double valueDifference = 0.00;
  static int numberOfInputNeurons;
  static int numberOfOutputNeurons;
   static int numberOfRecordsInFile;
  static String trainFileName;
  static String priceFileName;
  static String testFileName;
  static String chartTrainFileName;
  static String chartTestFileName;
  static String networkFileName;
  static int workingMode;
  static String cvsSplitBy = ",";

  static List<Double> xData = new ArrayList<Double>();
  static List<Double> yData1 = new ArrayList<Double>();
  static List<Double> yData2 = new ArrayList<Double>();

  static XYChart Chart;

 @Override
 public XYChart getChart()
  {
   // Create Chart

   XYSeries series1 = Chart.addSeries("Actual data", xData, yData1);
       XYSeries series2 = Chart.addSeries("Predict data", xData, yData2);

       series1.setLineColor(XChartSeriesColors.BLACK);
       series2.setLineColor(XChartSeriesColors.YELLOW);

       series1.setMarkerColor(Color.BLACK);
       series2.setMarkerColor(Color.WHITE);
       series1.setLineStyle(SeriesLines.SOLID);
       series2.setLineStyle(SeriesLines.DASH_DASH);
```

```
try
   {
     // Configuration

     // Set the mode the program should run
     workingMode = 1; // Run the program in the training mode

     if(workingMode == 1)
       {
         // Training mode
         numberOfRecordsInFile = 10001;
         trainFileName = "C:/Article_To_Publish/IGI_Global/Complex
         Formula_Calculate_Train_Norm.csv";
         chartTrainFileName = "C:/Article_To_Publish/IGI_Global/Complex
         Formula_Chart_Train_Results";
       }
     else
       {
          // Testing mode
          numberOfRecordsInFile = 10001;
           testFileName = "C:/Article_To_Publish/IGI_Global/Complex
           Formula_Calculate_Test_Norm.csv";
           chartTestFileName = "C:/Article_To_Publish/IGI_Global/
           ComplexFormula_Chart_Test_Results";
       }

     // Common part of config data
     networkFileName = "C:/Article_To_Publish/IGI_Global/Complex
     Formula_Saved_Network_File.csv";
     numberOfInputNeurons = 1;
     numberOfOutputNeurons = 1;

     if(workingMode == 1)
      {
        // Training mode
        File file1 = new File(chartTrainFileName);
        File file2 = new File(networkFileName);
```

```
            if(file1.exists())
              file1.delete();

            if(file2.exists())
              file2.delete();

            returnCode = 0;    // Clear the error Code

            do
             {
               returnCode = trainValidateSaveNetwork();
             }  while (returnCode > 0);
          }
         else
          {
             // Test mode
             loadAndTestNetwork();
          }
       }
      catch (Throwable t)
        {
          t.printStackTrace();
          System.exit(1);
        }
       finally
        {
          Encog.getInstance().shutdown();
        }
    Encog.getInstance().shutdown();

    return Chart;

  }  // End of the method

  // =======================================================
  // Load CSV to memory.
  // @return The loaded dataset.
  // =======================================================
```

```
public static MLDataSet loadCSV2Memory(String filename, int input, int
ideal, boolean headers, CSVFormat format, boolean significance)
  {
     DataSetCODEC codec = new CSVDataCODEC(new File(filename), format,
     headers, input, ideal, significance);
     MemoryDataLoader load = new MemoryDataLoader(codec);
     MLDataSet dataset = load.external2Memory();
     return dataset;
  }

// =====================================================
//  The main method.
//  @param Command line arguments. No arguments are used.
// =====================================================
public static void main(String[] args)
 {
   ExampleChart<XYChart> exampleChart = new ArticleIDI_ComplexFormula_
   Traditional();
   XYChart Chart = exampleChart.getChart();
   new SwingWrapper<XYChart>(Chart).displayChart();
 } // End of the main method

//======================================================================
// This method trains, Validates, and saves the trained network file
//======================================================================
static public double trainValidateSaveNetwork()
 {
   // Load the training CSV file in memory
   MLDataSet trainingSet =
     loadCSV2Memory(trainFileName,numberOfInputNeurons,numberOf
     OutputNeurons,
       true,CSVFormat.ENGLISH,false);

   // create a neural network
   BasicNetwork network = new BasicNetwork();
```

```
// Input layer
network.addLayer(new BasicLayer(null,true,1));

// Hidden layer
network.addLayer(new BasicLayer(new ActivationTANH(),true,7));
network.addLayer(new BasicLayer(new ActivationTANH(),true,7));
network.addLayer(new BasicLayer(new ActivationTANH(),true,7));
network.addLayer(new BasicLayer(new ActivationTANH(),true,7));
network.addLayer(new BasicLayer(new ActivationTANH(),true,7));
network.addLayer(new BasicLayer(new ActivationTANH(),true,7));
network.addLayer(new BasicLayer(new ActivationTANH(),true,7));

// Output layer
network.addLayer(new BasicLayer(new ActivationTANH(),false,1));

network.getStructure().finalizeStructure();
network.reset();

//Train the neural network
final ResilientPropagation train = new ResilientPropagation
(network, trainingSet);

int epoch = 1;

do
 {
  train.iteration();
  System.out.println("Epoch #" + epoch + " Error:" + train.getError());

   epoch++;

   if (epoch >= 6000 && network.calculateError(trainingSet) > 0.101)
       {
        returnCode = 1;

        System.out.println("Try again");
        return returnCode;
       }
 } while(train.getError() > 0.10);
```

```
// Save the network file
EncogDirectoryPersistence.saveObject(new File(networkFileName),
network);

System.out.println("Neural Network Results:");

double sumNormDifferencePerc = 0.00;
double averNormDifferencePerc = 0.00;
double maxNormDifferencePerc = 0.00;

int m = 0;

double stepValue = 0.00031;
double startingPoint = 1.00;
double xPoint = startingPoint - stepValue;

for(MLDataPair pair: trainingSet)
  {
      m++;
      xPoint = xPoint + stepValue;

      if(m == 0)
       continue;

       final MLData output = network.compute(pair.getInput());

       MLData inputData = pair.getInput();
       MLData actualData = pair.getIdeal();
       MLData predictData = network.compute(inputData);

       // Calculate and print the results
       normInputXPointValue = inputData.getData(0);
       normTargetXPointValue = actualData.getData(0);
       normPredictXPointValue = predictData.getData(0);

       denormInputXPointValue = ((minXPointDl - maxXPointDh)*
       normInputXPointValue -Nh*minXPointDl + maxXPointDh *Nl)/
       (Nl - Nh);
```

```
        denormTargetXPointValue =((minTargetValueDl - maxTargetValueDh)*
        normTargetXPointValue - Nh*minTargetValueDl + maxTarget
        ValueDh*Nl)/(Nl - Nh);

        denormPredictXPointValue =((minTargetValueDl - maxTargetValueDh)*
        normPredictXPointValue - Nh*minTargetValueDl +
        maxTargetValueDh*Nl)/(Nl - Nh);

        valueDifference =
          Math.abs(((denormTargetXPointValue - denormPredictXPointValue)/
          denormTargetXPointValue)*100.00);

      System.out.println ("xPoint = " + xPoint + " denormTargetXPoint
      Value = " + denormTargetXPointValue + "denormPredictXPointValue = "
                + denormPredictXPointValue + " valueDifference = " +
      valueDifference);

        sumNormDifferencePerc = sumNormDifferencePerc + valueDifference;

        if (valueDifference > maxNormDifferencePerc)
          maxNormDifferencePerc = valueDifference;

        xData.add(xPoint);
        yData1.add(denormTargetXPointValue);
        yData2.add(denormPredictXPointValue);

    }   // End for pair loop

    XYSeries series1 = Chart.addSeries("Actual data", xData, yData1);
    XYSeries series2 = Chart.addSeries("Predict data", xData, yData2);

    series1.setLineColor(XChartSeriesColors.BLACK);
    series2.setLineColor(XChartSeriesColors.YELLOW);

    series1.setMarkerColor(Color.BLACK);
    series2.setMarkerColor(Color.WHITE);
    series1.setLineStyle(SeriesLines.SOLID);
    series2.setLineStyle(SeriesLines.DASH_DASH);
```

```
     try
      {
        //Save the chart image
        BitmapEncoder.saveBitmapWithDPI(Chart, chartTrainFileName,
          BitmapFormat.JPG, 100);
        System.out.println ("Train Chart file has been saved") ;
      }
    catch (IOException ex)
     {
      ex.printStackTrace();
      System.exit(3);
     }

     // Finally, save this trained network
     EncogDirectoryPersistence.saveObject(new File(networkFileName),
     network);
     System.out.println ("Train Network has been saved");

     averNormDifferencePerc  = sumNormDifferencePerc/(numberOfRecords
     InFile-1);

     System.out.println(" ");
     System.out.println("maxErrorDifferencePerc = " + maxNormDifference
     Perc + " averErrorDifferencePerc = " + averNormDifferencePerc);

     returnCode = 0.00;
     return returnCode;

 }   // End of the method

//=================================================
// This method load and test the trained network
//=================================================
static public void loadAndTestNetwork()
 {
  System.out.println("Testing the networks results");
```

```
List<Double> xData = new ArrayList<Double>();
List<Double> yData1 = new ArrayList<Double>();
List<Double> yData2 = new ArrayList<Double>();

double targetToPredictPercent = 0;
double maxGlobalResultDiff = 0.00;
double averGlobalResultDiff = 0.00;
double sumGlobalResultDiff = 0.00;
double maxGlobalIndex = 0;
double normInputXPointValueFromRecord = 0.00;
double normTargetXPointValueFromRecord = 0.00;
double normPredictXPointValueFromRecord = 0.00;

BasicNetwork network;

maxGlobalResultDiff = 0.00;
averGlobalResultDiff = 0.00;
sumGlobalResultDiff = 0.00;

// Load the test dataset into memory
MLDataSet testingSet =
loadCSV2Memory(testFileName,numberOfInputNeurons,numberOfOutput
Neurons,true,CSVFormat.ENGLISH,false);

// Load the saved trained network
network =
  (BasicNetwork)EncogDirectoryPersistence.loadObject(new File(network
  FileName));

int i = - 1; // Index of the current record
double stepValue = 0.000298;
double startingPoint = 1.01;
double xPoint = startingPoint - stepValue;

for (MLDataPair pair:  testingSet)
 {
      i++;
      xPoint = xPoint + stepValue;
```

```
        MLData inputData = pair.getInput();
        MLData actualData = pair.getIdeal();
        MLData predictData = network.compute(inputData);

        // These values are Normalized as the whole input is
        normInputXPointValueFromRecord = inputData.getData(0);
        normTargetXPointValueFromRecord = actualData.getData(0);
        normPredictXPointValueFromRecord = predictData.getData(0);

        denormInputXPointValue = ((minXPointDl - maxXPointDh)*
          normInputXPointValueFromRecord - Nh*minXPointDl +
          maxXPointDh*Nl)/(Nl - Nh);
        denormTargetXPointValue = ((minTargetValueDl - maxTargetValueDh)*
          normTargetXPointValueFromRecord - Nh*minTargetValueDl + maxTarget
          ValueDh*Nl)/(Nl - Nh);
        denormPredictXPointValue =((minTargetValueDl - maxTargetValueDh)*
          normPredictXPointValueFromRecord - Nh*minTargetValueDl + maxTarget
          ValueDh*Nl)/(Nl - Nh);

        targetToPredictPercent = Math.abs((denormTargetXPointValue -
          denormPredictXPointValue)/denormTargetXPointValue*100);

        System.out.println("xPoint = " + xPoint +  "  denormTargetX
        PointValue = " + denormTargetXPointValue + "  denormPredictX
        PointValue = " + denormPredictXPointValue +"  targetToPredict
        Percent = " + targetToPredictPercent);

        if (targetToPredictPercent > maxGlobalResultDiff)
           maxGlobalResultDiff = targetToPredictPercent;

        sumGlobalResultDiff = sumGlobalResultDiff + targetToPredict
        Percent;

        // Populate chart elements
        xData.add(xPoint);
        yData1.add(denormTargetXPointValue);
        yData2.add(denormPredictXPointValue);

    }  // End for pair loop
```

```
        // Print the max and average results
        System.out.println(" ");
        averGlobalResultDiff = sumGlobalResultDiff/(numberOfRecordsInFile-1);

        System.out.println("maxErrorPerc = " + maxGlobalResultDiff);
        System.out.println("averErrorPerc = " + averGlobalResultDiff);

        // All testing batch files have been processed
        XYSeries series1 = Chart.addSeries("Actual", xData, yData1);
        XYSeries series2 = Chart.addSeries("Predicted", xData, yData2);

        series1.setLineColor(XChartSeriesColors.BLACK);
        series2.setLineColor(XChartSeriesColors.YELLOW);

        series1.setMarkerColor(Color.BLACK);
        series2.setMarkerColor(Color.WHITE);
        series1.setLineStyle(SeriesLines.SOLID);
        series2.setLineStyle(SeriesLines.DASH_DASH);

        // Save the chart image
        try
         {
           BitmapEncoder.saveBitmapWithDPI(Chart, chartTestFileName ,
           BitmapFormat.JPG, 100);
         }
        catch (Exception bt)
         {
           bt.printStackTrace();
         }

        System.out.println ("The Chart has been saved");
        System.out.println("End of testing for test records");

    } // End of the method

} // End of the class
```

Training Processing Results for Example 5a

Listing 9-2 shows the end fragment of the conventional network processing results.

Listing 9-2. The End Fragment of the Conventional Training Results

```
xPoint = 4.08605  TargetValue = 1.24795  PredictedValue = 1.15899
DifPerc = 7.12794
xPoint = 4.08636  TargetValue = 1.25699  PredictedValue = 1.16125
DifPerc = 7.61624
xPoint = 4.08667  TargetValue = 1.26602  PredictedValue = 1.16346
DifPerc = 8.10090
xPoint = 4.08698  TargetValue = 1.27504  PredictedValue = 1.16562
DifPerc = 8.58150
xPoint = 4.08729  TargetValue = 1.28404  PredictedValue = 1.16773
DifPerc = 9.05800
xPoint = 4.08760  TargetValue = 1.29303  PredictedValue = 1.16980
DifPerc = 9.53011
xPoint = 4.08791  TargetValue = 1.30199  PredictedValue = 1.17183
DifPerc = 9.99747
xPoint = 4.08822  TargetValue = 1.31093  PredictedValue = 1.17381
DifPerc = 10.4599
xPoint = 4.08853  TargetValue = 1.31984  PredictedValue = 1.17575
DifPerc = 10.9173
xPoint = 4.08884  TargetValue = 1.32871  PredictedValue = 1.17765
DifPerc = 11.3694
xPoint = 4.08915  TargetValue = 1.33755  PredictedValue = 1.17951
DifPerc = 11.8159
xPoint = 4.08946  TargetValue = 1.34635  PredictedValue = 1.18133
DifPerc = 12.25680
xPoint = 4.08978  TargetValue = 1.35510  PredictedValue = 1.18311
DifPerc = 12.69162
xPoint = 4.09008  TargetValue = 1.36380  PredictedValue = 1.18486
DifPerc = 13.12047
xPoint = 4.09039  TargetValue = 1.37244  PredictedValue = 1.18657
DifPerc = 13.54308
```

```
xPoint = 4.09070  TargetValue = 1.38103  PredictedValue = 1.18825
DifPerc = 13.95931
xPoint = 4.09101  TargetValue = 1.38956  PredictedValue = 1.18999
DifPerc = 14.36898
xPoint = 4.09132  TargetValue = 1.39802  PredictedValue = 1.19151
DifPerc = 14.77197
xPoint = 4.09164  TargetValue = 1.40642  PredictedValue = 1.19309
DifPerc = 15.16812
xPoint = 4.09194  TargetValue = 1.41473  PredictedValue = 1.19464
DifPerc = 15.55732
xPoint = 4.09225  TargetValue = 1.42297  PredictedValue = 1.19616
DifPerc = 15.93942
xPoint = 4.09256  TargetValue = 1.43113  PredictedValue = 1.19765
DifPerc = 16.31432
xPoint = 4.09287  TargetValue = 1.43919  PredictedValue = 1.19911
DifPerc = 16.68189
xPoint = 4.09318  TargetValue = 1.44717  PredictedValue = 1.20054
DifPerc = 17.04203
xPoint = 4.09349  TargetValue = 1.45505  PredictedValue = 1.20195
DifPerc = 17.39463
xPoint = 4.09380  TargetValue = 1.46283  PredictedValue = 1.20333
DifPerc = 17.73960
xPoint = 4.09411  TargetValue = 1.47051  PredictedValue = 1.20469
DifPerc = 18.07683
xPoint = 4.09442  TargetValue = 1.47808  PredictedValue = 1.20602
DifPerc = 18.40624
xPoint = 4.09473  TargetValue = 1.48553  PredictedValue = 1.20732
DifPerc = 18.72775
xPoint = 4.09504  TargetValue = 1.49287  PredictedValue = 1.20861
DifPerc = 19.04127
xPoint = 4.09535  TargetValue = 1.50009  PredictedValue = 1.20987
DifPerc = 19.34671
xPoint = 4.09566  TargetValue = 1.50718  PredictedValue = 1.21111
DifPerc = 19.64402
xPoint = 4.09597  TargetValue = 1.51414  PredictedValue = 1.21232
DifPerc = 19.93312
```

```
xPoint = 4.09628  TargetValue = 1.52097  PredictedValue = 1.21352
DifPerc = 20.21393
xPoint = 4.09659  TargetValue = 1.52766  PredictedValue = 1.21469
DifPerc = 20.48640
xPoint = 4.09690  TargetValue = 1.53420  PredictedValue = 1.21585
DifPerc = 20.75045
xPoint = 4.09721  TargetValue = 1.54060  PredictedValue = 1.21699
DifPerc = 21.00605
xPoint = 4.09752  TargetValue = 1.54686  PredictedValue = 1.21810
DifPerc = 21.25312
xPoint = 4.09783  TargetValue = 1.55296  PredictedValue = 1.21920
DifPerc = 21.49161
xPoint = 4.09814  TargetValue = 1.55890  PredictedValue = 1.22028
DifPerc = 21.72147
xPoint = 4.09845  TargetValue = 1.56468  PredictedValue = 1.22135
DifPerc = 21.94265
xPoint = 4.09876  TargetValue = 1.57030  PredictedValue = 1.22239
DifPerc = 22.15511
xPoint = 4.09907  TargetValue = 1.57574  PredictedValue = 1.22342
DifPerc = 22.35878
xPoint = 4.09938  TargetValue = 1.58101  PredictedValue = 1.22444
DifPerc = 22.55363
xPoint = 4.09969  TargetValue = 1.58611  PredictedValue = 1.22544
DifPerc = 22.73963

maxErrorPerc = 86.08183780343387
averErrorPerc = 10.116005438206885
```

With the conventional process, the approximation results are as follows:

- The maximum error percent exceeds 86.08 percent.
- The average error percent exceeds 10. 11 percent.

Figure 9-3 shows the chart of the training approximation results using conventional network processing.

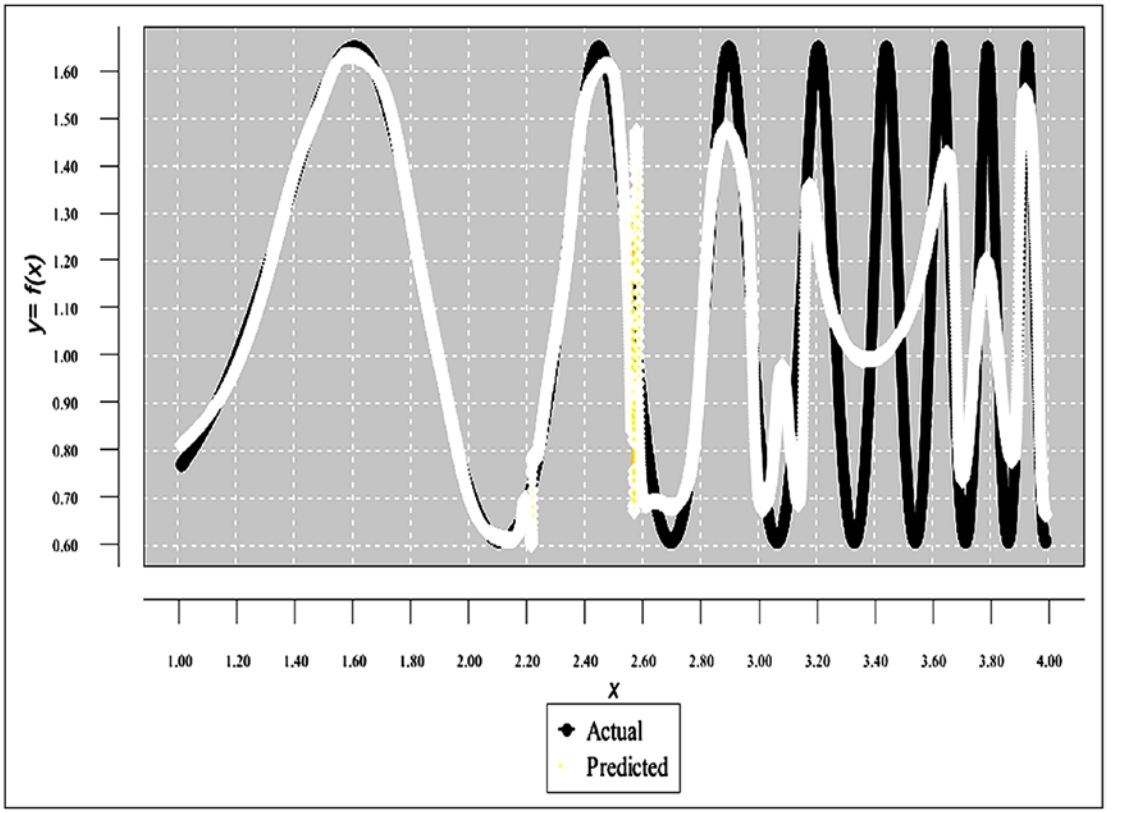


Figure 9-3. *Chart of the training approximation results using conventional network processing*

Obviously, with such a large difference between the actual and predicted values, such an approximation is useless.

Approximation of a Continuous Function with Complex Topology Using the Micro-Batch Method

Now, let's approximate this function using the micro-batch method. Again, the normalized training data set is broken into a set of training micro-batch files, and they then become the input to the training process. Listing 9-3 shows the ending fragment of the training processing results (using the macro-batch method) after execution.

Listing 9-3. Ending Fragment of the Training Processing Results (Using the Macro-Batch Method)

```
DayNumber = 9950  TargetValue = 1.19376  PredictedValue = 1.19376
DiffPerc = 4.66352E-6
DayNumber = 9951  TargetValue = 1.20277  PredictedValue = 1.20277
DiffPerc = 5.30417E-6
DayNumber = 9952  TargetValue = 1.21180  PredictedValue = 1.21180
DiffPerc = 4.79291E-6
DayNumber = 9953  TargetValue = 1.22083  PredictedValue = 1.22083
DiffPerc = 5.03070E-6
DayNumber = 9954  TargetValue = 1.22987  PredictedValue = 1.22987
DiffPerc = 3.79647E-6
DayNumber = 9955  TargetValue = 1.23891  PredictedValue = 1.23891
DiffPerc = 8.06431E-6
DayNumber = 9956  TargetValue = 1.24795  PredictedValue = 1.24795
DiffPerc = 7.19851E-6
DayNumber = 9957  TargetValue = 1.25699  PredictedValue = 1.25699
DiffPerc = 4.57148E-6
DayNumber = 9958  TargetValue = 1.26602  PredictedValue = 1.26602
DiffPerc = 5.88300E-6
DayNumber = 9959  TargetValue = 1.27504  PredictedValue = 1.27504
DiffPerc = 3.02448E-6
DayNumber = 9960  TargetValue = 1.28404  PredictedValue = 1.28404
DiffPerc = 7.04155E-6
DayNumber = 9961  TargetValue = 1.29303  PredictedValue = 1.29303
DiffPerc = 8.62206E-6
DayNumber = 9962  TargetValue = 1.30199  PredictedValue = 1.30199
DiffPerc = 9.16473E-8
DayNumber = 9963  TargetValue = 1.31093  PredictedValue = 1.31093
DiffPerc = 1.89459E-6
DayNumber = 9964  TargetValue = 1.31984  PredictedValue = 1.31984
DiffPerc = 4.16695E-6
DayNumber = 9965  TargetValue = 1.32871  PredictedValue = 1.32871
DiffPerc = 8.68118E-6
```

```
DayNumber = 9966  TargetValue = 1.33755  PredictedValue = 1.33755
DiffPerc = 4.55866E-6
DayNumber = 9967  TargetValue = 1.34635  PredictedValue = 1.34635
DiffPerc = 6.67697E-6
DayNumber = 9968  TargetValue = 1.35510  PredictedValue = 1.35510
DiffPerc = 4.80264E-6
DayNumber = 9969  TargetValue = 1.36378  PredictedValue = 1.36380
DiffPerc = 8.58688E-7
DayNumber = 9970  TargetValue = 1.37244  PredictedValue = 1.37245
DiffPerc = 5.19317E-6
DayNumber = 9971  TargetValue = 1.38103  PredictedValue = 1.38104
DiffPerc = 7.11052E-6
DayNumber = 9972  TargetValue = 1.38956  PredictedValue = 1.38956
DiffPerc = 5.15382E-6
DayNumber = 9973  TargetValue = 1.39802  PredictedValue = 1.39802
DiffPerc = 5.90734E-6
DayNumber = 9974  TargetValue = 1.40642  PredictedValue = 1.40642
DiffPerc = 6.20744E-7
DayNumber = 9975  TargetValue = 1.41473  PredictedValue = 1.41473
DiffPerc = 5.67234E-7
DayNumber = 9976  TargetValue = 1.42297  PredictedValue = 1.42297
DiffPerc = 5.54862E-6
DayNumber = 9977  TargetValue = 1.43113  PredictedValue = 1.43113
DiffPerc = 3.28318E-6
DayNumber = 9978  TargetValue = 1.43919  PredictedValue = 1.43919
DiffPerc = 7.84136E-6
DayNumber = 9979  TargetValue = 1.44717  PredictedValue = 1.44717
DiffPerc = 6.51767E-6
DayNumber = 9980  TargetValue = 1.45505  PredictedValue = 1.45505
DiffPerc = 6.59220E-6
DayNumber = 9981  TargetValue = 1.46283  PredictedValue = 1.46283
DiffPerc = 9.08060E-7
DayNumber = 9982  TargetValue = 1.47051  PredictedValue = 1.47051
DiffPerc = 8.59549E-6
DayNumber = 9983  TargetValue = 1.47808  PredictedValue = 1.47808
DiffPerc = 5.49575E-7
```

```
DayNumber = 9984  TargetValue = 1.48553  PredictedValue = 1.48553
DiffPerc = 1.07879E-6
DayNumber = 9985  TargetValue = 1.49287  PredictedValue = 1.49287
DiffPerc = 2.22734E-6
DayNumber = 9986  TargetValue = 1.50009  PredictedValue = 1.50009
DiffPerc = 1.28405E-6
DayNumber = 9987  TargetValue = 1.50718  PredictedValue = 1.50718
DiffPerc = 8.88272E-6
DayNumber = 9988  TargetValue = 1.51414  PredictedValue = 1.51414
DiffPerc = 4.91930E-6
DayNumber = 9989  TargetValue = 1.52097  PredictedValue = 1.52097
DiffPerc = 3.46714E-6
DayNumber = 9990  TargetValue = 1.52766  PredictedValue = 1.52766
DiffPerc = 7.67496E-6
DayNumber = 9991  TargetValue = 1.53420  PredictedValue = 1.53420
DiffPerc = 4.67918E-6
DayNumber = 9992  TargetValue = 1.54061  PredictedValue = 1.54061
DiffPerc = 2.20484E-6
DayNumber = 9993  TargetValue = 1.54686  PredictedValue = 1.54686
DiffPerc = 7.42466E-6
DayNumber = 9994  TargetValue = 1.55296  PredictedValue = 1.55296
DiffPerc = 3.86183E-6
DayNumber = 9995  TargetValue = 1.55890  PredictedValue = 1.55890
DiffPerc = 6.34568E-7
DayNumber = 9996  TargetValue = 1.56468  PredictedValue = 1.56468
DiffPerc = 6.23860E-6
DayNumber = 9997  TargetValue = 1.57029  PredictedValue = 1.57029
DiffPerc = 3.66380E-7
DayNumber = 9998  TargetValue = 1.57574  PredictedValue = 1.57574
DiffPerc = 4.45560E-6
DayNumber = 9999  TargetValue = 1.58101  PredictedValue = 1.58101
DiffPerc = 6.19952E-6
DayNumber = 10000  TargetValue = 1.5861  PredictedValue = 1.58611
DiffPerc = 1.34336E-6

maxGlobalResultDiff = 1.3433567671366473E-6
averGlobalResultDiff = 2.686713534273295E-10
```

The training processing results (that uses the micro-batch method) are as follows:

1. The maximum error is less than 0.00000134 percent.
2. The average error is less than 0.000000000269 percent.

Figure 9-4 shows the chart of the training approximation results (using the micro-batch method). Both charts are practically equal (actual values are black, and predicted values are white).

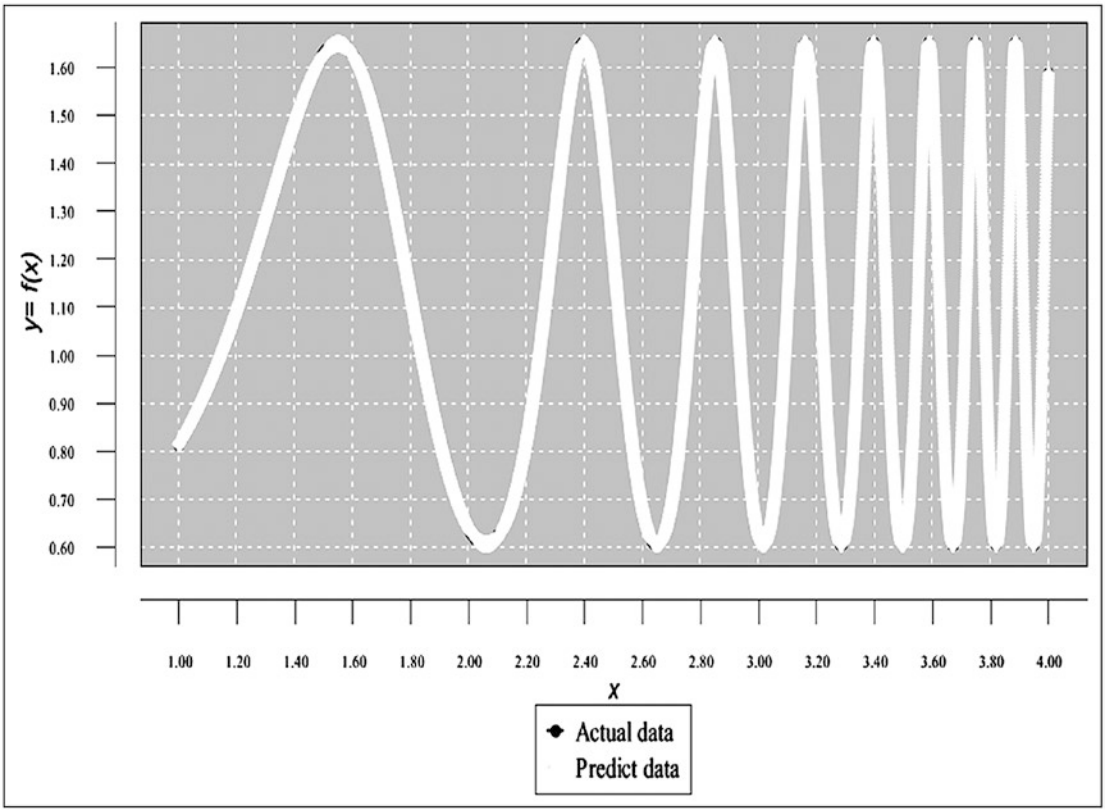


Figure 9-4. *Chart of the training approximation results (using the micro-batch method)*

Testing Processing for Example 5a

Like with the normalization of the training data set, the normalized testing data set is broken into a set of micro-batch files that are now the input to the testing process.

Listing 9-4 shows the program code.

Listing 9-4. Program Code

```
// ====================================================
// Approximation of continuous function with complex topology
// using the micro-batch method. The input is the normalized set of
// micro-batch files. Each micro-batch includes a single day record
// that contains two fields:
// - normDayValue
// - normTargetValue
//
// The number of inputLayer neurons is 1
// The number of outputLayer neurons is 1
// ====================================================

package articleigi_complexformula_microbatchest;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.*;
import java.util.Properties;
import java.time.YearMonth;
import java.awt.Color;
import java.awt.Font;
import java.io.BufferedReader;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.LocalDate;
```

```
import java.time.Month;
import java.time.ZoneId;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.List;
import java.util.Locale;
import java.util.Properties;

import org.encog.Encog;
import org.encog.engine.network.activation.ActivationTANH;
import org.encog.engine.network.activation.ActivationReLU;
import org.encog.ml.data.MLData;
import org.encog.ml.data.MLDataPair;
import org.encog.ml.data.MLDataSet;
import org.encog.ml.data.buffer.MemoryDataLoader;
import org.encog.ml.data.buffer.codec.CSVDataCODEC;
import org.encog.ml.data.buffer.codec.DataSetCODEC;
import org.encog.neural.networks.BasicNetwork;
import org.encog.neural.networks.layers.BasicLayer;
import org.encog.neural.networks.training.propagation.resilient.
ResilientPropagation;
import org.encog.persist.EncogDirectoryPersistence;
import org.encog.util.csv.CSVFormat;

import org.knowm.xchart.SwingWrapper;
import org.knowm.xchart.XYChart;
import org.knowm.xchart.XYChartBuilder;
import org.knowm.xchart.XYSeries;
import org.knowm.xchart.demo.charts.ExampleChart;
import org.knowm.xchart.style.Styler.LegendPosition;
import org.knowm.xchart.style.colors.ChartColor;
import org.knowm.xchart.style.colors.XChartSeriesColors;
import org.knowm.xchart.style.lines.SeriesLines;
import org.knowm.xchart.style.markers.SeriesMarkers;
import org.knowm.xchart.BitmapEncoder;
import org.knowm.xchart.BitmapEncoder.BitmapFormat;
```

```
import org.knowm.xchart.QuickChart;
import org.knowm.xchart.SwingWrapper;

public class ArticleIGI_ComplexFormula_Microbatchest implements
ExampleChart<XYChart>
{
   // Normalization parameters

   // Normalizing interval
   static double Nh =  1;
   static double Nl = -1;

   // First 1
   static double minXPointDl = 0.95;
   static double maxXPointDh = 4.05;

   // Column 2
   static double minTargetValueDl = 0.60;
   static double maxTargetValueDh = 1.65;

   static String cvsSplitBy = ",";
   static Properties prop = null;

   static String strWorkingMode;
   static String strNumberOfBatchesToProcess;
   static String strTrainFileNameBase;
   static String strTestFileNameBase;
   static String strSaveTrainNetworkFileBase;
   static String strSaveTestNetworkFileBase;
   static String strValidateFileName;
   static String strTrainChartFileName;
   static String strTestChartFileName;
   static String strFunctValueTrainFile;
   static String strFunctValueTestFile;
   static int intDayNumber;
   static double doubleDayNumber;
   static int intWorkingMode;
   static int numberOfTrainBatchesToProcess;
   static int numberOfTestBatchesToProcess;
```

```
  static int intNumberOfRecordsInTrainFile;
  static int intNumberOfRecordsInTestFile;
  static int intNumberOfRowsInBatches;
  static int intInputNeuronNumber;
  static int intOutputNeuronNumber;
  static String strOutputFileName;
  static String strSaveNetworkFileName;
  static String strDaysTrainFileName;
  static XYChart Chart;
  static String iString;
  static double inputFunctValueFromFile;
  static double targetToPredictFunctValueDiff;
  static int[] returnCodes  = new int[3];

  static List<Double> xData = new ArrayList<Double>();
  static List<Double> yData1 = new ArrayList<Double>();
  static List<Double> yData2 = new ArrayList<Double>();

  static double[] DaysyearDayTraining = new double[10200];
  static String[] strTrainingFileNames = new String[10200];
  static String[] strTestingFileNames = new String[10200];
  static String[] strSaveTrainNetworkFileNames = new String[10200];
  static double[] linkToSaveNetworkDayKeys = new double[10200];
  static double[] linkToSaveNetworkTargetFunctValueKeys = new double[10200];
  static double[] arrTrainFunctValues = new double[10200];
  static double[] arrTestFunctValues = new double[10200];

@Override
public XYChart getChart()
 {
   // Create Chart

  Chart = new XYChartBuilder().width(900).height(500).title(getClass().
    getSimpleName()).xAxisTitle("day").yAxisTitle("y=f(day)").build();

  // Customize Chart
  Chart = new  XYChartBuilder().width(900).height(500).title(getClass().
          getSimpleName()).xAxisTitle("x").yAxisTitle("y= f(x)").build();
```

```
// Customize Chart
Chart.getStyler().setPlotBackgroundColor(ChartColor.getAWTColor
(ChartColor.GREY));
Chart.getStyler().setPlotGridLinesColor(new Color(255, 255, 255));

//Chart.getStyler().setPlotBackgroundColor(ChartColor.
getAWTColor(ChartColor.WHITE));
//Chart.getStyler().setPlotGridLinesColor(new Color(0, 0, 0));
Chart.getStyler().setChartBackgroundColor(Color.WHITE);
//Chart.getStyler().setLegendBackgroundColor(Color.PINK);
Chart.getStyler().setLegendBackgroundColor(Color.WHITE);
//Chart.getStyler().setChartFontColor(Color.MAGENTA);
Chart.getStyler().setChartFontColor(Color.BLACK);
Chart.getStyler().setChartTitleBoxBackgroundColor(new Color(0, 222, 0));
Chart.getStyler().setChartTitleBoxVisible(true);
Chart.getStyler().setChartTitleBoxBorderColor(Color.BLACK);
Chart.getStyler().setPlotGridLinesVisible(true);
Chart.getStyler().setAxisTickPadding(20);
Chart.getStyler().setAxisTickMarkLength(15);
Chart.getStyler().setPlotMargin(20);
Chart.getStyler().setChartTitleVisible(false);
Chart.getStyler().setChartTitleFont(new Font(Font.MONOSPACED,
Font.BOLD, 24));
Chart.getStyler().setLegendFont(new Font(Font.SERIF, Font.PLAIN, 18));
//Chart.getStyler().setLegendPosition(LegendPosition.InsideSE);
Chart.getStyler().setLegendPosition(LegendPosition.OutsideS);
Chart.getStyler().setLegendSeriesLineLength(12);
Chart.getStyler().setAxisTitleFont(new Font(Font.SANS_SERIF,
Font.ITALIC, 18));
Chart.getStyler().setAxisTickLabelsFont(new Font(Font.SERIF,
Font.PLAIN, 11));
Chart.getStyler().setDatePattern("yyyy-MM");
Chart.getStyler().setDecimalPattern("#0.00");

// Config data
```

```
// Set the mode the program should run
intWorkingMode = 1;  // Training mode

if ( intWorkingMode == 1)
  {
numberOfTrainBatchesToProcess = 10000;
numberOfTestBatchesToProcess = 9999;
intNumberOfRowsInBatches = 1;
intInputNeuronNumber = 1;
intOutputNeuronNumber = 1;
strTrainFileNameBase = "C:/Article_To_Publish/IGI_Global/Work_Files_
ComplexFormula/ComplexFormula_Train_Norm_Batch_";
strTestFileNameBase = "C:/Article_To_Publish/IGI_Global/Work_Files_
ComplexFormula/ComplexFormula_Test_Norm_Batch_";
strSaveTrainNetworkFileBase =
  "C:/Article_To_Publish/IGI_Global/Work_Files_ComplexFormula/Save_
  Network_MicroBatch_";
strTrainChartFileName =
  "C:/Article_To_Publish/IGI_Global/Chart_Microbatch_Train_Results.jpg";
strTestChartFileName =
  "C:/Article_To_Publish/IGI_Global/Chart_Microbatch_Test_MicroBatch.jpg";

// Generate training batch file names and the corresponding
// SaveNetwork file names

intDayNumber = -1;  // Day number for the chart

for (int i = 0; i < numberOfTrainBatchesToProcess; i++)
 {
   intDayNumber++;

   iString = Integer.toString(intDayNumber);

   strOutputFileName = strTrainFileNameBase + iString + ".csv";

   strSaveNetworkFileName = strSaveTrainNetworkFileBase + iString + ".csv";

   strTrainingFileNames[intDayNumber] = strOutputFileName;
   strSaveTrainNetworkFileNames[intDayNumber] = strSaveNetworkFileName;
```

```
}  // End the FOR loop

 // Build the array linkToSaveNetworkFunctValueDiffKeys
 String tempLine;
 double tempNormFunctValueDiff = 0.00;
 double tempNormFunctValueDiffPerc = 0.00;
 double tempNormTargetFunctValueDiffPerc = 0.00;

 String[] tempWorkFields;

 try
  {
     intDayNumber = -1;  // Day number for the chart

     for (int m = 0; m < numberOfTrainBatchesToProcess; m++)
       {
          intDayNumber++;

          BufferedReader br3 = new BufferedReader(new
              FileReader(strTrainingFileNames[intDayNumber]));
          tempLine = br3.readLine();

          // Skip the label record and zero batch record
          tempLine = br3.readLine();

          // Break the line using comma as separator
          tempWorkFields = tempLine.split(cvsSplitBy);

          tempNormFunctValueDiffPerc = Double.parseDouble(tempWork
          Fields[0]);
          tempNormTargetFunctValueDiffPerc = Double.parseDouble
          (tempWorkFields[1]);

          linkToSaveNetworkDayKeys[intDayNumber] = tempNormFunctValue
          DiffPerc;
          linkToSaveNetworkTargetFunctValueKeys[intDayNumber] =
               tempNormTargetFunctValueDiffPerc;

      }  // End the FOR loop

   }
```

```
else
  {
      // Testing mode
      // Generate testing batch file names

       intDayNumber = -1;

      for (int i = 0; i < numberOfTestBatchesToProcess; i++)
       {
         intDayNumber++;
         iString = Integer.toString(intDayNumber);

         // Construct the testing batch names
         strOutputFileName = strTestFileNameBase + iString + ".csv";
         strTestingFileNames[intDayNumber] = strOutputFileName;

       }  // End the FOR loop

   }   // End of IF

 }     // End for try
catch (IOException io1)
 {
   io1.printStackTrace();
   System.exit(1);
 }

 if(intWorkingMode == 1)
   {
     // Training mode

     // Load, train, and test Function Values file in memory
     loadTrainFunctValueFileInMemory();

     int paramErrorCode;
     int paramBatchNumber;
     int paramR;
     int paramDayNumber;
     int paramS;
```

```
        File file1 = new File(strTrainChartFileName);

        if(file1.exists())
          file1.delete();

        returnCodes[0] = 0;    // Clear the error Code
        returnCodes[1] = 0;    // Set the initial batch Number to 0;
        returnCodes[2] = 0;    // Day number;

       do
        {
          paramErrorCode = returnCodes[0];
          paramBatchNumber = returnCodes[1];
          paramDayNumber = returnCodes[2];

          returnCodes =
            trainBatches(paramErrorCode,paramBatchNumber,paramDayNumber);
        } while (returnCodes[0] > 0);

      }   // End the train logic
     else
      {
        // Testing mode

        File file2 = new File(strTestChartFileName);


        if(file2.exists())
          file2.delete();

        loadAndTestNetwork();


       // End the test logic
     }

     Encog.getInstance().shutdown();
     //System.exit(0);
     return Chart;

}  // End of method
```

```
// =======================================================
// Load CSV to memory.
// @return The loaded dataset.
// =======================================================
public static MLDataSet loadCSV2Memory(String filename, int input,
int ideal, boolean headers, CSVFormat format, boolean significance)
  {
     DataSetCODEC codec = new CSVDataCODEC(new File(filename), format,
     headers, inpu, ideal, significance);
     MemoryDataLoader load = new MemoryDataLoader(codec);
     MLDataSet dataset = load.external2Memory();
     return dataset;
  }

// =======================================================
//  The main method.
//  @param Command line arguments. No arguments are used.
// =======================================================
public static void main(String[] args)
 {

   ExampleChart<XYChart> exampleChart = new ArticleIGI_ComplexFormula_
   Microbatchest();
   XYChart Chart = exampleChart.getChart();
   new SwingWrapper<XYChart>(Chart).displayChart();
 } // End of the main method

//=========================================================================
// This method trains batches as individual network1s
// saving them in separate trained datasets
//=========================================================================
static public int[] trainBatches(int paramErrorCode, int paramBatchNumber,
                                 int paramDayNumber)
  {
    int rBatchNumber;
    double targetToPredictFunctValueDiff = 0;
```

```
 double maxGlobalResultDiff = 0.00;
 double averGlobalResultDiff = 0.00;
 double sumGlobalResultDiff = 0.00;

 double normInputFunctValueDiffPercFromRecord = 0.00;
 double normTargetFunctValue1 = 0.00;
 double normPredictFunctValue1 = 0.00;
 double denormInputDayFromRecord1;
 double denormInputFunctValueDiffPercFromRecord;
 double denormTargetFunctValue1 = 0.00;
 double denormAverPredictFunctValue11 = 0.00;

 BasicNetwork network1 = new BasicNetwork();

 // Input layer
 network1.addLayer(new BasicLayer(null,true,intInputNeuronNumber));

 // Hidden layer.
 network1.addLayer(new BasicLayer(new ActivationTANH(),true,7));
 network1.addLayer(new BasicLayer(new ActivationTANH(),true,7));
 network1.addLayer(new BasicLayer(new ActivationTANH(),true,7));
 network1.addLayer(new BasicLayer(new ActivationTANH(),true,7));
 network1.addLayer(new BasicLayer(new ActivationTANH(),true,7));
 network1.addLayer(new BasicLayer(new ActivationTANH(),true,7));
 network1.addLayer(new BasicLayer(new ActivationTANH(),true,7));

// Output layer
network1.addLayer(new BasicLayer(new ActivationTANH(),false, intOutput
NeuronNumber));

network1.getStructure().finalizeStructure();
network1.reset();

maxGlobalResultDiff = 0.00;
averGlobalResultDiff = 0.00;
sumGlobalResultDiff = 0.00;

// Loop over batches
intDayNumber = paramDayNumber;  // Day number for the chart
```

```
for (rBatchNumber = paramBatchNumber; rBatchNumber < numberOfTrain
BatchesToProcess; rBatchNumber++)
 {
   intDayNumber++;

  // Load the training file in memory
  MLDataSet trainingSet =
     loadCSV2Memory(strTrainingFileNames[rBatchNumber],intInput
     NeuronNumber,intOutputNeuronNumber,true,CSVFormat.ENGLISH,false);

  // train the neural network1
  ResilientPropagation train = new ResilientPropagation(network1,
  trainingSet);

  int epoch = 1;

  do
    {
      train.iteration();

      epoch++;

      for (MLDataPair pair11:  trainingSet)
       {
         MLData inputData1 = pair11.getInput();
         MLData actualData1 = pair11.getIdeal();
         MLData predictData1 = network1.compute(inputData1);

         // These values are Normalized as the whole input is
         normInputFunctValueDiffPercFromRecord = inputData1.
         getData(0);

         normTargetFunctValue1 = actualData1.getData(0);
         normPredictFunctValue1 = predictData1.getData(0);

         denormInputFunctValueDiffPercFromRecord =((minXPointDl -
         maxXPointDh)*normInputFunctValueDiffPercFromRecord - Nh*
         minXPointDl + maxXPointDh*Nl)/(Nl - Nh);
```

```
            denormTargetFunctValue1 = ((minTargetValueDl - maxTarget
            ValueDh)*normTargetFunctValue1 - Nh*minTargetValueDl +
            maxTargetValueDh*Nl)/(Nl - Nh);

            denormAverPredictFunctValue11 =((minTargetValueDl - maxTarget
            ValueDh)*normPredictFunctValue1 - Nh*minTargetValueDl +
            maxTargetValueDh*Nl)/(Nl - Nh);

            //inputFunctValueFromFile = arrTrainFunctValues[rBatchNumber];

            targetToPredictFunctValueDiff = (Math.abs(denormTarget
            FunctValue1 - denormAverPredictFunctValue11)/denormTarget
            FunctValue1)*100;
          }

         if (epoch >= 1000 && targetToPredictFunctValueDiff > 0.0000091)
          {
            returnCodes[0] = 1;
            returnCodes[1] = rBatchNumber;
            returnCodes[2] = intDayNumber-1;

            return returnCodes;
           }

        } while(targetToPredictFunctValueDiff > 0.000009);

// This batch is optimized

// Save the network1 for the cur rend batch
EncogDirectoryPersistence.saveObject(new
      File(strSaveTrainNetworkFileNames[rBatchNumber]),network1);

// Get the results after the network1 optimization
int i = - 1;

for (MLDataPair pair1:  trainingSet)
 {
  i++;

  MLData inputData1 = pair1.getInput();
  MLData actualData1 = pair1.getIdeal();
  MLData predictData1 = network1.compute(inputData1);
```

```
// These values are Normalized as the whole input is
normInputFunctValueDiffPercFromRecord = inputData1.getData(0);
normTargetFunctValue1 = actualData1.getData(0);
normPredictFunctValue1 = predictData1.getData(0);

// De-normalize the obtained values
denormInputFunctValueDiffPercFromRecord =((minXPointDl - maxXPointDh)*
normInputFunctValueDiffPercFromRecord - Nh*minXPointDl +
maxXPointDh*Nl)/(Nl - Nh);

denormTargetFunctValue1 = ((minTargetValueDl - maxTargetValueDh)*
normTargetFunctValue1 - Nh*minTargetValueDl + maxTargetValueDh*Nl)/
(Nl - Nh);

denormAverPredictFunctValue11 =((minTargetValueDl - maxTargetValueDh)*
normPredictFunctValue1 - Nh*minTargetValueDl + maxTarget
ValueDh*Nl)/(Nl - Nh);

//inputFunctValueFromFile = arrTrainFunctValues[rBatchNumber];

targetToPredictFunctValueDiff = (Math.abs(denormTargetFunctValue1 -
denormAverPredictFunctValue11)/denormTargetFunctValue1)*100;

System.out.println("intDayNumber = " + intDayNumber +  "  target
FunctionValue = " +
     denormTargetFunctValue1 + "  predictFunctionValue = " +
     denormAverPredictFunctValue11 + "  valurDiff = " + target
     ToPredictFunctValueDiff);

if (targetToPredictFunctValueDiff > maxGlobalResultDiff)
  maxGlobalResultDiff =targetToPredictFunctValueDiff;

sumGlobalResultDiff = sumGlobalResultDiff +targetToPredictFunct
ValueDiff;

// Populate chart elements
//doubleDayNumber = (double) rBatchNumber+1;
xData.add(denormInputFunctValueDiffPercFromRecord);
yData1.add(denormTargetFunctValue1);
yData2.add(denormAverPredictFunctValue11);
```

```
   }  // End for FunctValue pair1 loop

}  // End of the loop over batches

sumGlobalResultDiff = sumGlobalResultDiff +targetToPredictFunct
ValueDiff;
averGlobalResultDiff = sumGlobalResultDiff/numberOfTrainBatchesTo
Process;

// Print the max and average results

System.out.println(" ");
System.out.println(" ");
System.out.println("maxGlobalResultDiff = " + maxGlobalResultDiff);
System.out.println("averGlobalResultDiff = " + averGlobalResultDiff);

XYSeries series1 = Chart.addSeries("Actual data", xData, yData1);
XYSeries series2 = Chart.addSeries("Predict data", xData, yData2);

series1.setLineColor(XChartSeriesColors.BLACK);
series2.setLineColor(XChartSeriesColors.YELLOW);

series1.setMarkerColor(Color.BLACK);
series2.setMarkerColor(Color.WHITE);
series1.setLineStyle(SeriesLines.SOLID);
series2.setLineStyle(SeriesLines.DASH_DASH);

// Save the chart image
try
  {
    BitmapEncoder.saveBitmapWithDPI(Chart, strTrainChartFileName,
    BitmapFormat.JPG, 100);
  }
 catch (Exception bt)
  {
    bt.printStackTrace();
  }

System.out.println ("The Chart has been saved");

returnCodes[0] = 0;
```

```
    returnCodes[1] = 0;
    returnCodes[2] = 0;

    return returnCodes;

   }  // End of method

  //================================================
  // Load the previously saved trained network1 and tests it by
  // processing the Test record
  //================================================

  static public void loadAndTestNetwork()
   {
     System.out.println("Testing the network1s results");

     List<Double> xData = new ArrayList<Double>();
     List<Double> yData1 = new ArrayList<Double>();
     List<Double> yData2 = new ArrayList<Double>();

     double targetToPredictFunctValueDiff = 0;
     double maxGlobalResultDiff = 0.00;
     double averGlobalResultDiff = 0.00;
     double sumGlobalResultDiff = 0.00;
     double maxGlobalIndex = 0;

     double normInputDayFromRecord1 = 0.00;
     double normTargetFunctValue1 = 0.00;
     double normPredictFunctValue1 = 0.00;
     double denormInputDayFromRecord1 = 0.00;
     double denormTargetFunctValue1 = 0.00;
     double denormAverPredictFunctValue1 = 0.00;

     double normInputDayFromRecord2 = 0.00;
     double normTargetFunctValue2 = 0.00;
     double normPredictFunctValue2 = 0.00;
     double denormInputDayFromRecord2 = 0.00;
     double denormTargetFunctValue2 = 0.00;
     double denormAverPredictFunctValue2 = 0.00;
```

```
 double normInputDayFromTestRecord = 0.00;
 double denormInputDayFromTestRecord = 0.00;
 double denormAverPredictFunctValue = 0.00;

 double denormTargetFunctValueFromTestRecord = 0.00;

 String tempLine;
 String[] tempWorkFields;
 double dayKeyFromTestRecord = 0.00;
 double targetFunctValueFromTestRecord = 0.00;
 double r1 = 0.00;
 double r2 = 0.00;
 BufferedReader br4;

 BasicNetwork network1;
 BasicNetwork network2;
 int k1 = 0;
 int k3 = 0;

try
 {
    // Process testing records
    maxGlobalResultDiff = 0.00;
    averGlobalResultDiff = 0.00;
    sumGlobalResultDiff = 0.00;

    for (k1 = 0; k1 < numberOfTestBatchesToProcess; k1++)
     {
        // if(k1 == 9998)
        //   k1 = k1;

        // Read the corresponding test micro-batch file.
        br4 = new BufferedReader(new FileReader(strTestingFileNames[k1]));
        tempLine = br4.readLine();

        // Skip the label record
        tempLine = br4.readLine();
```

```
// Break the line using comma as separator
tempWorkFields = tempLine.split(cvsSplitBy);

dayKeyFromTestRecord = Double.parseDouble(tempWorkFields[0]);
targetFunctValueFromTestRecord = Double.parseDouble
(tempWorkFields[1]);

// De-normalize the dayKeyFromTestRecord
denormInputDayFromTestRecord = ((minXPointDl - maxXPointDh)*
dayKeyFromTestRecord - Nh*minXPointDl + maxXPointDh*Nl)/
(Nl - Nh);

  // De-normalize the targetFunctValueFromTestRecord
denormTargetFunctValueFromTestRecord = ((minTargetValueDl -
maxTargetValueDh)*targetFunctValueFromTestRecord - Nh*
minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

// Load the corresponding training micro-batch dataset in memory
MLDataSet trainingSet1 = loadCSV2Memory(strTrainingFile
Names[k1],intInputNeuronNumber,intOutputNeuronNumber,true,
CSVFormat.ENGLISH,false);

//MLDataSet testingSet =
//   loadCSV2Memory(strTestingFileNames[k1],
     intInputNeuronNumber,
//     intOutputNeuronNumber,true,CSVFormat.ENGLISH,false);

network1 =
 (BasicNetwork)EncogDirectoryPersistence.
   loadObject(new File(strSaveTrainNetworkFileNames[k1]));

 // Get the results after the network1 optimization
 int iMax = 0;
 int i = - 1; // Index of the array to get results

 for (MLDataPair pair1:  trainingSet1)
  {
```

```
        i++;
        iMax = i+1;

        MLData inputData1 = pair1.getInput();
        MLData actualData1 = pair1.getIdeal();
        MLData predictData1 = network1.compute(inputData1);

        // These values are Normalized
        normInputDayFromRecord1 = inputData1.getData(0);
        normTargetFunctValue1 = actualData1.getData(0);
        normPredictFunctValue1 = predictData1.getData(0);

        // De-normalize the obtained values
        denormInputDayFromRecord1 = ((minXPointDl - maxXPointDh)*
        normInputDayFromRecord1 - Nh*minXPointDl +
        maxXPointDh*Nl)/(Nl - Nh);

        denormTargetFunctValue1 = ((minTargetValueDl - maxTarget
        ValueDh)*normTargetFunctValue1 - Nh*minTargetValueDl +
        maxTargetValueDh*Nl)/(Nl - Nh);

        denormAverPredictFunctValue1 =((minTargetValueDl -
        maxTargetValueDh)*normPredictFunctValue1 - Nh*minTarget
        ValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

    }  // End for pair1

    // ---------------------------------------------------------
    // Now calculate everything again for the SaveNetwork (which
    // key is greater than dayKeyFromTestRecord value)in memory
    // ---------------------------------------------------------

MLDataSet trainingSet2 = loadCSV2Memory(strTrainingFileNames
[k1+1],intInputNeuronNumber,
   intOutputNeuronNumber,true,CSVFormat.ENGLISH,false);

network2 = (BasicNetwork)EncogDirectoryPersistence.loadObject
           (new File(strSaveTrainNetworkFileNames[k1+1]));

    // Get the results after the network1 optimization
```

```
iMax = 0;
i = - 1; // Index of the array to get results

for (MLDataPair pair2:  trainingSet2)
{
   i++;
   iMax = i+1;

   MLData inputData2 = pair2.getInput();
   MLData actualData2 = pair2.getIdeal();
   MLData predictData2 = network2.compute(inputData2);

   // These values are Normalized
   normInputDayFromRecord2 = inputData2.getData(0);
   normTargetFunctValue2 = actualData2.getData(0);
   normPredictFunctValue2 = predictData2.getData(0);

   // De-normalize the obtained values
   denormInputDayFromRecord2 = ((minXPointDl - maxXPointDh)*
   normInputDayFromRecord2 - Nh*minXPointDl + maxX
   PointDh*Nl)/(Nl - Nh);

   denormTargetFunctValue2 = ((minTargetValueDl - maxTarget
   ValueDh)*normTargetFunctValue2 - Nh*minTargetValueDl +
   maxTargetValueDh*Nl)/(Nl - Nh);

   denormAverPredictFunctValue2 =((minTargetValueDl -
   maxTargetValueDh)*normPredictFunctValue2 - Nh*minTarget
   ValueDl + maxTargetValueDh*Nl)/(Nl - Nh);
}  // End for pair1 loop

// Get the average of the denormAverPredictFunctValue1 and
   denormAverPredictFunctValue2
denormAverPredictFunctValue = (denormAverPredictFunctValue1 +
denormAverPredictFunctValue2)/2;

targetToPredictFunctValueDiff = (Math.abs(denormTargetFunct
ValueFromTestRecord - enormAverPredictFunctValue)/
ddenormTargetFunctValueFromTestRecord)*100;
```

```
            System.out.println("Record Number = " + k1 + "  DayNumber = " +
            denormInputDayFromTestRecord + "  denormTargetFunctValue
            FromTestRecord = " + denormTargetFunctValueFromTestRecord +
            "  denormAverPredictFunctValue = " + denormAverPredictFunct
            Value + "  valurDiff = " + targetToPredictFunctValueDiff);

            if (targetToPredictFunctValueDiff > maxGlobalResultDiff)
             {
               maxGlobalIndex = iMax;
               maxGlobalResultDiff =targetToPredictFunctValueDiff;
             }

            sumGlobalResultDiff = sumGlobalResultDiff + targetToPredict
            FunctValueDiff;
            // Populate chart elements

            xData.add(denormInputDayFromTestRecord);
            yData1.add(denormTargetFunctValueFromTestRecord);
            yData2.add(denormAverPredictFunctValue);

     }   // End of loop using k1

   // Print the max and average results

   System.out.println(" ");

   averGlobalResultDiff = sumGlobalResultDiff/numberOfTestBatches
   ToProcess;

   System.out.println("maxGlobalResultDiff = " + maxGlobalResultDiff +
   "  i = " + maxGlobalIndex);
   System.out.println("averGlobalResultDiff = " + averGlobalResultDiff);

 }     // End of TRY
catch (IOException e1)
 {
       e1.printStackTrace();
 }
```

```
    // All testing batch files have been processed
    XYSeries series1 = Chart.addSeries("Actual data", xData, yData1);
    XYSeries series2 = Chart.addSeries("Predict data", xData, yData2);

    series1.setLineColor(XChartSeriesColors.BLACK);
    series2.setLineColor(XChartSeriesColors.YELLOW);

    series1.setMarkerColor(Color.BLACK);
    series2.setMarkerColor(Color.WHITE);
    series1.setLineStyle(SeriesLines.SOLID);
    series2.setLineStyle(SeriesLines.DASH_DASH);

    // Save the chart image
    try
     {
        BitmapEncoder.saveBitmapWithDPI(Chart, strTrainChartFileName,
          BitmapFormat.JPG, 100);
     }
    catch (Exception bt)
      {
        bt.printStackTrace();
      }

     System.out.println ("The Chart has been saved");
     System.out.println("End of testing for mini-batches training");

  } // End of the method

} // End of the  Encog class
```

Listing 9-5 shows the ending fragment of the testing results after execution.

Listing 9-5. Ending Fragment of the Testing Processing Results

```
DayNumber = 3.98411  TargetValue = 1.17624  AverPredicedValue = 1.18028
DiffPerc = 0.34348
DayNumber = 3.98442  TargetValue = 1.18522  AverPredicedValue = 1.18927
DiffPerc = 0.34158
DayNumber = 3.98472  TargetValue = 1.19421  AverPredicedValue = 1.19827
DiffPerc = 0.33959
```

```
DayNumber = 3.98502  TargetValue = 1.20323  AverPredicedValue = 1.20729
DiffPerc = 0.33751
DayNumber = 3.98532  TargetValue = 1.21225  AverPredicedValue = 1.21631
DiffPerc = 0.33534
DayNumber = 3.98562  TargetValue = 1.22128  AverPredicedValue = 1.22535
DiffPerc = 0.33307
DayNumber = 3.98592  TargetValue = 1.23032  AverPredicedValue = 1.23439
DiffPerc = 0.33072
DayNumber = 3.98622  TargetValue = 1.23936  AverPredicedValue = 1.24343
DiffPerc = 0.32828
DayNumber = 3.98652  TargetValue = 1.24841  AverPredicedValue = 1.25247
DiffPerc = 0.32575
DayNumber = 3.98682  TargetValue = 1.25744  AverPredicedValue = 1.26151
DiffPerc = 0.32313
DayNumber = 3.98712  TargetValue = 1.26647  AverPredicedValue = 1.27053
DiffPerc = 0.32043
DayNumber = 3.98742  TargetValue = 1.27549  AverPredicedValue = 1.27954
DiffPerc = 0.31764
DayNumber = 3.98772  TargetValue = 1.28449  AverPredicedValue = 1.28854
DiffPerc = 0.31477
DayNumber = 3.98802  TargetValue = 1.29348  AverPredicedValue = 1.29751
DiffPerc = 0.31181
DayNumber = 3.98832  TargetValue = 1.30244  AverPredicedValue = 1.30646
DiffPerc = 0.30876
DayNumber = 3.98862  TargetValue = 1.31138  AverPredicedValue = 1.31538
DiffPerc = 0.30563
DayNumber = 3.98892  TargetValue = 1.32028  AverPredicedValue = 1.32428
DiffPerc = 0.30242
DayNumber = 3.98922  TargetValue = 1.32916  AverPredicedValue = 1.33313
DiffPerc = 0.29913
DayNumber = 3.98952  TargetValue = 1.33799  AverPredicedValue = 1.34195
DiffPerc = 0.29576
DayNumber = 3.98982  TargetValue = 1.34679  AverPredicedValue = 1.35072
DiffPerc = 0.29230
```

```
DayNumber = 3.99012  TargetValue = 1.35554  AverPredicedValue = 1.35945
DiffPerc = 0.28876
DayNumber = 3.99042  TargetValue = 1.36423  AverPredicedValue = 1.36812
DiffPerc = 0.28515
DayNumber = 3.99072  TargetValue = 1.37288  AverPredicedValue = 1.37674
DiffPerc = 0.28144
DayNumber = 3.99102  TargetValue = 1.38146  AverPredicedValue = 1.38530
DiffPerc = 0.27768
DayNumber = 3.99132  TargetValue = 1.38999  AverPredicedValue = 1.39379
DiffPerc = 0.27383
DayNumber = 3.99162  TargetValue = 1.39844  AverPredicedValue = 1.40222
DiffPerc = 0.26990
DayNumber = 3.99192  TargetValue = 1.40683  AverPredicedValue = 1.41057
DiffPerc = 0.26590
DayNumber = 3.99222  TargetValue = 1.41515  AverPredicedValue = 1.41885
DiffPerc = 0.26183
DayNumber = 3.99252  TargetValue = 1.42338  AverPredicedValue = 1.42705
DiffPerc = 0.25768
DayNumber = 3.99282  TargetValue = 1.43153  AverPredicedValue = 1.43516
DiffPerc = 0.25346
DayNumber = 3.99312  TargetValue = 1.43960  AverPredicedValue = 1.44318
DiffPerc = 0.24918
DayNumber = 3.99342  TargetValue = 1.44757  AverPredicedValue = 1.45111
DiffPerc = 0.24482
DayNumber = 3.99372  TargetValue = 1.45544  AverPredicedValue = 1.45894
DiffPerc = 0.24040
DayNumber = 3.99402  TargetValue = 1.46322  AverPredicedValue = 1.46667
DiffPerc = 0.23591
DayNumber = 3.99432  TargetValue = 1.47089  AverPredicedValue = 1.47429
DiffPerc = 0.23134
DayNumber = 3.99462  TargetValue = 1.47845  AverPredicedValue = 1.48180
DiffPerc = 0.22672
DayNumber = 3.99492  TargetValue = 1.48590  AverPredicedValue = 1.48920
DiffPerc = 0.22204
```

```
DayNumber = 3.99522  TargetValue = 1.49323  AverPredicedValue = 1.49648
DiffPerc = 0.21729
DayNumber = 3.99552  TargetValue = 1.50044  AverPredicedValue = 1.50363
DiffPerc = 0.21247
DayNumber = 3.99582  TargetValue = 1.50753  AverPredicedValue = 1.51066
DiffPerc = 0.20759
DayNumber = 3.99612  TargetValue = 1.51448  AverPredicedValue = 1.51755
DiffPerc = 0.20260
DayNumber = 3.99642  TargetValue = 1.52130  AverPredicedValue = 1.52431
DiffPerc = 0.19770
DayNumber = 3.99672  TargetValue = 1.52799  AverPredicedValue = 1.53093
DiffPerc = 0.19260
DayNumber = 3.99702  TargetValue = 1.53453  AverPredicedValue = 1.53740
DiffPerc = 0.18751
DayNumber = 3.99732  TargetValue = 1.54092  AverPredicedValue = 1.54373
DiffPerc = 0.18236
DayNumber = 3.99762  TargetValue = 1.54717  AverPredicedValue = 1.54991
DiffPerc = 0.17715
DayNumber = 3.99792  TargetValue = 1.55326  AverPredicedValue = 1.55593
DiffPerc = 0.17188
DayNumber = 3.99822  TargetValue = 1.55920  AverPredicedValue = 1.56179
DiffPerc = 0.16657
DayNumber = 3.99852  TargetValue = 1.56496  AverPredicedValue = 1.56749
DiffPerc = 0.16120
DayNumber = 3.99882  TargetValue = 1.57057  AverPredicedValue = 1.57302
DiffPerc = 0.15580
DayNumber = 3.99912  TargetValue = 1.57601  AverPredicedValue = 1.57838
DiffPerc = 0.15034
DayNumber = 3.99942  TargetValue = 1.58127  AverPredicedValue = 1.58356
DiffPerc = 0.14484

maxGlobalResultDiff = 0.3620154382225759
averGlobalResultDiff = 0.07501532301280595
```

The testing processing results (using the micro-batch method) are as follows:

- The maximum error is less than 0.36 percent.
- The average error is less than 0.075 percent.

Figure 9-5 shows the chart of the testing processing results (using the micro-batch method). Again, both charts are very close and practically overlap (actual values are black, and predicted values are white).

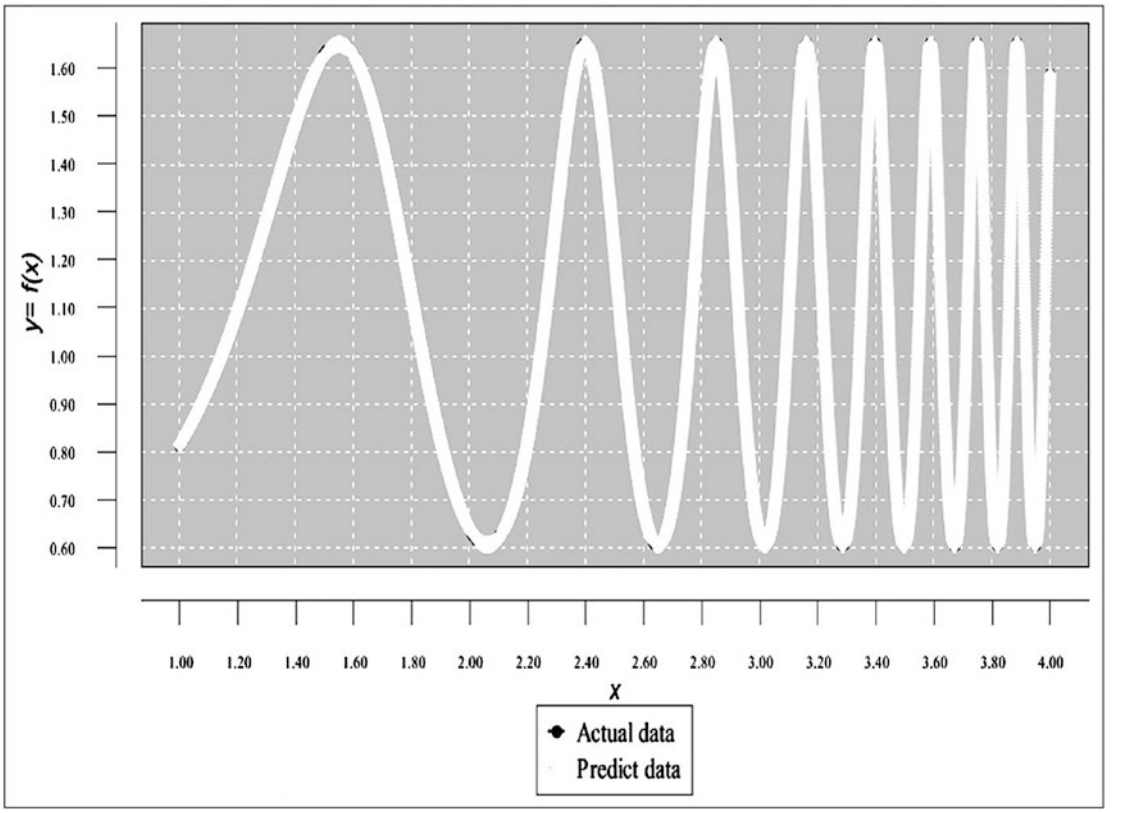


Figure 9-5. *Chart of the testing processing results (using the micro-batch method)*

Example 5b: Approximation of Spiral-Like Functions

This section continues the discussion of approximating functions with difficult topologies. Specifically, it discusses a group of functions that are spiral-like. These functions have a common property; at some points, they have multiple function values for a single x point.

Functions in this group are notoriously difficult to approximate using neural networks. You will attempt to approximate the function shown in Figure 9-6 first using the conventional method (which does not work well) and then using the micro-batch method.

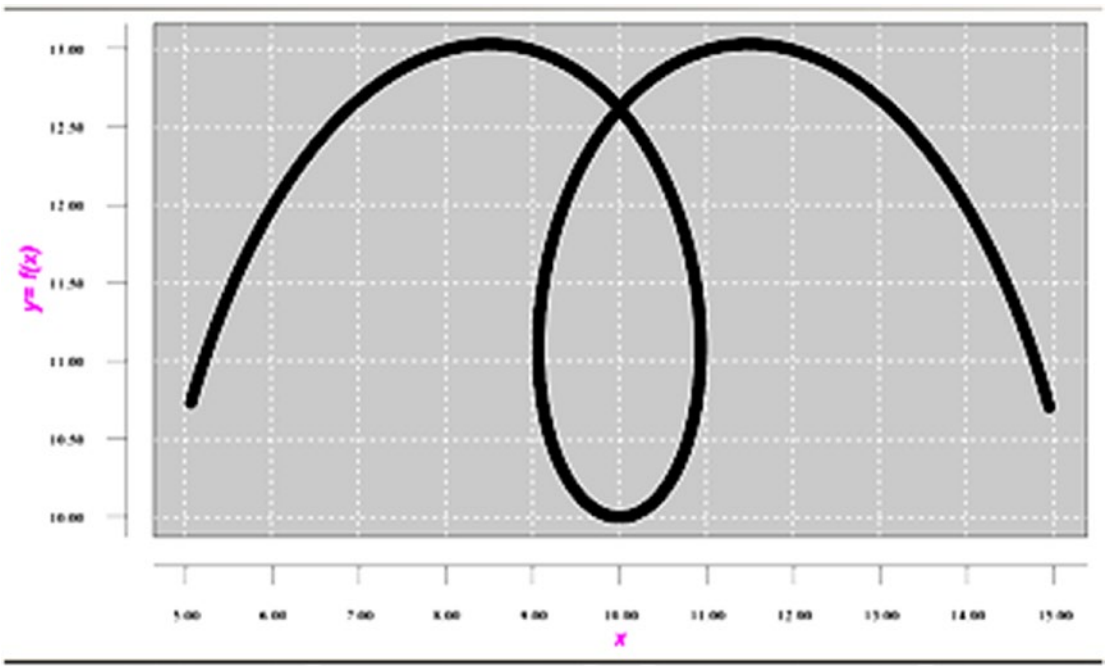


Figure 9-6. *Function with multiple values for some xPoints*

The function is described by these two equations, where t is an angle:

$$x(t) = 10 + 0.5^{*} t^{*} \text{Cos}(0.3^{*} t)$$
$$y(t) = 10 + 0.5^{*} t^{*} \text{Sin}(0.3^{*} t)$$

Figure 9-6 shows the chart produced by plotting the values of x and y. Again, we are pretending that the function formula is unknown and that the function is given to you by its values at 1,000 points. As usual, you will first try to approximate this function in a conventional way. Table 9-3 shows the fragment of the training data set.

Table 9-3. *Fragment of the Training Data Set*

x	y
14.94996248	10.70560004
14.93574853	10.73381636
14.92137454	10.76188757
14.90684173	10.78981277
14.89215135	10.81759106
14.87730464	10.84522155
14.86230283	10.87270339
14.84714718	10.90003569
14.83183894	10.92721761
14.81637936	10.9542483
14.80076973	10.98112693
14.78501129	11.00785266
14.76910532	11.03442469
14.7530531	11.06084221
14.73685592	11.08710443
14.72051504	11.11321054
14.70403178	11.13915979
14.68740741	11.1649514
14.67064324	11.19058461
14.65374057	11.21605868
14.6367007	11.24137288
14.61952494	11.26652647
14.60221462	11.29151873
14.58477103	11.31634896

(*continued*)

Table 9-3. (*continued*)

x	y
14.56719551	11.34101647
14.54948938	11.36552056
14.53165396	11.38986056
14.51369059	11.41403579
14.49560061	11.43804562
14.47738534	11.46188937
14.45904613	11.48556643

The testing data set is prepared for the points not used in training. Table 9-4 shows the fragment of the testing data set.

Table 9-4. *Fragment of the Testing Data Set*

x	y
14.9499625	10.70560004
14.9357557	10.73380229
14.921389	10.76185957
14.9068637	10.78977099
14.8921809	10.81753565
14.8773419	10.84515266
14.8623481	10.87262116
14.8472005	10.89994029
14.8319005	10.92710918
14.8164493	10.954127
14.8008481	10.98099291
14.7850984	11.00770609

(*continued*)

Table 9-4. (*continued*)

x	y
14.7692012	11.03426572
14.7531579	11.060671
14.7369698	11.08692113
14.7206381	11.11301533
14.7041642	11.13895282
14.6875493	11.16473284
14.6707947	11.19035462
14.6539018	11.21581743
14.6368718	11.24112053
14.619706	11.26626319
14.6024058	11.29124469
14.5849724	11.31606434
14.5674072	11.34072143
14.5497115	11.36521527
14.5318866	11.38954519
14.5139339	11.41371053
14.4958547	11.43771062
14.4776503	11.46154483
14.4593221	11.48521251

Both the training and testing data sets have been normalized before processing.

Network Architecture for Example 5b

Figure 9-7 shows the network architecture.

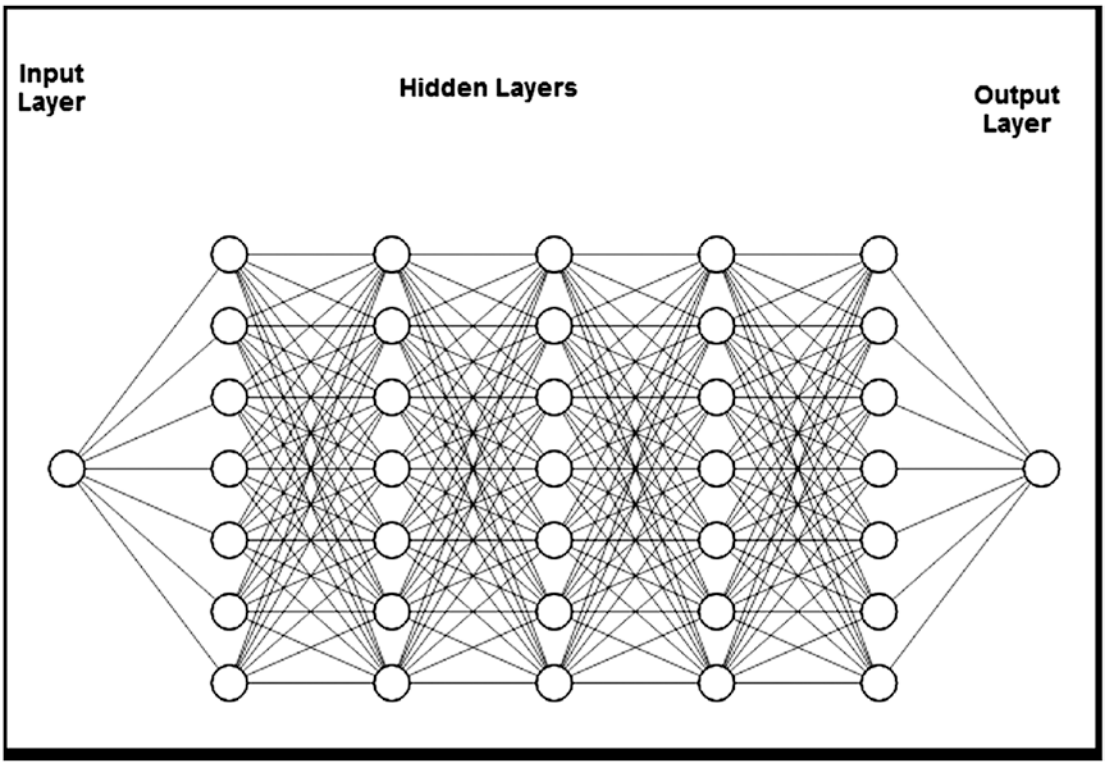


Figure 9-7. *Network architecture*

Program Code for Example 5b

Listing 9-6 shows the program code of the approximation using the conventional process.

Listing 9-6. Program Code of the Conventional Approximation Process

```
// ================================================
// Approximation spiral-like function using the conventional process.
// The input file is normalized.
// ================================================
package sample8;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
```

```
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.*;
import java.util.Properties;
import java.time.YearMonth;
import java.awt.Color;
import java.awt.Font;
import java.io.BufferedReader;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.LocalDate;
import java.time.Month;
import java.time.ZoneId;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.List;
import java.util.Locale;
import java.util.Properties;

import org.encog.Encog;
import org.encog.engine.network.activation.ActivationTANH;
import org.encog.engine.network.activation.ActivationReLU;
import org.encog.ml.data.MLData;
import org.encog.ml.data.MLDataPair;
import org.encog.ml.data.MLDataSet;
import org.encog.ml.data.buffer.MemoryDataLoader;
import org.encog.ml.data.buffer.codec.CSVDataCODEC;
import org.encog.ml.data.buffer.codec.DataSetCODEC;
import org.encog.neural.networks.BasicNetwork;
import org.encog.neural.networks.layers.BasicLayer;
```

```
import org.encog.neural.networks.training.propagation.resilient.
ResilientPropagation;
import org.encog.persist.EncogDirectoryPersistence;
import org.encog.util.csv.CSVFormat;

import org.knowm.xchart.SwingWrapper;
import org.knowm.xchart.XYChart;
import org.knowm.xchart.XYChartBuilder;
import org.knowm.xchart.XYSeries;
import org.knowm.xchart.demo.charts.ExampleChart;
import org.knowm.xchart.style.Styler.LegendPosition;
import org.knowm.xchart.style.colors.ChartColor;
import org.knowm.xchart.style.colors.XChartSeriesColors;
import org.knowm.xchart.style.lines.SeriesLines;
import org.knowm.xchart.style.markers.SeriesMarkers;
import org.knowm.xchart.BitmapEncoder;
import org.knowm.xchart.BitmapEncoder.BitmapFormat;
import org.knowm.xchart.QuickChart;
import org.knowm.xchart.SwingWrapper;

public class Sample8 implements ExampleChart<XYChart>
{
    // Interval to normalize
   static double Nh =  1;
   static double Nl = -1;

   // First column
   static double minXPointDl = 1.00;
   static double maxXPointDh = 20.00;

   // Second column - target data
   static double minTargetValueDl = 1.00;
   static double maxTargetValueDh = 20.00;
   static double doublePointNumber = 0.00;
   static int intPointNumber = 0;
   static InputStream input = null;
   static double[] arrPrices = new double[2500];
   static double normInputXPointValue = 0.00;
```

```
  static double normPredictXPointValue = 0.00;
  static double normTargetXPointValue = 0.00;
  static double normDifferencePerc = 0.00;
  static double returnCode = 0.00;
  static double denormInputXPointValue = 0.00;
  static double denormPredictXPointValue = 0.00;
  static double denormTargetXPointValue = 0.00;
  static double valueDifference = 0.00;
  static int numberOfInputNeurons;
  static int numberOfOutputNeurons;
  static int intNumberOfRecordsInTestFile;
  static String trainFileName;
  static String priceFileName;
  static String testFileName;
  static String chartTrainFileName;
  static String chartTestFileName;
  static String networkFileName;
  static int workingMode;
  static String cvsSplitBy = ",";

  static int numberOfInputRecords = 0;

  static List<Double> xData = new ArrayList<Double>();
  static List<Double> yData1 = new ArrayList<Double>();
  static List<Double> yData2 = new ArrayList<Double>();

  static XYChart Chart;

 @Override
 public XYChart getChart()
  {
   // Create Chart

   Chart = new  XYChartBuilder().width(900).height(500).title(getClass().
             getSimpleName()).xAxisTitle("x").yAxisTitle("y= f(x)").build();
```

```
// Customize Chart
Chart = new  XYChartBuilder().width(900).height(500).title(getClass().
          getSimpleName()).xAxisTitle("x").yAxisTitle("y= f(x)").build();

// Customize Chart
Chart.getStyler().setPlotBackgroundColor(ChartColor.
getAWTColor(ChartColor.GREY));
Chart.getStyler().setPlotGridLinesColor(new Color(255, 255, 255));

//Chart.getStyler().setPlotBackgroundColor(ChartColor.getAWTColor
(ChartColor.WHITE));
//Chart.getStyler().setPlotGridLinesColor(new Color(0, 0, 0));
Chart.getStyler().setChartBackgroundColor(Color.WHITE);
//Chart.getStyler().setLegendBackgroundColor(Color.PINK);
Chart.getStyler().setLegendBackgroundColor(Color.WHITE);
//Chart.getStyler().setChartFontColor(Color.MAGENTA);
Chart.getStyler().setChartFontColor(Color.BLACK);
Chart.getStyler().setChartTitleBoxBackgroundColor(new Color(0, 222, 0));
Chart.getStyler().setChartTitleBoxVisible(true);
Chart.getStyler().setChartTitleBoxBorderColor(Color.BLACK);
Chart.getStyler().setPlotGridLinesVisible(true);
Chart.getStyler().setAxisTickPadding(20);
Chart.getStyler().setAxisTickMarkLength(15);
Chart.getStyler().setPlotMargin(20);
Chart.getStyler().setChartTitleVisible(false);
Chart.getStyler().setChartTitleFont(new Font(Font.MONOSPACED,
Font.BOLD, 24));
Chart.getStyler().setLegendFont(new Font(Font.SERIF, Font.PLAIN, 18));
//Chart.getStyler().setLegendPosition(LegendPosition.InsideSE);
Chart.getStyler().setLegendPosition(LegendPosition.OutsideS);
Chart.getStyler().setLegendSeriesLineLength(12);
Chart.getStyler().setAxisTitleFont(new Font(Font.SANS_SERIF, Font.
ITALIC, 18));
Chart.getStyler().setAxisTickLabelsFont(new Font(Font.SERIF, Font.
PLAIN, 11));
```

```
    Chart.getStyler().setDatePattern("yyyy-MM");
    Chart.getStyler().setDecimalPattern("#0.00");

    try
       {
         // Configuration

         // Set the mode the program should run

         workingMode = 1; // Training mode

         if(workingMode == 1)
           {
             // Training mode
             numberOfInputRecords = 1001;
             trainFileName =
               "/My_Neural_Network_Book/Book_Examples/Sample8_Calculate_
               Train_Norm.csv";
             chartTrainFileName =
               "C:/My_Neural_Network_Book/Book_Examples/
                 Sample8_Chart_ComplexFormula_Spiral_Train_Results.csv";
           }
         else
           {
              // Testing mode
              numberOfInputRecords = 1003;
              intNumberOfRecordsInTestFile = 3;
              testFileName = "C:/Book_Examples/Sample2_Norm.csv";
              chartTestFileName = "XYLine_Test_Results_Chart";
          }

         // Common part of config data
         networkFileName = "C:/My_Neural_Network_Book/Book_Examples/
               Sample8_Saved_Network_File.csv";

         numberOfInputNeurons = 1;
         numberOfOutputNeurons = 1;

         // Check the working mode to run
```

```
      if(workingMode == 1)
        {
          // Training mode
          File file1 = new File(chartTrainFileName);
          File file2 = new File(networkFileName);

          if(file1.exists())
            file1.delete();

          if(file2.exists())
            file2.delete();

          returnCode = 0;    // Clear the error Code

          do
           {
             returnCode = trainValidateSaveNetwork();
           }   while (returnCode > 0);
       }
      else
       {
          // Test mode
          loadAndTestNetwork();
       }
    }
   catch (Throwable t)
     {
       t.printStackTrace();
       System.exit(1);
     }
    finally
     {
       Encog.getInstance().shutdown();
     }
 Encog.getInstance().shutdown();

 return Chart;

}  // End of the method
```

```
// =======================================================
// Load CSV to memory.
// @return The loaded dataset.
// =======================================================
public static MLDataSet loadCSV2Memory(String filename, int input, int
ideal, boolean headers, CSVFormat format, boolean significance)
  {
     DataSetCODEC codec = new CSVDataCODEC(new File(filename), format,
     headers, input, ideal, significance);
     MemoryDataLoader load = new MemoryDataLoader(codec);
     MLDataSet dataset = load.external2Memory();
     return dataset;
  }

// =======================================================
//  The main method.
//  @param Command line arguments. No arguments are used.
// =======================================================
public static void main(String[] args)
 {
   ExampleChart<XYChart> exampleChart = new Sample8();
   XYChart Chart = exampleChart.getChart();
   new SwingWrapper<XYChart>(Chart).displayChart();
 } // End of the main method

//=====================================================================
// This method trains, Validates, and saves the trained network file
//=====================================================================
static public double trainValidateSaveNetwork()
 {
   // Load the training CSV file in memory
   MLDataSet trainingSet =
     loadCSV2Memory(trainFileName,numberOfInputNeurons,numberOf
     OutputNeurons,true,CSVFormat.ENGLISH,false);
```

```
// create a neural network
BasicNetwork network = new BasicNetwork();

// Input layer
network.addLayer(new BasicLayer(null,true,1));

// Hidden layer
network.addLayer(new BasicLayer(new ActivationTANH(),true,10));
network.addLayer(new BasicLayer(new ActivationTANH(),true,10));
network.addLayer(new BasicLayer(new ActivationTANH(),true,10));
network.addLayer(new BasicLayer(new ActivationTANH(),true,10));
network.addLayer(new BasicLayer(new ActivationTANH(),true,10));
network.addLayer(new BasicLayer(new ActivationTANH(),true,10));
network.addLayer(new BasicLayer(new ActivationTANH(),true,10));

// Output layer
//network.addLayer(new BasicLayer(new ActivationLOG(),false,1));
network.addLayer(new BasicLayer(new ActivationTANH(),false,1));

network.getStructure().finalizeStructure();
network.reset();

// train the neural network
final ResilientPropagation train = new ResilientPropagation(network,
trainingSet);

int epoch = 1;

do
 {
  train.iteration();
  System.out.println("Epoch #" + epoch + " Error:" + train.getError());

   epoch++;

   if (epoch >= 11000 && network.calculateError(trainingSet) > 0.2251)
       {
        returnCode = 1;

        System.out.println("Try again");
```

```
        return returnCode;
       }
 } while(train.getError() > 0.225);

// Save the network file
EncogDirectoryPersistence.saveObject(new File(networkFileName),
network);

System.out.println("Neural Network Results:");

double sumNormDifferencePerc = 0.00;
double averNormDifferencePerc = 0.00;
double maxNormDifferencePerc = 0.00;

int m = 0;
double xPointer = 0.00;

for(MLDataPair pair: trainingSet)
  {
      m++;
      xPointer++;

      //if(m == 0)
      // continue;

       final MLData output = network.compute(pair.getInput());

       MLData inputData = pair.getInput();
       MLData actualData = pair.getIdeal();
       MLData predictData = network.compute(inputData);

       // Calculate and print the results
       normInputXPointValue = inputData.getData(0);
       normTargetXPointValue = actualData.getData(0);
       normPredictXPointValue = predictData.getData(0);

       denormInputXPointValue = ((minXPointDl - maxXPointDh)*normInpu
       tXPointValue - Nh*minXPointDl + maxXPointDh *Nl)/(Nl - Nh);
```

```
        denormTargetXPointValue =((minTargetValueDl - maxTargetValueDh)*
        normTargetXPointValue - Nh*minTargetValueDl + maxTarget
        ValueDh*Nl)/(Nl - Nh);
        denormPredictXPointValue =((minTargetValueDl - maxTarget
        ValueDh)* normPredictXPointValue - Nh*minTargetValueDl + max
        TargetValueDh*Nl)/(Nl - Nh);

        valueDifference = Math.abs(((denormTargetXPointValue -
        denormPredictXPointValue)/denormTargetXPointValue)*100.00);

        System.out.println ("Day = " + denormInputXPointValue +
        "  denormTargetXPointValue = " + denormTargetXPointValue +
        "  denormPredictXPointValue = " + denormPredictXPointValue +
        "  valueDifference = " + valueDifference);
        //System.out.println("intPointNumber = " + intPointNumber);

        sumNormDifferencePerc = sumNormDifferencePerc + valueDifference;

        if (valueDifference > maxNormDifferencePerc)
          maxNormDifferencePerc = valueDifference;

        xData.add(denormInputXPointValue);
        yData1.add(denormTargetXPointValue);
        yData2.add(denormPredictXPointValue);

    }   // End for pair loop

    XYSeries series1 = Chart.addSeries("Actual data", xData, yData1);
    XYSeries series2 = Chart.addSeries("Predict data", xData, yData2);

    series1.setLineColor(XChartSeriesColors.BLACK);
    series2.setLineColor(XChartSeriesColors.LIGHT_GREY);

    series1.setMarkerColor(Color.BLACK);
    series2.setMarkerColor(Color.WHITE);
    series1.setLineStyle(SeriesLines.NONE);
    series2.setLineStyle(SeriesLines.SOLID);
```

```
        try
         {
           //Save the chart image
           BitmapEncoder.saveBitmapWithDPI(Chart, chartTrainFileName,
             BitmapFormat.JPG, 100);
           System.out.println ("Train Chart file has been saved") ;
         }
       catch (IOException ex)
        {
         ex.printStackTrace();
         System.exit(3);
        }

        // Finally, save this trained network
        EncogDirectoryPersistence.saveObject(new File(networkFileName),
        network);
        System.out.println ("Train Network has been saved") ;

        averNormDifferencePerc  = sumNormDifferencePerc/numberOfInput
        Records;


        System.out.println(" ");
        System.out.println("maxNormDifferencePerc = " + maxNormDifference
        Perc + averNormDifferencePerc = " + averNormDifferencePerc);"

        returnCode = 0.00;
        return returnCode;

    }   // End of the method

   //================================================
   // This method load and test the trained network
   //================================================
   static public void loadAndTestNetwork()
    {
     System.out.println("Testing the networks results");
```

```
List<Double> xData = new ArrayList<Double>();
List<Double> yData1 = new ArrayList<Double>();
List<Double> yData2 = new ArrayList<Double>();

double targetToPredictPercent = 0;
double maxGlobalResultDiff = 0.00;
double averGlobalResultDiff = 0.00;
double sumGlobalResultDiff = 0.00;
double maxGlobalIndex = 0;
double normInputXPointValueFromRecord = 0.00;
double normTargetXPointValueFromRecord = 0.00;
double normPredictXPointValueFromRecord = 0.00;

BasicNetwork network;

maxGlobalResultDiff = 0.00;
averGlobalResultDiff = 0.00;
sumGlobalResultDiff = 0.00;

// Load the test dataset into memory
MLDataSet testingSet =
loadCSV2Memory(testFileName,numberOfInputNeurons,numberOfOutput
Neurons,true, CSVFormat.ENGLISH,false);

// Load the saved trained network
network =
  (BasicNetwork)EncogDirectoryPersistence.loadObject(new File(network
  FileName));

int i = - 1; // Index of the current record
double xPoint = -0.00;

for (MLDataPair pair:  testingSet)
 {
      i++;
      xPoint = xPoint + 2.00;
```

```
            MLData inputData = pair.getInput();
            MLData actualData = pair.getIdeal();
            MLData predictData = network.compute(inputData);

            // These values are Normalized as the whole input is
            normInputXPointValueFromRecord = inputData.getData(0);
            normTargetXPointValueFromRecord = actualData.getData(0);
            normPredictXPointValueFromRecord = predictData.getData(0);

            denormInputXPointValue = ((minXPointDl - maxXPointDh)*
            normInputXPointValueFromRecord - Nh*minXPointDl + maxX
            PointDh*Nl)/(Nl - Nh);
            denormTargetXPointValue = ((minTargetValueDl - maxTargetValueDh)*
            normTargetXPointValueFromRecord - Nh*minTargetValueDl +
            maxTargetValueDh*Nl)/(Nl - Nh);
            denormPredictXPointValue =((minTargetValueDl - maxTargetValueDh)*
            normPredictXPointValueFromRecord - Nh*minTargetValueDl +
            maxTargetValueDh*Nl)/(Nl - Nh);

            targetToPredictPercent = Math.abs((denormTargetXPointValue -
            denormPredictXPointValue)/denormTargetXPointValue*100);

            System.out.println("xPoint = " + xPoint +  "  denormTargetX
            PointValue = " + denormTargetXPointValue + "  denormPredictX
            PointValue = " + denormPredictXPointValue + " targetToPredict
            Percent = " + targetToPredictPercent);

            if (targetToPredictPercent > maxGlobalResultDiff)
               maxGlobalResultDiff = targetToPredictPercent;

            sumGlobalResultDiff = sumGlobalResultDiff + targetToPredict
            Percent;

            // Populate chart elements
            xData.add(xPoint);
            yData1.add(denormTargetXPointValue);
            yData2.add(denormPredictXPointValue);

        }  // End for pair loop
```

```
        // Print the max and average results
        System.out.println(" ");
        averGlobalResultDiff = sumGlobalResultDiff/numberOfInputRecords;

        System.out.println("maxGlobalResultDiff = " + maxGlobalResultDiff +
        "  i = " + maxGlobalIndex);
        System.out.println("averGlobalResultDiff = " + averGlobalResultDiff);

        // All testing batch files have been processed
        XYSeries series1 = Chart.addSeries("Actual", xData, yData1);
        XYSeries series2 = Chart.addSeries("Predicted", xData, yData2);

        series1.setLineColor(XChartSeriesColors.BLUE);
        series2.setMarkerColor(Color.ORANGE);
        series1.setLineStyle(SeriesLines.SOLID);
        series2.setLineStyle(SeriesLines.SOLID);

        // Save the chart image
        try
         {
           BitmapEncoder.saveBitmapWithDPI(Chart, chartTestFileName ,
           BitmapFormat.JPG, 100);
         }
        catch (Exception bt)
         {
           bt.printStackTrace();
         }

        System.out.println ("The Chart has been saved");
        System.out.println("End of testing for test records");

    } // End of the method

} // End of the class
```

The function was approximated using the conventional network processing. Listing 9-7 shows the end fragment of the conventional processing results.

Listing 9-7. The End Fragment of the Conventional Training Results

```
Day = 5.57799  TargetValue = 11.53242  PredictedValue = 1.15068
DiffPerc = 90.02216
Day = 5.55941  TargetValue = 11.50907  PredictedValue = 1.15073
DiffPerc = 90.00153
Day = 5.54095  TargetValue = 11.48556  PredictedValue = 1.15077
DiffPerc = 89.98067
Day = 5.52261  TargetValue = 11.46188  PredictedValue = 1.15082
DiffPerc = 89.95958
Day = 5.50439  TargetValue = 11.43804  PredictedValue = 1.15086
DiffPerc = 89.93824
Day = 5.48630  TargetValue = 11.41403  PredictedValue = 1.15091
DiffPerc = 89.91667
Day = 5.46834  TargetValue = 11.38986  PredictedValue = 1.15096
DiffPerc = 89.89485
Day = 5.45051  TargetValue = 11.36552  PredictedValue = 1.15100
DiffPerc = 89.87280
Day = 5.43280  TargetValue = 11.34101  PredictedValue = 1.15105
DiffPerc = 89.85049
Day = 5.41522  TargetValue = 11.31634  PredictedValue = 1.15110
DiffPerc = 89.82794
Day = 5.39778  TargetValue = 11.29151  PredictedValue = 1.15115
DiffPerc = 89.80515
Day = 5.38047  TargetValue = 11.26652  PredictedValue = 1.15120
DiffPerc = 89.78210
Day = 5.36329  TargetValue = 11.24137  PredictedValue = 1.15125
DiffPerc = 89.75880
Day = 5.34625  TargetValue = 11.21605  PredictedValue = 1.15130
DiffPerc = 89.73525
Day = 5.32935  TargetValue = 11.19058  PredictedValue = 1.15134
DiffPerc = 89.71144
Day = 5.31259  TargetValue = 11.16495  PredictedValue = 1.15139
DiffPerc = 89.68737
Day = 5.29596  TargetValue = 11.13915  PredictedValue = 1.15144
DiffPerc = 89.66305
```

```
Day = 5.27948  TargetValue = 11.11321  PredictedValue = 1.15149
DiffPerc = 89.63846
Day = 5.26314  TargetValue = 11.08710  PredictedValue = 1.15154
DiffPerc = 89.61361
Day = 5.24694  TargetValue = 11.06084  PredictedValue = 1.15159
DiffPerc = 89.58850
Day = 5.23089  TargetValue = 11.03442  PredictedValue = 1.15165
DiffPerc = 89.56311
Day = 5.21498  TargetValue = 11.00785  PredictedValue = 1.15170
DiffPerc = 89.53746
Day = 5.19923  TargetValue = 10.98112  PredictedValue = 1.15175
DiffPerc = 89.51153
Day = 5.18362  TargetValue = 10.95424  PredictedValue = 1.15180
DiffPerc = 89.48534
Day = 5.16816  TargetValue = 10.92721  PredictedValue = 1.15185
DiffPerc = 89.45886
Day = 5.15285  TargetValue = 10.90003  PredictedValue = 1.15190
DiffPerc = 89.43211
Day = 5.13769  TargetValue = 10.87270  PredictedValue = 1.15195
DiffPerc = 89.40508
Day = 5.12269  TargetValue = 10.84522  PredictedValue = 1.15200
DiffPerc = 89.37776
Day = 5.10784  TargetValue = 10.81759  PredictedValue = 1.15205
DiffPerc = 89.35016
Day = 5.09315  TargetValue = 10.78981  PredictedValue = 1.15210
DiffPerc = 89.32228
Day = 5.07862  TargetValue = 10.76188  PredictedValue = 1.15215
DiffPerc = 89.29410

maxErrorPerc = 91.1677948809837
averErrorPerc = 90.04645291133258
```

With the conventional process, the approximation results are as follows:

- The maximum error percent exceeds 91.16 percent.
- The average error percent exceeds 90,0611 percent.

Figure 9-8 shows the chart of the training approximation results using the conventional network processing.

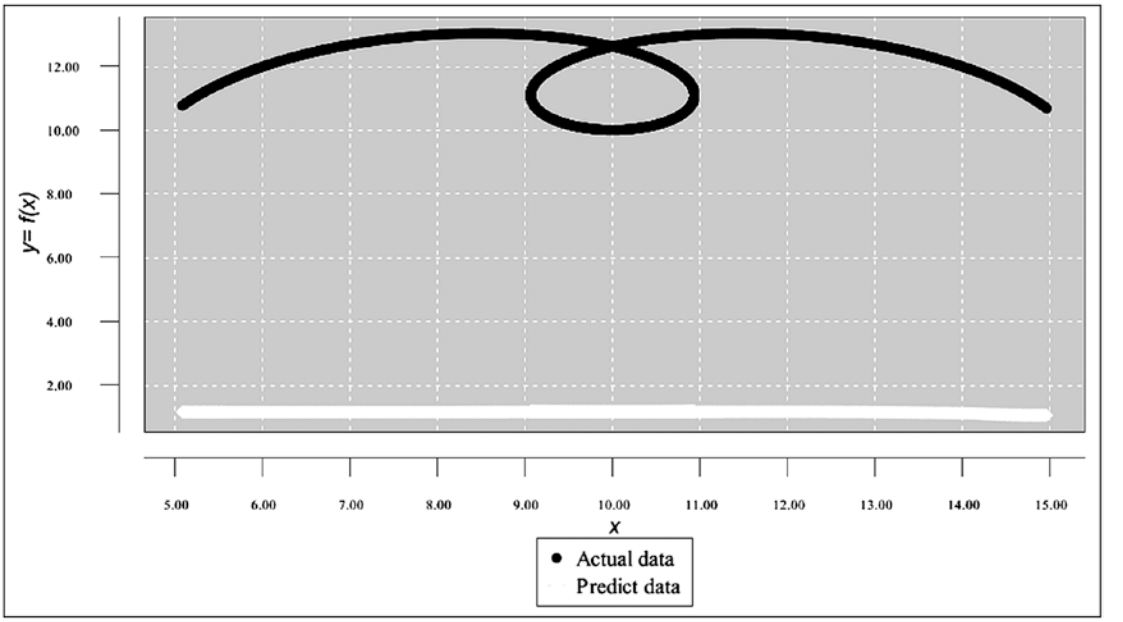


Figure 9-8. *Chart of the training approximation results using the conventional network processing*

Obviously, such an approximation is completely useless.

Approximation of the Same Function Using the Micro-Batch Method

Now, let's approximate this function using the micro-batch method. Again, the normalized training data set is broken into a set of training micro-batch files, and it is now the input to the training process. Listing 9-8 shows the program code for the training method using the micro-batch process.

Listing 9-8. Program Code for the Training Method Using the Micro-Batch Process

```
// ================================================================
// Approximation the spiral-like function using the micro-batch method.
// The input is the normalized set of micro-batch files  (each micro-batch
// includes a single day record).
// Each record consists of:
// - normDayValue
// - normTargetValue
//
// The number of inputLayer neurons is 1
// The number of outputLayer neurons is 1
// Each network is saved on disk and a map is created to link each saved
   trained
// network with the corresponding training micro-batch file.
// ================================================================

package sample8_microbatches;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.*;
import java.util.Properties;
import java.time.YearMonth;
import java.awt.Color;
import java.awt.Font;
import java.io.BufferedReader;
import java.time.Month;
import java.time.ZoneId;
```

```
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;
import java.util.Locale;
import java.util.Properties;
import org.encog.Encog;
import org.encog.engine.network.activation.ActivationTANH;
import org.encog.engine.network.activation.ActivationReLU;
import org.encog.ml.data.MLData;
import org.encog.ml.data.MLDataPair;
import org.encog.ml.data.MLDataSet;
import org.encog.ml.data.buffer.MemoryDataLoader;
import org.encog.ml.data.buffer.codec.CSVDataCODEC;
import org.encog.ml.data.buffer.codec.DataSetCODEC;
import org.encog.neural.networks.BasicNetwork;
import org.encog.neural.networks.layers.BasicLayer;
import org.encog.neural.networks.training.propagation.resilient.
  ResilientPropagation;
import org.encog.persist.EncogDirectoryPersistence;
import org.encog.util.csv.CSVFormat;

import org.knowm.xchart.SwingWrapper;
import org.knowm.xchart.XYChart;
import org.knowm.xchart.XYChartBuilder;
import org.knowm.xchart.XYSeries;
import org.knowm.xchart.demo.charts.ExampleChart;
import org.knowm.xchart.style.Styler.LegendPosition;
import org.knowm.xchart.style.colors.ChartColor;
import org.knowm.xchart.style.colors.XChartSeriesColors;
import org.knowm.xchart.style.lines.SeriesLines;
import org.knowm.xchart.style.markers.SeriesMarkers;
import org.knowm.xchart.BitmapEncoder;
import org.knowm.xchart.BitmapEncoder.BitmapFormat;
import org.knowm.xchart.QuickChart;
import org.knowm.xchart.SwingWrapper;
```

```
public class Sample8_Microbatches implements ExampleChart<XYChart>
{
   // Normalization parameters

   // Normalizing interval
   static double Nh =  1;
   static double Nl = -1;

   // inputFunctValueDiffPerc
  static double inputDayDh = 20.00;
  static double inputDayDl = 1.00;

  // targetFunctValueDiffPerc
  static double targetFunctValueDiffPercDh = 20.00;
  static double targetFunctValueDiffPercDl = 1.00;

   static String cvsSplitBy = ",";
   static Properties prop = null;

   static String strWorkingMode;
   static String strNumberOfBatchesToProcess;
   static String strTrainFileNameBase;
   static String strTestFileNameBase;
   static String strSaveTrainNetworkFileBase;
   static String strSaveTestNetworkFileBase;
   static String strValidateFileName;
   static String strTrainChartFileName;
   static String strTestChartFileName;
   static String strFunctValueTrainFile;
   static String strFunctValueTestFile;
   static int intDayNumber;
   static double doubleDayNumber;
   static int intWorkingMode;
   static int numberOfTrainBatchesToProcess;
   static int numberOfTestBatchesToProcess;
   static int intNumberOfRecordsInTrainFile;
   static int intNumberOfRecordsInTestFile;
   static int intNumberOfRowsInBatches;
```

```
  static int intInputNeuronNumber;
  static int intOutputNeuronNumber;
  static String strOutputFileName;
  static String strSaveNetworkFileName;
  static String strDaysTrainFileName;
  static XYChart Chart;
  static String iString;
  static double inputFunctValueFromFile;
  static double targetToPredictFunctValueDiff;
  static int[] returnCodes  = new int[3];

  static List<Double> xData = new ArrayList<Double>();
  static List<Double> yData1 = new ArrayList<Double>();
  static List<Double> yData2 = new ArrayList<Double>();

  static double[] DaysyearDayTraining = new double[1200];
  static String[] strTrainingFileNames = new String[1200];
  static String[] strTestingFileNames = new String[1200];
  static String[] strSaveTrainNetworkFileNames = new String[1200];
  static double[] linkToSaveNetworkDayKeys = new double[1200];
  static double[] linkToSaveNetworkTargetFunctValueKeys = new double[1200];
  static double[] arrTrainFunctValues = new double[1200];
  static double[] arrTestFunctValues = new double[1200];

@Override
public XYChart getChart()
 {
   // Create Chart

  Chart = new XYChartBuilder().width(900).height(500).title(getClass().
    getSimpleName()).xAxisTitle("day").yAxisTitle("y=f(day)").build();

  // Customize Chart
  Chart.getStyler().setPlotBackgroundColor(ChartColor.getAWTColor
  (ChartColor.GREY));
  Chart.getStyler().setPlotGridLinesColor(new Color(255, 255, 255));
  Chart.getStyler().setChartBackgroundColor(Color.WHITE);
  Chart.getStyler().setLegendBackgroundColor(Color.PINK);
```

```
Chart.getStyler().setChartFontColor(Color.MAGENTA);
Chart.getStyler().setChartTitleBoxBackgroundColor(new Color(0, 222, 0));
Chart.getStyler().setChartTitleBoxVisible(true);
Chart.getStyler().setChartTitleBoxBorderColor(Color.BLACK);
Chart.getStyler().setPlotGridLinesVisible(true);
Chart.getStyler().setAxisTickPadding(20);
Chart.getStyler().setAxisTickMarkLength(15);
Chart.getStyler().setPlotMargin(20);
Chart.getStyler().setChartTitleVisible(false);
Chart.getStyler().setChartTitleFont(new Font(Font.MONOSPACED,
Font.BOLD, 24));
Chart.getStyler().setLegendFont(new Font(Font.SERIF, Font.PLAIN, 18));
// Chart.getStyler().setLegendPosition(LegendPosition.InsideSE);
Chart.getStyler().setLegendPosition(LegendPosition.OutsideE);
Chart.getStyler().setLegendSeriesLineLength(12);
Chart.getStyler().setAxisTitleFont(new Font(Font.SANS_SERIF,
Font.ITALIC, 18));
Chart.getStyler().setAxisTickLabelsFont(new Font(Font.SERIF,
Font.PLAIN, 11));
//Chart.getStyler().setDayPattern("yyyy-MM");
Chart.getStyler().setDecimalPattern("#0.00");

// Config data

// Set the mode the program should run
intWorkingMode = 1;  // Training mode

if(intWorkingMode == 1)
  {
    numberOfTrainBatchesToProcess = 1000;
    numberOfTestBatchesToProcess = 999;
    intNumberOfRowsInBatches = 1;
    intInputNeuronNumber = 1;
    intOutputNeuronNumber = 1;
    strTrainFileNameBase = "C:/My_Neural_Network_Book/Book_Examples/
    Work_Files/Sample8_Microbatch_Train_";
```

```
strTestFileNameBase = "C:/My_Neural_Network_Book/Book_Examples/
Work_Files/Sample8_Microbatch_Test_";
strSaveTrainNetworkFileBase = "C:/My_Neural_Network_Book/Book_
Examples/Work_Files/Sample8_Save_Network_Batch_";
strTrainChartFileName = "C:/Book_Examples/Sample8_Chart_Train_File_
Microbatch.jpg";
strTestChartFileName = "C:/Book_Examples/Sample8_Chart_Test_File_
Microbatch.jpg";

 // Generate training batches file names and the corresponding
 // SaveNetwork file names

 intDayNumber = -1;  // Day number for the chart

 for (int i = 0; i < numberOfTrainBatchesToProcess; i++)
   {
      intDayNumber++;

      iString = Integer.toString(intDayNumber);

      if (intDayNumber >= 10 & intDayNumber < 100  )
        {
          strOutputFileName = strTrainFileNameBase + "0" +
          iString + ".csv";
          strSaveNetworkFileName = strSaveTrainNetwork
          FileBase + "0" + iString + ".csv";
        }
      else
 {
    if (intDayNumber < 10)
      {
         strOutputFileName = strTrainFileNameBase + "00" +
         iString + ".csv";
         strSaveNetworkFileName = strSaveTrainNetworkFileBase +
         "00" + iString + ".csv";
      }
```

```
    else
      {
       strOutputFileName = strTrainFileNameBase + iString + ".csv";

       strSaveNetworkFileName = strSaveTrainNetworkFileBase +
         iString + ".csv";
      }
   }

    strTrainingFileNames[intDayNumber] = strOutputFileName;
    strSaveTrainNetworkFileNames[intDayNumber] =
    strSaveNetworkFileName;

}  // End the FOR loop

// Build the array linkToSaveNetworkFunctValueDiffKeys
String tempLine;
double tempNormFunctValueDiff = 0.00;
double tempNormFunctValueDiffPerc = 0.00;
double tempNormTargetFunctValueDiffPerc = 0.00;

String[] tempWorkFields;

try
 {
    intDayNumber = -1;  // Day number for the chart

    for (int m = 0; m < numberOfTrainBatchesToProcess; m++)
      {
         intDayNumber++;

         BufferedReader br3 = new BufferedReader(new
           FileReader(strTrainingFileNames[intDayNumber]));

         tempLine = br3.readLine();

         // Skip the label record and zero batch record
         tempLine = br3.readLine();

         // Break the line using comma as separator
         tempWorkFields = tempLine.split(cvsSplitBy);
```

```
            tempNormFunctValueDiffPerc = Double.parseDouble
            (tempWorkFields[0]);
            tempNormTargetFunctValueDiffPerc = Double.parseDouble
            (tempWorkFields[1]);

            linkToSaveNetworkDayKeys[intDayNumber] = tempNormFunct
            ValueDiffPerc;

            linkToSaveNetworkTargetFunctValueKeys[intDayNumber] =
            tempNormTargetFunctValueDiffPerc;

        }  // End the FOR loop

    }
else
    {
        // Testing mode. Generate testing batch file names
        intDayNumber = -1;

        for (int i = 0; i < numberOfTestBatchesToProcess; i++)
          {
            intDayNumber++;
            iString = Integer.toString(intDayNumber);

             // Construct the testing batch names
             if (intDayNumber >= 10 & intDayNumber < 100  )
               {
                 strOutputFileName = strTestFileNameBase + "0" +
                 iString + ".csv";
               }
             else
               {
                 if (intDayNumber < 10)
                  {
                   strOutputFileName = strTestFileNameBase + "00" +
                   iString + ".csv";
                   }
```

```
                    else
                      {
                        strOutputFileName = strTestFileNameBase +
                        iString + ".csv";
                      }
                }

          strTestingFileNames[intDayNumber] = strOutputFileName;

        }  // End the FOR loop

       }   // End of IF

    }     // End for try
   catch (IOException io1)
    {
      io1.printStackTrace();
      System.exit(1);
    }
 }
else
 {
       // Train mode

       // Load, train, and test Function Values file in memory
       loadTrainFunctValueFileInMemory();

       int paramErrorCode;
       int paramBatchNumber;
       int paramR;
       int paramDayNumber;
       int paramS;

       File file1 = new File(strTrainChartFileName);

       if(file1.exists())
         file1.delete();
```

```
            returnCodes[0] = 0;    // Clear the error Code
            returnCodes[1] = 0;    // Set the initial batch Number to 0;
            returnCodes[2] = 0;    // Day number;

           do
            {
              paramErrorCode = returnCodes[0];
              paramBatchNumber = returnCodes[1];
              paramDayNumber = returnCodes[2];

              returnCodes =
                trainBatches(paramErrorCode,paramBatchNumber,paramDayNumber);
            } while (returnCodes[0] > 0);

          }   // End the train logic
         else
          {
            // Testing mode

            File file2 = new File(strTestChartFileName);

            if(file2.exists())
              file2.delete();

            loadAndTestNetwork();

           // End the test logic
         }

         Encog.getInstance().shutdown();
         //System.exit(0);
         return Chart;

     }  // End of method

    // =======================================================
    // Load CSV to memory.
    // @return The loaded dataset.
    // =======================================================
```

```
public static MLDataSet loadCSV2Memory(String filename, int input, int
ideal, boolean headers, CSVFormat format, boolean significance)
  {
     DataSetCODEC codec = new CSVDataCODEC(new File(filename), format,
     headers, input, ideal, significance);
     MemoryDataLoader load = new MemoryDataLoader(codec);
     MLDataSet dataset = load.external2Memory();
     return dataset;
  }

// =======================================================
//  The main method.
//  @param Command line arguments. No arguments are used.
// =======================================================
public static void main(String[] args)
 {

   ExampleChart<XYChart> exampleChart = new Sample8_Microbatches();
   XYChart Chart = exampleChart.getChart();
   new SwingWrapper<XYChart>(Chart).displayChart();
 } // End of the main method

//=======================================
// This method trains batches as individual network1s
// saving them in separate trained datasets
//=======================================
static public int[] trainBatches(int paramErrorCode,
                                 int paramBatchNumber,int
                                 paramDayNumber)
  {
    int rBatchNumber;
    double targetToPredictFunctValueDiff = 0;
    double maxGlobalResultDiff = 0.00;
    double averGlobalResultDiff = 0.00;
    double sumGlobalResultDiff = 0.00;
    double normInputFunctValueFromRecord = 0.00;
    double normTargetFunctValue1 = 0.00;
```

```
        double normPredictFunctValue1 = 0.00;
        double denormInputDayFromRecord;
        double denormInputFunctValueFromRecord = 0.00;
        double denormTargetFunctValue = 0.00;
        double denormPredictFunctValue1 = 0.00;

        BasicNetwork network1 = new BasicNetwork();

        // Input layer
        network1.addLayer(new BasicLayer(null,true,intInputNeuronNumber));

      // Hidden layer.
      network1.addLayer(new BasicLayer(new ActivationTANH(),true,7));
      network1.addLayer(new BasicLayer(new ActivationTANH(),true,7));
      network1.addLayer(new BasicLayer(new ActivationTANH(),true,7));
      network1.addLayer(new BasicLayer(new ActivationTANH(),true,7));
      network1.addLayer(new BasicLayer(new ActivationTANH(),true,7));

       // Output layer
       network1.addLayer(new BasicLayer(new ActivationTANH(),false,
       intOutputNeuronNumber));

       network1.getStructure().finalizeStructure();
       network1.reset();

       maxGlobalResultDiff = 0.00;
       averGlobalResultDiff = 0.00;
       sumGlobalResultDiff = 0.00;

       // Loop over batches
       intDayNumber = paramDayNumber;  // Day number for the chart

       for (rBatchNumber = paramBatchNumber; rBatchNumber < numberOfTrain
       BatchesToProcess;
           rBatchNumber++)
        {
          intDayNumber++;

          //if(intDayNumber == 502)
          // rBatchNumber = rBatchNumber;
```

```
// Load the training file in memory
MLDataSet trainingSet = loadCSV2Memory(strTrainingFileNames
[rBatchNumber],intInputNeuronNumber,intOutputNeuronNumber,
true,CSVFormat.ENGLISH,false);

// train the neural network1
ResilientPropagation train = new ResilientPropagation(network1,
trainingSet);

int epoch = 1;

do
  {
    train.iteration();
    epoch++;

    for (MLDataPair pair11:  trainingSet)
     {
       MLData inputData1 = pair11.getInput();
       MLData actualData1 = pair11.getIdeal();
       MLData predictData1 = network1.compute(inputData1);

       // These values are Normalized as the whole input is
       normInputFunctValueFromRecord = inputData1.getData(0);

       normTargetFunctValue1 = actualData1.getData(0);
       normPredictFunctValue1 = predictData1.getData(0);

       denormInputFunctValueFromRecord =((inputDayDl -
       inputDayDh)*normInputFunctValueFromRecord - Nh*inputDayDl +
       inputDayDh*Nl)/(Nl - Nh);
       denormTargetFunctValue = ((targetFunctValueDiffPercDl -
       targetFunctValueDiffPercDh)*normTargetFunctValue1 -
       Nh*targetFunctValueDiffPercDl + targetFunctValue
       DiffPercDh*Nl)/(Nl - Nh);
       denormPredictFunctValue1 =((targetFunctValueDiffPercDl -
       targetFunctValueDiffPercDh)*normPredictFunctValue1 -
       Nh*targetFunctValueDiffPercDl + targetFunctValueDiff
       PercDh*Nl)/(Nl - Nh);
```

```
            //inputFunctValueFromFile = arrTrainFunctValues[rBatchNumber];

            targetToPredictFunctValueDiff = (Math.abs(denormTarget
            FunctValue - enormPredictFunctValue1)/ddenormTarget
            FunctValue)*100;
          }

        if (epoch >= 1000 && Math.abs(targetToPredictFunctValueDiff) >
        0.0000091)
         {
           returnCodes[0] = 1;
           returnCodes[1] = rBatchNumber;
           returnCodes[2] = intDayNumber-1;

           return returnCodes;
          }

         //System.out.println("intDayNumber = " + intDayNumber);

       } while(Math.abs(targetToPredictFunctValueDiff) > 0.000009);

// This batch is optimized

// Save the network1 for the current batch
EncogDirectoryPersistence.saveObject(newFile(strSaveTrainNetwork
FileNames[rBatchNumber]),network1);

// Get the results after the network1 optimization
int i = - 1;

for (MLDataPair pair1:  trainingSet)
 {
  i++;

  MLData inputData1 = pair1.getInput();
  MLData actualData1 = pair1.getIdeal();
  MLData predictData1 = network1.compute(inputData1);
```

```
        // These values are Normalized as the whole input is
        normInputFunctValueFromRecord = inputData1.getData(0);
        normTargetFunctValue1 = actualData1.getData(0);
        normPredictFunctValue1 = predictData1.getData(0);

        // De-normalize the obtained values
        denormInputFunctValueFromRecord =((inputDayDl - inputDayDh)*
        normInputFunctValueFromRecord - Nh*inputDayDl + inputDayDh*Nl)/
        (Nl - Nh);

        denormTargetFunctValue = ((targetFunctValueDiffPercDl - targetFunct
        ValueDiffPercDh)*normTargetFunctValue1 - DiffPercDl + target
        FunctValueDiffPercDh*Nl)/(Nl - Nh);

        denormPredictFunctValue1 =((targetFunctValueDiffPercDl - targetFunct
        ValueDiffPercDh)*normPredictFunctValue1 - Nh*targetFunctValue
        DiffPercDl + targetFunctValueDiffPercDh*Nl)/(Nl - Nh);

        //inputFunctValueFromFile = arrTrainFunctValues[rBatchNumber];

        targetToPredictFunctValueDiff = (Math.abs(denormTargetFunctValue -
        denormPredictFunctValue1)/denormTargetFunctValue)*100;

        System.out.println("intDayNumber = " + intDayNumber +  "  target
        FunctionValue = " + denormTargetFunctValue + "  predictFunction
        Value = " + denormPredictFunctValue1 + "  valurDiff = " +
        targetToPredictFunctValueDiff);

        if (targetToPredictFunctValueDiff > maxGlobalResultDiff)
          maxGlobalResultDiff =targetToPredictFunctValueDiff;

        sumGlobalResultDiff = sumGlobalResultDiff +targetToPredict
        FunctValueDiff;

        // Populate chart elements
        xData.add(denormInputFunctValueFromRecord);
        yData1.add(denormTargetFunctValue);
        yData2.add(denormPredictFunctValue1);

      }  // End for FunctValue pair1 loop

    }  // End of the loop over batches
```

```
    sumGlobalResultDiff = sumGlobalResultDiff +targetToPredict
    FunctValueDiff;
    averGlobalResultDiff = sumGlobalResultDiff/numberOfTrainBatches
    ToProcess;

    // Print the max and average results

    System.out.println(" ");
    System.out.println(" ");
    System.out.println("maxGlobalResultDiff = " + maxGlobalResultDiff);
    System.out.println("averGlobalResultDiff = " + averGlobalResultDiff);

    XYSeries series1 = Chart.addSeries("Actual", xData, yData1);
    XYSeries series2 = Chart.addSeries("Forecasted", xData, yData2);

    series1.setMarkerColor(Color.BLACK);
    series2.setMarkerColor(Color.WHITE);
    series1.setLineStyle(SeriesLines.SOLID);
    series2.setLineStyle(SeriesLines.SOLID);

    // Save the chart image
    try
      {
        BitmapEncoder.saveBitmapWithDPI(Chart, strTrainChartFileName,
        BitmapFormat.JPG, 100);
      }
     catch (Exception bt)
      {
        bt.printStackTrace();
      }

    System.out.println ("The Chart has been saved");

    returnCodes[0] = 0;
    returnCodes[1] = 0;
    returnCodes[2] = 0;

    return returnCodes;

   }  // End of method
```

```
//=======================================================
// Load the previously saved trained network1 and tests it by
// processing the Test record
//=======================================================

static public void loadAndTestNetwork()
 {
   System.out.println("Testing the network1s results");

   List<Double> xData = new ArrayList<Double>();
   List<Double> yData1 = new ArrayList<Double>();
   List<Double> yData2 = new ArrayList<Double>();

   double targetToPredictFunctValueDiff = 0;
   double maxGlobalResultDiff = 0.00;
   double averGlobalResultDiff = 0.00;
   double sumGlobalResultDiff = 0.00;
   double maxGlobalIndex = 0;

   double normInputDayFromRecord1 = 0.00;
   double normTargetFunctValue1 = 0.00;
   double normPredictFunctValue1 = 0.00;
   double denormInputDayFromRecord = 0.00;
   double denormTargetFunctValue = 0.00;
   double denormPredictFunctValue = 0.00;
   double normInputDayFromRecord2 = 0.00;
   double normTargetFunctValue2 = 0.00;
   double normPredictFunctValue2 = 0.00;
   double denormInputDayFromRecord2 = 0.00;
   double denormTargetFunctValue2 = 0.00;
   double denormPredictFunctValue2 = 0.00;
   double normInputDayFromTestRecord = 0.00;
   double denormInputDayFromTestRecord = 0.00;
   double denormTargetFunctValueFromTestRecord = 0.00;

   String tempLine;
   String[] tempWorkFields;
   double dayKeyFromTestRecord = 0.00;
```

```
double targetFunctValueFromTestRecord = 0.00;
double r1 = 0.00;
double r2 = 0.00;
BufferedReader br4;

BasicNetwork network1;
BasicNetwork network2;
int k1 = 0;
int k3 = 0;

try
 {
    // Process testing records
    maxGlobalResultDiff = 0.00;
    averGlobalResultDiff = 0.00;
    sumGlobalResultDiff = 0.00;

    for (k1 = 0; k1 < numberOfTestBatchesToProcess; k1++)
     {
        if(k1 == 100)
          k1 = k1;

        // Read the corresponding test micro-batch file.
        br4 = new BufferedReader(new FileReader(strTestingFileNames[k1]));
        tempLine = br4.readLine();

        // Skip the label record
        tempLine = br4.readLine();

        // Break the line using comma as separator
        tempWorkFields = tempLine.split(cvsSplitBy);

        dayKeyFromTestRecord = Double.parseDouble(tempWorkFields[0]);
        targetFunctValueFromTestRecord = Double.parseDouble
        (tempWorkFields[1]);

        // De-normalize the dayKeyFromTestRecord
        denormInputDayFromTestRecord =
        ((inputDayDl - inputDayDh)*dayKeyFromTestRecord -
        Nh*inputDayDl + inputDayDh*Nl)/(Nl - Nh);
```

```
  // De-normalize the targetFunctValueFromTestRecord
denormTargetFunctValueFromTestRecord =
((targetFunctValueDiffPercDl - targetFunctValueDiffPercDh)*
targetFunctValueFromTestRecord - Nh*targetFunctValueDiffPercDl +
targetFunctValueDiffPercDh*Nl)/(Nl - Nh);

// Load the corresponding training micro-batch dataset in memory
MLDataSet trainingSet1 = loadCSV2Memory(strTrainingFileNames
[k1],intInputNeuronNumber,intOutputNeuronNumber,
true,CSVFormat.ENGLISH,false);

//MLDataSet testingSet =
//   loadCSV2Memory(strTestingFileNames[k1],intInputNeuronNumber,
//   intOutputNeuronNumber,true,CSVFormat.ENGLISH,false);

network1 =
 (BasicNetwork)EncogDirectoryPersistence.
   loadObject(new File(strSaveTrainNetworkFileNames[k1]));

 // Get the results after the network1 optimization
 int iMax = 0;
 int i = - 1; // Index of the array to get results

 for (MLDataPair pair1:  trainingSet1)
  {
    i++;
    iMax = i+1;

    MLData inputData1 = pair1.getInput();
    MLData actualData1 = pair1.getIdeal();
    MLData predictData1 = network1.compute(inputData1);

    // These values are Normalized as the whole input is
    normInputDayFromRecord1 = inputData1.getData(0);
    normTargetFunctValue1 = actualData1.getData(0);
    normPredictFunctValue1 = predictData1.getData(0);
```

```
        denormInputDayFromRecord = ((inputDayDl - inputDayDh)*
        normInputDayFromRecord1 - Nh*inputDayDl +
        inputDayDh*Nl)/(Nl - Nh);

        denormTargetFunctValue = ((targetFunctValueDiffPercDl -
        targetFunctValueDiffPercDh)*normTargetFunctValue1 - Nh*
        targetFunctValueDiffPercDl + targetFunctValue
        DiffPercDh*Nl)/(Nl - Nh);

        denormPredictFunctValue =((targetFunctValueDiffPercDl -
        targetFunctValueDiffPercDh)*normPredictFunctValue1 - Nh*
        targetFunctValueDiffPercDl + targetFunctValue
        DiffPercDh*Nl)/(Nl - Nh);

        targetToPredictFunctValueDiff = (Math.abs(denormTarget
        FunctValue - denormPredictFunctValue)/denormTarget
        FunctValue)*100;

        System.out.println("Record Number = " + k1 + "  DayNumber =
        " + denormInputDayFromTestRecord +
        "  denormTargetFunctValueFromTestRecord = " + denormTarget
        FunctValueFromTestRecord + "  denormPredictFunctValue = " +
        denormPredictFunctValue + "  valurDiff = " + target
        ToPredictFunctValueDiff);

        if (targetToPredictFunctValueDiff > maxGlobalResultDiff)
         {
           maxGlobalIndex = iMax;
           maxGlobalResultDiff =targetToPredictFunctValueDiff;
         }

        sumGlobalResultDiff = sumGlobalResultDiff +
          targetToPredictFunctValueDiff;

        // Populate chart elements

        xData.add(denormInputDayFromTestRecord);
        yData1.add(denormTargetFunctValueFromTestRecord);
        yData2.add(denormPredictFunctValue);

    }  // End for pair2 loop
```

```
     }   // End of loop using k1

    // Print the max and average results

    System.out.println(" ");

    averGlobalResultDiff = sumGlobalResultDiff/numberOfTestBatches
    ToProcess;

    System.out.println("maxGlobalResultDiff = " + maxGlobalResultDiff +
    "  i = " + maxGlobalIndex);
    System.out.println("averGlobalResultDiff = " + averGlobalResultDiff);

  }     // End of TRY
    catch (FileNotFoundException nf)
  {
        nf.printStackTrace();
  }
 catch (IOException e1)
  {
        e1.printStackTrace();
  }

// All testing batch files have been processed
XYSeries series1 = Chart.addSeries("Actual", xData, yData1);
XYSeries series2 = Chart.addSeries("Forecasted", xData, yData2);

series1.setLineColor(XChartSeriesColors.BLACK);
series2.setLineColor(XChartSeriesColors.LIGHT_GREY);

series1.setMarkerColor(Color.BLACK);
series2.setMarkerColor(Color.WHITE);
series1.setLineStyle(SeriesLines.SOLID);
series2.setLineStyle(SeriesLines.SOLID);
```

```
    // Save the chart image
    try
     {
         BitmapEncoder.saveBitmapWithDPI(Chart, strTrainChartFileName,
         BitmapFormat.JPG, 100);
     }
    catch (Exception bt)
     {
        bt.printStackTrace();
     }

    System.out.println ("The Chart has been saved");
    System.out.println("End of testing for mini-batches training");

   } // End of the method

 } // End of the  Encog class
```

Listing 9-9 shows the ending fragment of the training processing results (using the macro-batch method) after execution.

Listing 9-9. Ending Fragment of the Training Processing Results (Using the Macro-Batch Method)

```
DayNumber = 947  targetFunctionValue = 12.02166
predictFunctionValue = 12.02166 valurDiff = 5.44438E-6
DayNumber = 948  targetFunctionValue = 12.00232
predictFunctionValue = 12.00232 valurDiff = 3.83830E-6
DayNumber = 949  targetFunctionValue = 11.98281
predictFunctionValue = 11.98281 valurDiff = 2.08931E-6
DayNumber = 950  targetFunctionValue = 11.96312
predictFunctionValue = 11.96312 valurDiff = 6.72376E-6
DayNumber = 951  targetFunctionValue = 11.94325
predictFunctionValue = 11.94325 valurDiff = 4.16461E-7
DayNumber = 952  targetFunctionValue = 11.92320
predictFunctionValue = 11.92320 valurDiff = 1.27943E-6
DayNumber = 953  targetFunctionValue = 11.90298
predictFunctionValue = 11.90298 valurDiff = 8.38334E-6
```

```
DayNumber = 954  targetFunctionValue = 11.88258
predictFunctionValue = 11.88258 valurDiff = 5.87549E-6
DayNumber = 955  targetFunctionValue = 11.86200
predictFunctionValue = 11.86200 valurDiff = 4.55675E-6
DayNumber = 956  targetFunctionValue = 11.84124
predictFunctionValue = 11.84124 valurDiff = 6.53477E-6
DayNumber = 957  targetFunctionValue = 11.82031
predictFunctionValue = 11.82031 valurDiff = 2.55647E-6
DayNumber = 958  targetFunctionValue = 11.79920
predictFunctionValue = 11.79920 valurDiff = 8.20278E-6
DayNumber = 959  targetFunctionValue = 11.77792
predictFunctionValue = 11.77792 valurDiff = 4.94157E-7
DayNumber = 960  targetFunctionValue = 11.75647
predictFunctionValue = 11.75647 valurDiff = 1.48410E-6
DayNumber = 961  targetFunctionValue = 11.73483
predictFunctionValue = 11.73484 valurDiff = 3.67970E-6
DayNumber = 962  targetFunctionValue = 11.71303
predictFunctionValue = 11.71303 valurDiff = 6.83684E-6
DayNumber = 963  targetFunctionValue = 11.69105
predictFunctionValue = 11.69105 valurDiff = 4.30269E-6
DayNumber = 964  targetFunctionValue = 11.66890
predictFunctionValue = 11.66890 valurDiff = 1.69128E-6
DayNumber = 965  targetFunctionValue = 11.64658
predictFunctionValue = 11.64658 valurDiff = 7.90340E-6
DayNumber = 966  targetFunctionValue = 11.62409
predictFunctionValue = 11.62409 valurDiff = 8.19566E-6
DayNumber = 967  targetFunctionValue = 11.60142
predictFunctionValue = 11.60143 valurDiff = 4.52810E-6
DayNumber = 968  targetFunctionValue = 11.57859
predictFunctionValue = 11.57859 valurDiff = 6.21339E-6
DayNumber = 969  targetFunctionValue = 11.55559
predictFunctionValue = 11.55559 valurDiff = 7.36500E-6
DayNumber = 970  targetFunctionValue = 11.53241
predictFunctionValue = 11.53241 valurDiff = 3.67611E-6
DayNumber = 971  targetFunctionValue = 11.50907
predictFunctionValue = 11.50907 valurDiff = 2.04084E-6
```

```
DayNumber = 972  targetFunctionValue = 11.48556
predictFunctionValue = 11.48556 valurDiff = 3.10021E-6
DayNumber = 973  targetFunctionValue = 11.46188
predictFunctionValue = 11.46188 valurDiff = 1.04282E-6
DayNumber = 974  targetFunctionValue = 11.43804
predictFunctionValue = 11.43804 valurDiff = 6.05919E-7
DayNumber = 975  targetFunctionValue = 11.41403
predictFunctionValue = 11.41403 valurDiff = 7.53612E-6
DayNumber = 976  targetFunctionValue = 11.38986
predictFunctionValue = 11.38986 valurDiff = 5.25148E-6
DayNumber = 977  targetFunctionValue = 11.36552
predictFunctionValue = 11.36551 valurDiff = 6.09695E-6
DayNumber = 978  targetFunctionValue = 11.34101
predictFunctionValue = 11.34101 valurDiff = 6.10243E-6
DayNumber = 979  targetFunctionValue = 11.31634
predictFunctionValue = 11.31634 valurDiff = 1.14757E-6
DayNumber = 980  targetFunctionValue = 11.29151
predictFunctionValue = 11.29151 valurDiff = 6.88624E-6
DayNumber = 981  targetFunctionValue = 11.26652
predictFunctionValue = 11.26652 valurDiff = 1.22488E-6
DayNumber = 982  targetFunctionValue = 11.24137
predictFunctionValue = 11.24137 valurDiff = 7.90076E-6
DayNumber = 983  targetFunctionValue = 11.21605
predictFunctionValue = 11.21605 valurDiff = 6.28815E-6
DayNumber = 984  targetFunctionValue = 11.19058
predictFunctionValue = 11.19058 valurDiff = 6.75453E-7
DayNumber = 985  targetFunctionValue = 11.16495
predictFunctionValue = 11.16495 valurDiff = 7.05756E-6
DayNumber = 986  targetFunctionValue = 11.13915
predictFunctionValue = 11.13915 valurDiff = 4.99135E-6
DayNumber = 987  targetFunctionValue = 11.11321
predictFunctionValue = 11.11321 valurDiff = 8.69072E-6
DayNumber = 988  targetFunctionValue = 11.08710
predictFunctionValue = 11.08710 valurDiff = 7.41462E-6
```

```
DayNumber = 989  targetFunctionValue = 11.06084
predictFunctionValue = 11.06084 valurDiff = 1.54419E-6
DayNumber = 990  targetFunctionValue = 11.03442
predictFunctionValue = 11.03442 valurDiff = 4.10382E-6
DayNumber = 991  targetFunctionValue = 11.00785
predictFunctionValue = 11.00785 valurDiff = 1.71356E-6
DayNumber = 992  targetFunctionValue = 10.98112
predictFunctionValue = 10.98112 valurDiff = 5.21117E-6
DayNumber = 993  targetFunctionValue = 10.95424
predictFunctionValue = 10.95424 valurDiff = 4.91220E-7
DayNumber = 994  targetFunctionValue = 10.92721
predictFunctionValue = 10.92721 valurDiff = 7.11803E-7
DayNumber = 995  targetFunctionValue = 10.90003
predictFunctionValue = 10.90003 valurDiff = 8.30447E-6
DayNumber = 996  targetFunctionValue = 10.87270
predictFunctionValue = 10.87270 valurDiff = 6.86302E-6
DayNumber = 997  targetFunctionValue = 10.84522
predictFunctionValue = 10.84522 valurDiff = 6.56004E-6
DayNumber = 998  targetFunctionValue = 10.81759
predictFunctionValue = 10.81759 valurDiff = 6.24024E-6
DayNumber = 999  targetFunctionValue = 10.78981
predictFunctionValue = 10.78981 valurDiff = 8.63897E-6
DayNumber = 1000  targetFunctionValue = 10.76181
predictFunctionValue = 10.76188 valurDiff = 7.69201E-6

maxErrorPerc = 1.482606020077711E-6
averErrorPerc = 2.965212040155422E-9
```

The training processing results (that use the micro-batch method) are as follows:

- The maximum error is less than 0.00000148 percent.
- The average error is less than 0.00000000269 percent.

Figure 9-9 shows the chart of the training approximation results (using the micro-batch method). Both charts practically overlap (actual values are in black, and predicted values are in white).

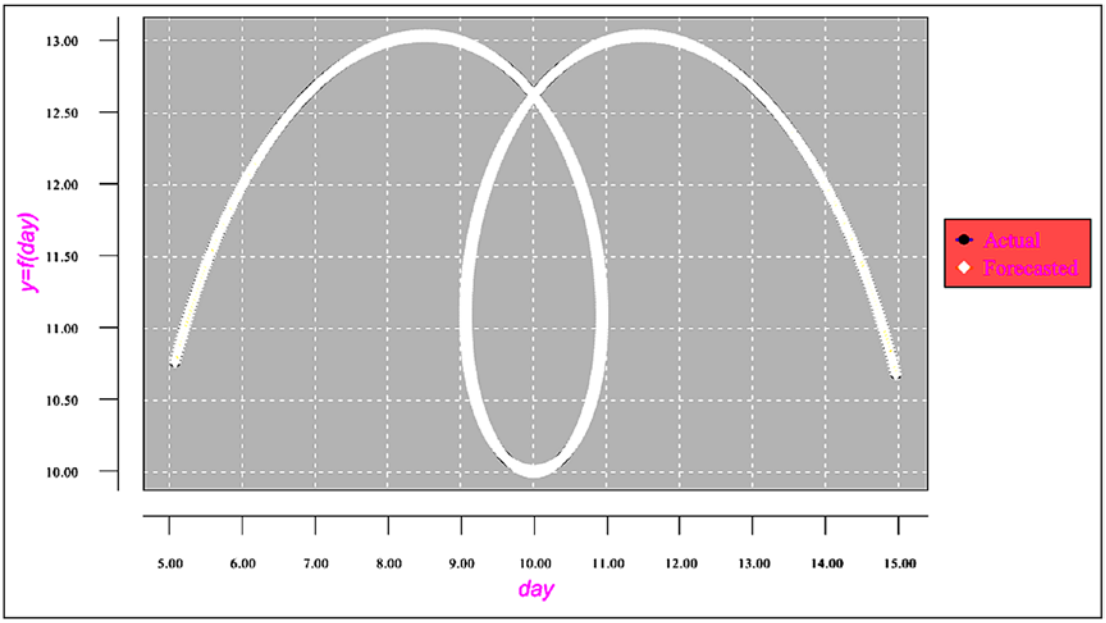


Figure 9-9. *Chart of the training approximation results (using the micro-batch method)*

Like with the normalized training data set, the normalized testing data set is broken into a set of micro-batch files that are now the input to the testing process.

Listing 9-10 shows the ending fragment of the testing results after execution.

Listing 9-10. Ending Fragment of the Testing Processing Results

```
DayNumber = 6.00372 TargettValue = 11.99207
PredictedValue = 12.00232  DiffPerc = 3.84430E-6
DayNumber = 5.98287 TargettValue = 11.97248
PredictedValue = 11.98281  DiffPerc = 2.09221E-6
DayNumber = 5.96212 TargettValue = 11.95270
PredictedValue = 11.96312  DiffPerc = 6.72750E-6
DayNumber = 5.94146 TargettValue = 11.93275
PredictedValue = 11.94325  DiffPerc = 4.20992E-7
DayNumber = 5.92089 TargettValue = 11.91262
PredictedValue = 11.92320  DiffPerc = 1.27514E-6
DayNumber = 5.90042 TargettValue = 11.89231
PredictedValue = 11.90298  DiffPerc = 8.38833E-6
```

```
DayNumber = 5.88004 TargettValue = 11.87183
PredictedValue = 11.88258  DiffPerc = 5.88660E-6
DayNumber = 5.85977 TargettValue = 11.85116
PredictedValue = 11.86200  DiffPerc = 4.55256E-6
DayNumber = 5.83959 TargettValue = 11.83033
PredictedValue = 11.84124  DiffPerc = 6.53740E-6
DayNumber = 5.81952 TargettValue = 11.80932
PredictedValue = 11.82031  DiffPerc = 2.55227E-6
DayNumber = 5.79955 TargettValue = 11.78813
PredictedValue = 11.79920  DiffPerc = 8.20570E-6
DayNumber = 5.77968 TargettValue = 11.76676
PredictedValue = 11.77792  DiffPerc = 4.91208E-7
DayNumber = 5.75992 TargettValue = 11.74523
PredictedValue = 11.75647  DiffPerc = 1.48133E-6
DayNumber = 5.74026 TargettValue = 11.72352
PredictedValue = 11.73484  DiffPerc = 3.68852E-6
DayNumber = 5.72071 TargettValue = 11.70163
PredictedValue = 11.71303  DiffPerc = 6.82806E-6
DayNumber = 5.70128 TargettValue = 11.67958
PredictedValue = 11.69105  DiffPerc = 4.31230E-6
DayNumber = 5.68195 TargettValue = 11.65735
PredictedValue = 11.66890  DiffPerc = 1.70449E-6
DayNumber = 5.66274 TargettValue = 11.63495
PredictedValue = 11.64658  DiffPerc = 7.91193E-6
DayNumber = 5.64364 TargettValue = 11.61238
PredictedValue = 11.62409  DiffPerc = 8.20057E-6
DayNumber = 5.62465 TargettValue = 11.58964
PredictedValue = 11.60143  DiffPerc = 4.52651E-6
DayNumber = 5.60578 TargettValue = 11.56673
PredictedValue = 11.57859  DiffPerc = 6.20537E-6
DayNumber = 5.58703 TargettValue = 11.54365
PredictedValue = 11.55559  DiffPerc = 7.37190E-6
DayNumber = 5.56840 TargettValue = 11.52040
PredictedValue = 11.53241  DiffPerc = 3.68228E-6
```

```
DayNumber = 5.54989 TargettValue = 11.49698
PredictedValue = 11.50907  DiffPerc = 2.05114E-6
DayNumber = 5.53150 TargettValue = 11.47340
PredictedValue = 11.48556  DiffPerc = 3.10919E-6
DayNumber = 5.51323 TargettValue = 11.44965
PredictedValue = 11.46188  DiffPerc = 1.03517E-6
DayNumber = 5.49509 TargettValue = 11.42573
PredictedValue = 11.43804  DiffPerc = 6.10184E-7
DayNumber = 5.47707 TargettValue = 11.40165
PredictedValue = 11.41403  DiffPerc = 7.53367E-6
DayNumber = 5.45918 TargettValue = 11.37740
PredictedValue = 11.38986  DiffPerc = 5.25199E-6
DayNumber = 5.44142 TargettValue = 11.35299
PredictedValue = 11.36551  DiffPerc = 6.09026E-6
DayNumber = 5.42379 TargettValue = 11.32841
PredictedValue = 11.34101  DiffPerc = 6.09049E-6
DayNumber = 5.40629 TargettValue = 11.30368
PredictedValue = 11.31634  DiffPerc = 1.13713E-6
DayNumber = 5.38893 TargettValue = 11.27878
PredictedValue = 11.29151  DiffPerc = 6.88165E-6
DayNumber = 5.37169 TargettValue = 11.25371
PredictedValue = 11.26652  DiffPerc = 1.22300E-6
DayNumber = 5.35460 TargettValue = 11.22849
PredictedValue = 11.24137  DiffPerc = 7.89661E-6
DayNumber = 5.33763 TargettValue = 11.20311
PredictedValue = 11.21605  DiffPerc = 6.30025E-6
DayNumber = 5.32081 TargettValue = 11.17756
PredictedValue = 11.19058  DiffPerc = 6.76200E-7
DayNumber = 5.30412 TargettValue = 11.15186
PredictedValue = 11.16495  DiffPerc = 7.04606E-6
DayNumber = 5.28758 TargettValue = 11.12601
PredictedValue = 11.13915  DiffPerc = 4.98925E-6
DayNumber = 5.27118 TargettValue = 11.09999
PredictedValue = 11.11321  DiffPerc = 8.69060E-6
```

```
DayNumber = 5.25492 TargettValue = 11.07382
PredictedValue = 11.08710  DiffPerc = 7.41171E-6
DayNumber = 5.23880 TargettValue = 11.04749
PredictedValue = 11.06084  DiffPerc = 1.54138E-6
DayNumber = 5.22283 TargettValue = 11.02101
PredictedValue = 11.03442  DiffPerc = 4.09728E-6
DayNumber = 5.20701 TargettValue = 10.99437
PredictedValue = 11.00785  DiffPerc = 1.71899E-6
DayNumber = 5.19133 TargettValue = 10.96758
PredictedValue = 10.98112  DiffPerc = 5.21087E-6
DayNumber = 5.17581 TargettValue = 10.94064
PredictedValue = 10.95424  DiffPerc = 4.97273E-7
DayNumber = 5.16043 TargettValue = 10.91355
PredictedValue = 10.92721  DiffPerc = 7.21563E-7
DayNumber = 5.14521 TargettValue = 10.88630
PredictedValue = 10.90003  DiffPerc = 8.29551E-6
DayNumber = 5.13013 TargettValue = 10.85891
PredictedValue = 10.87270  DiffPerc = 6.86988E-6
DayNumber = 5.11522 TargettValue = 10.83136
PredictedValue = 10.84522  DiffPerc = 6.55538E-6
DayNumber = 5.10046 TargettValue = 10.80367
PredictedValue = 10.81759  DiffPerc = 6.24113E-6
DayNumber = 5.08585 TargettValue = 10.77584
PredictedValue = 10.78981  DiffPerc = 8.64007E-6

maxErrorPerc = 9.002677165459051E-6
averErrorPerc = 4.567068981414947E-6
```

The testing processing results (using the micro-batch method) are as follows:

- The maximum error is less than 0.00000900 percent.
- The average error is less than 0.00000457 percent.

Figure 9-10 shows the chart of the testing processing results (using the micro-batch method). Again, both charts practically overlap (actual values are black, and predicted values are white).

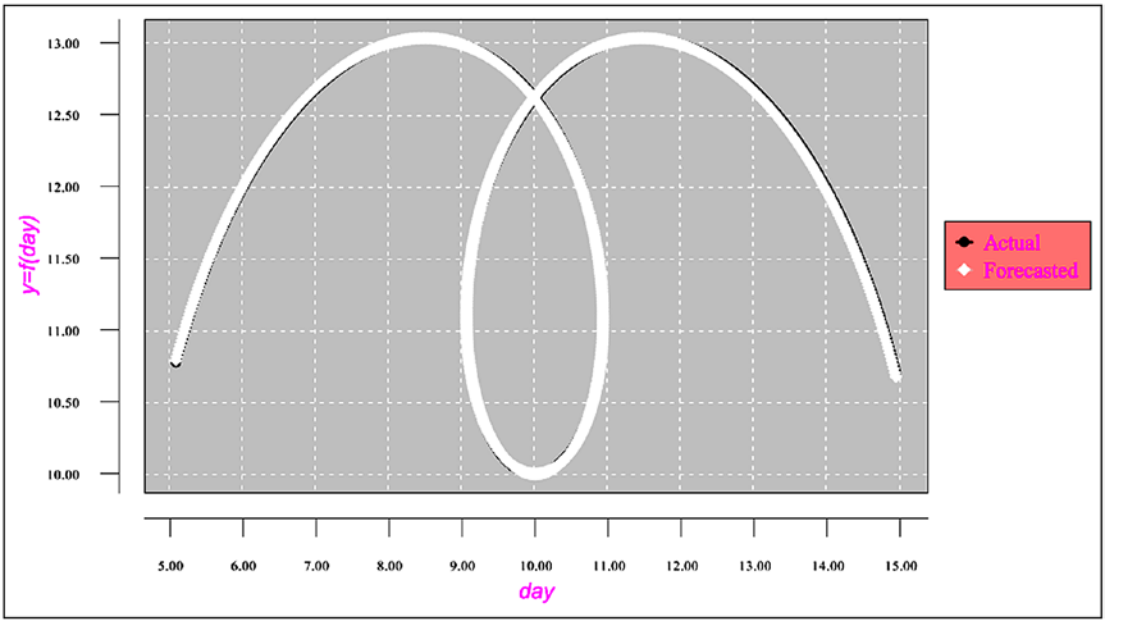


Figure 9-10. *Chart of the testing processing results (using the micro-batch method)*

Summary

Neural networks have problems approximating continuous functions with complex topologies. It is difficult to obtain a good-quality approximation for such functions. This chapter showed that the micro-batch method is able to approximate such functions with high-precision results. Up to now you used neural networks to solve the regression tasks. In the next chapter, you will learn how to use neural networks for the classification of objects.

CHAPTER 10

Using Neural Networks to Classify Objects

In this chapter, you'll use a neural network to classify objects. *Classification* means recognizing various objects and determining the class to which those objects belong. As with many areas of artificial intelligence, classification is easily done by humans but can be quite difficult for computers.

Example 6: Classification of Records

For this example, you are presented with five books, and each book belongs to a different area of human knowledge: medical, programming, engineering, electrical, or music. You are also given the three most frequently used words in each book. See Listing 10-1.

Many records are provided for this example, and each record includes three words. If all three words in a record belong to a certain book, then the program should determine that the record belongs to that book. If a record has a mixture of words that don't belong to any of the five books, then the program should classify that record as belonging to an unknown book.

This example looks simple; in fact, it might look like it does not need a neural network and that the problem could be resolved by using regular programming logic. However, when the volume of books and records becomes much larger, with a large number of unpredictable combinations of words included in each record and with the condition that only a certain small number of words from one book is sufficient for a record to belong to a certain book, then artificial intelligence is needed to handle such a task.

I. Livshin, *Artificial Neural Networks with Java*, https://doi.org/10.1007/978-1-4842-4421-0_10

Listing 10-1. List of Five Books with Three Most Frequent Words

```
Book 1. Medical.
 surgery, blood, prescription,

Book 2. Programming.
file, java, debugging

Book3. Engineering.
combustion, screw, machine

Book 4. Electrical.
volt, solenoid, diode

Book 5. Music.
adagio,  hymn, opera,

Extra words. We will use words in this list to include them in the test
dataset.
customer, wind, grass, paper, calculator, flower, printer ,desk, photo,
map, pen, floor.
```

To simplify the processing, you assign numbers to all words and use those numbers instead of words when building the training and testing data sets. Table 10-1 shows the words-to-numbers cross-reference.

Table 10-1. *Words-to-Numbers Cross-Reference*

Word	Assigned Number
surgery	1
blood	2
prescription	3
file	4
java	5
debugging	6
combustion	7

(*continued*)

Table 10-1. (*continued*)

Word	Assigned Number
screw	8
machine	9
volt	10
solenoid	11
diode	12
adagio	13
hymn	14
opera	15
customer	16
wind	17
grass	18
paper	19
calculator	20
flower	21
printer	22
desk	23
photo	24
map	25
pen	26
floor	27

Training Data Set

Each record in the training data set consists of three fields that hold words from the list of most frequently used words in the books. Also included in each record are five target fields, indicating the book to which the record belongs. Notice that this is the first example in the book where the network has five target fields. This information is used for training the network. For example, the combination 1, 0, 0, 0, 0 means book #1;

the combination 0, 1, 0, 0, 0 means book #2; and so on. Also, for each book, you need to build six records in the training data set instead of one. These six records include all the possible permutation of words in a record. Table 10-2 shows all the possible permutation of words in all records. I'm using italics to highlight the portion of each record that holds the word numbers.

Table 10-2. *Permutation of Words in All Records*

Records for Book 1							
1	*2*	*3*	1	0	0	0	0
1	*3*	*2*	1	0	0	0	0
2	*1*	*3*	1	0	0	0	0
2	*3*	*1*	1	0	0	0	0
3	*1*	*2*	1	0	0	0	0
3	*2*	*1*	1	0	0	0	0
Records for Book 2							
4	*5*	*6*	0	1	0	0	0
4	*6*	*5*	0	1	0	0	0
5	*4*	*6*	0	1	0	0	0
5	*6*	*4*	0	1	0	0	0
6	*4*	*5*	0	1	0	0	0
6	*5*	*4*	0	1	0	0	0
Records for Book 3							
7	*8*	*9*	0	0	1	0	0
7	*9*	*8*	0	0	1	0	0
8	*7*	*9*	0	0	1	0	0
8	*9*	*7*	0	0	1	0	0
9	*7*	*8*	0	0	1	0	0
9	*8*	*7*	0	0	1	0	0

(*continued*)

Table 10-2. (*continued*)

Records for Book 4							
10	*11*	*12*	0	0	0	1	0
10	*12*	*11*	0	0	0	1	0
11	*10*	*12*	0	0	0	1	0
11	*12*	*10*	0	0	0	1	0
12	*10*	*11*	0	0	0	1	0
12	*11*	*10*	0	0	0	1	0
Records for Book 5							
13	*14*	*15*	0	0	0	0	1
13	*15*	*14*	0	0	0	0	1
14	*13*	*15*	0	0	0	0	1
14	*15*	*13*	0	0	0	0	1
15	*13*	*14*	0	0	0	0	1
15	*14*	*13*	0	0	0	0	1

Putting it all together, Table 10-3 shows the training data set.

Table 10-3. *Training Data Set*

Word1	Word2	Word3	Target1	Target2	Target3	Target4	Target5
1	2	3	1	0	0	0	0
1	3	2	1	0	0	0	0
2	1	3	1	0	0	0	0
2	3	1	1	0	0	0	0
3	1	2	1	0	0	0	0
3	2	1	1	0	0	0	0

(*continued*)

Table 10-3. (*continued*)

Word1	Word2	Word3	Target1	Target2	Target3	Target4	Target5
4	5	6	0	1	0	0	0
4	6	5	0	1	0	0	0
5	4	6	0	1	0	0	0
5	6	4	0	1	0	0	0
6	4	5	0	1	0	0	0
6	5	4	0	1	0	0	0
7	8	9	0	0	1	0	0
7	9	8	0	0	1	0	0
8	7	9	0	0	1	0	0
8	9	7	0	0	1	0	0
9	7	8	0	0	1	0	0
9	8	7	0	0	1	0	0
10	11	12	0	0	0	1	0
10	12	11	0	0	0	1	0
11	10	12	0	0	0	1	0
11	12	10	0	0	0	1	0
12	10	11	0	0	0	1	0
12	11	10	0	0	0	1	0
13	14	15	0	0	0	0	1
13	15	14	0	0	0	0	1
14	13	15	0	0	0	0	1
14	15	13	0	0	0	0	1
15	13	14	0	0	0	0	1
15	14	13	0	0	0	0	1

Network Architecture

The network has an input layer with three input neurons, six hidden layers with seven neurons each, and an output layer with five neurons. Figure 10-1 shows the network architecture.

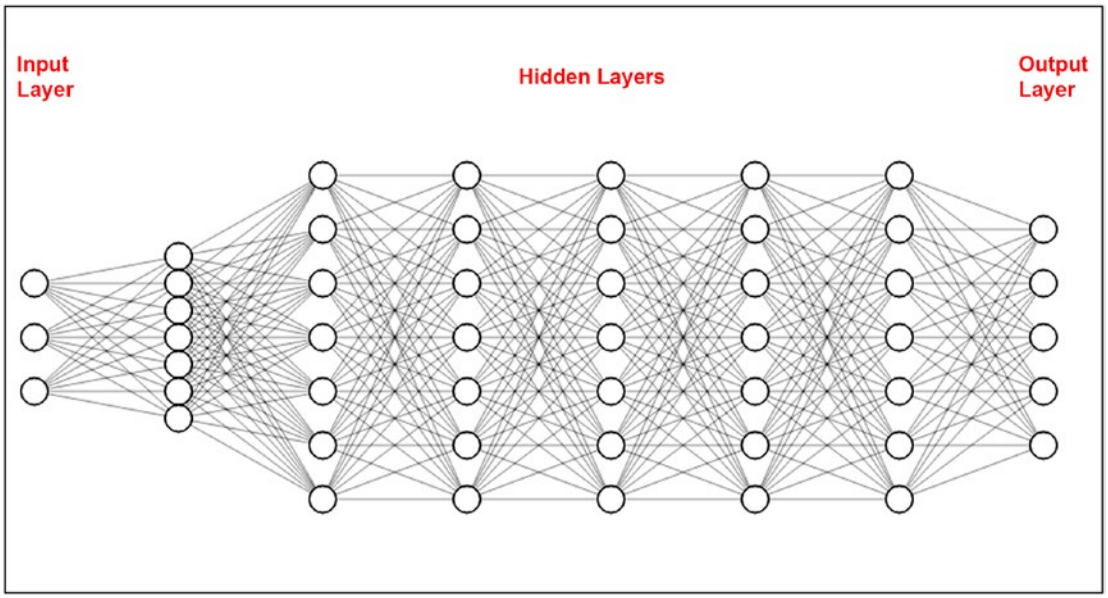


Figure 10-1. *Network architecture*

Testing Data Set

The testing data set consists of records with randomly included words/numbers. These records don't belong to any single book, despite that some of them include one or two words from the most frequently used list. Table 10-4 shows the testing data set.

Table 10-4. *Testing Data Set*

Word1	Word2	Word3	Target 1	Target 2	Target 3	Target 4	Target 5
1	2	16	0	0	0	0	0
4	17	5	0	0	0	0	0
8	9	18	0	0	0	0	0
19	10	11	0	0	0	0	0
15	20	13	0	0	0	0	0
27	1	26	0	0	0	0	0
14	23	22	0	0	0	0	0
21	20	18	0	0	0	0	0
25	23	24	0	0	0	0	0
11	9	6	0	0	0	0	0
3	5	8	0	0	0	0	0
6	10	15	0	0	0	0	0
16	17	18	0	0	0	0	0
19	1	8	0	0	0	0	0
27	23	17	0	0	0	0	0

There is no need to include the target columns in the testing file; however, they are included for convenience (to compare the predicted and actual results). These columns are not used for processing. As usual, you need to normalize the training and testing data sets on the interval [-1, 1]. Because this example features multiple neurons in the input and output layers, you will need the normalization source code.

Program Code for Data Normalization

Listing 10-2 shows the program code that normalizes the training and testing data sets.

Listing 10-2. Program Code for Data Normalization

```
// ============================================================
// This program normalizes all columns of the input CSV dataset putting the
// result in the output CSV file.
//
// The first column of the input dataset includes the X point value and the
// second column of the input dataset includes the value of the function at
// the point X.
// ============================================================

package sample5_norm;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.file.*;

public class Sample5_Norm
{
   // Interval to normalize
   static double Nh =  1;
   static double Nl = -1;

   // First column
   static double minXPointDl = 1.00;
   static double maxXPointDh = 1000.00;

   // Second column - target data
   static double minTargetValueDl = 60.00;
   static double maxTargetValueDh = 1600.00;
```

```
public static double normalize(double value, double Dh, double Dl)
 {
   double normalizedValue = (value - Dl)*(Nh - Nl)/(Dh - Dl) + Nl;

   return normalizedValue;
 }

public static void main(String[] args)
 {
    // Normalize train file
    String inputFileName = "C:/Book_Examples/Sample5_Train_Real.csv";
    String outputNormFileName = "C:/Book_Examples/Sample5_Train_Norm.csv";

    // Normalize test file
    // String inputFileName = "C:/Book_Examples/Sample5_Test_Real.csv";
    // String outputNormFileName = "C:/Book_Examples/Sample5_Test_Norm.csv";

    BufferedReader br = null;
    PrintWriter out = null;

    String line = "";
    String cvsSplitBy = ",";

    double inputXPointValue;
    double targetXPointValue;

    double normInputXPointValue;
    double normTargetXPointValue;

    String strNormInputXPointValue;
    String strNormTargetXPointValue;

    String fullLine;

    int i = -1;

    try
     {
      Files.deleteIfExists(Paths.get(outputNormFileName));

      br = new BufferedReader(new FileReader(inputFileName));
      out = new
       PrintWriter(new BufferedWriter(new FileWriter(outputNormFileName)));
```

```
while ((line = br.readLine()) != null)
 {
    i++;

    if(i == 0)
     {
       // Write the label line
       out.println(line);
     }
    else
    {
     // Break the line using comma as separator
     String[] workFields = line.split(cvsSplitBy);

     inputXPointValue = Double.parseDouble(workFields[0]);
     targetXPointValue = Double.parseDouble( workFields[1]);

     // Normalize these fields
     normInputXPointValue =
       normalize(inputXPointValue, maxXPointDh, minXPointDl);
     normTargetXPointValue =
     normalize(targetXPointValue, maxTargetValueDh, minTargetValueDl);

     // Convert normalized fields to string, so they can be inserted
     //into the output CSV file
     strNormInputXPointValue = Double.toString(normInput
     XPointValue);
     strNormTargetXPointValue = Double.toString(normTarget
     XPointValue);

     // Concatenate these fields into a string line with
     //coma separator
     fullLine  =
       strNormInputXPointValue + "," + strNormTargetXPointValue;

     // Put fullLine into the output file
     out.println(fullLine);

    } // End of IF Else
```

```
            }    // end of while

          }  // end of TRY

        catch (FileNotFoundException e)
         {
             e.printStackTrace();
             System.exit(1);
         }
        catch (IOException io)
         {
             io.printStackTrace();
         }
        finally
         {
           if (br != null)
             {
                 try
                  {
                     br.close();
                     out.close();
                  }
                 catch (IOException e)
                  {
                     e.printStackTrace();
                  }
             }
         }

    }

}
```

Table 10-5 shows the normalized training data set.

Table 10-5. *Normalized Training Data Set*

Word 1	Word 2	Word 3	Target 1	Target 2	Target 3	Target 4	Target 5
-1	-0.966101695	-0.93220339	1	-1	-1	-1	-1
-1	-0.93220339	-0.966101695	1	-1	-1	-1	-1
-0.966101695	-1	-0.93220339	1	-1	-1	-1	-1
-0.966101695	-0.93220339	-1	1	-1	-1	-1	-1
-0.93220339	-1	-0.966101695	1	-1	-1	-1	-1
-0.93220339	-0.966101695	-1	1	-1	-1	-1	-1
-0.898305085	-0.86440678	-0.830508475	-1	1	-1	-1	-1
-0.898305085	-0.830508475	-0.86440678	-1	1	-1	-1	-1
-0.86440678	-0.898305085	-0.830508475	-1	1	-1	-1	-1
-0.86440678	-0.830508475	-0.898305085	-1	1	-1	-1	-1
-0.830508475	-0.898305085	-0.86440678	-1	1	-1	-1	-1
-0.830508475	-0.86440678	-0.898305085	-1	1	-1	-1	-1
-0.796610169	-0.762711864	-0.728813559	-1	-1	1	-1	-1
-0.796610169	-0.728813559	-0.762711864	-1	-1	1	-1	-1
-0.762711864	-0.796610169	-0.728813559	-1	-1	1	-1	-1
-0.762711864	-0.728813559	-0.796610169	-1	-1	1	-1	-1
-0.728813559	-0.796610169	-0.762711864	-1	-1	1	-1	-1
-0.728813559	-0.762711864	-0.796610169	-1	-1	1	-1	-1
-0.694915254	-0.661016949	-0.627118644	-1	-1	-1	1	-1
-0.694915254	-0.627118644	-0.661016949	-1	-1	-1	1	-1
-0.661016949	-0.694915254	-0.627118644	-1	-1	-1	1	-1
-0.661016949	-0.627118644	-0.694915254	-1	-1	-1	1	-1
-0.627118644	-0.694915254	-0.661016949	-1	-1	-1	1	-1
-0.627118644	-0.661016949	-0.694915254	-1	-1	-1	1	-1

(*continued*)

Table 10-5. (*continued*)

Word 1	Word 2	Word 3	Target 1	Target 2	Target 3	Target 4	Target 5
-0.593220339	-0.559322034	-0.525423729	-1	-1	-1	-1	1
-0.593220339	-0.525423729	-0.559322034	-1	-1	-1	-1	1
-0.559322034	-0.593220339	-0.525423729	-1	-1	-1	-1	1
-0.559322034	-0.525423729	-0.593220339	-1	-1	-1	-1	1
-0.525423729	-0.593220339	-0.559322034	-1	-1	-1	-1	1
-0.525423729	-0.559322034	-0.593220339	-1	-1	-1	-1	1

Table 10-6 shows the normalized testing data set.

Table 10-6. *Normalized Testing Data Set*

Word 1	Word 2	Word 3	Target 1	Target 2	Target 3	Target 4	Target 5
-1	-0.966101695	-0.491525424	-1	-1	-1	-1	-1
-0.898305085	-0.457627119	-0.86440678	-1	-1	-1	-1	-1
-0.762711864	-0.728813559	-0.423728814	-1	-1	-1	-1	-1
-0.389830508	-0.694915254	-0.661016949	-1	-1	-1	-1	-1
-0.525423729	-0.355932203	-0.593220339	-1	-1	-1	-1	-1
-0.118644068	-1	0.152542373	-1	-1	-1	-1	-1
-0.559322034	-0.254237288	-0.288135593	-1	-1	-1	-1	-1
-0.322033898	-0.355932203	-0.423728814	-1	-1	-1	-1	-1
-0.186440678	-0.254237288	-0.220338983	-1	-1	-1	-1	-1
-0.661016949	-0.728813559	-0.830508475	-1	-1	-1	-1	-1
-0.93220339	-0.86440678	-0.762711864	-1	-1	-1	-1	-1
-0.830508475	-0.69491525	-0.525423729	-1	-1	-1	-1	-1
-0.491525424	-0.45762711	-0.423728814	-1	-1	-1	-1	-1
-0.389830508	-1	-0.762711864	-1	-1	-1	-1	-1
-0.118644068	-0.25423728	-0.457627119	-1	-1	-1	-1	-1

Program Code for Classification

Listing 10-3 shows the classification program code.

Listing 10-3. Classification Program Code

```
// ============================================================================
// Example of using neural network for classification of objects.
// The normalized training/testing files consists of records of the following
// format: 3 input fields (word numbers)and 5 target fields (indicate the book
// the record belongs to).
// ============================================================================

package sample6;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.*;
import java.util.Properties;
import java.time.YearMonth;
import java.awt.Color;
import java.awt.Font;
import java.io.BufferedReader;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.LocalDate;
import java.time.Month;
import java.time.ZoneId;
import java.util.ArrayList;
```

```
import java.util.Calendar;
import java.util.Date;
import java.util.List;
import java.util.Locale;
import java.util.Properties;

import org.encog.Encog;
import org.encog.engine.network.activation.ActivationTANH;
import org.encog.engine.network.activation.ActivationReLU;
import org.encog.ml.data.MLData;
import org.encog.ml.data.MLDataPair;
import org.encog.ml.data.MLDataSet;
import org.encog.ml.data.buffer.MemoryDataLoader;
import org.encog.ml.data.buffer.codec.CSVDataCODEC;
import org.encog.ml.data.buffer.codec.DataSetCODEC;
import org.encog.neural.networks.BasicNetwork;
import org.encog.neural.networks.layers.BasicLayer;
import org.encog.neural.networks.training.propagation.resilient.
ResilientPropagation;
import org.encog.persist.EncogDirectoryPersistence;
import org.encog.util.csv.CSVFormat;

import org.knowm.xchart.SwingWrapper;
import org.knowm.xchart.XYChart;
import org.knowm.xchart.XYChartBuilder;
import org.knowm.xchart.XYSeries;
import org.knowm.xchart.demo.charts.ExampleChart;
import org.knowm.xchart.style.Styler.LegendPosition;
import org.knowm.xchart.style.colors.ChartColor;
import org.knowm.xchart.style.colors.XChartSeriesColors;
import org.knowm.xchart.style.lines.SeriesLines;
import org.knowm.xchart.style.markers.SeriesMarkers;
import org.knowm.xchart.BitmapEncoder;
import org.knowm.xchart.BitmapEncoder.BitmapFormat;
import org.knowm.xchart.QuickChart;
import org.knowm.xchart.SwingWrapper;
```

```
public class Sample6 implements ExampleChart<XYChart>
{
    // Interval to normalize data
   static double Nh;
   static double Nl;

   // Normalization parameters for workBook number
   static double minWordNumberDl;
   static double maxWordNumberDh;

   // Normalization parameters for target values
   static double minTargetValueDl;
   static double maxTargetValueDh;

   static double doublePointNumber = 0.00;
   static int intPointNumber = 0;
   static InputStream input = null;
   static double[] arrPrices = new double[2500];
   static double normInputWordNumber_01 = 0.00;
   static double normInputWordNumber_02 = 0.00;
   static double normInputWordNumber_03 = 0.00;
   static double denormInputWordNumber_01 = 0.00;
   static double denormInputWordNumber_02 = 0.00;
   static double denormInputWordNumber_03 = 0.00;
   static double normTargetBookNumber_01 = 0.00;
   static double normTargetBookNumber_02 = 0.00;
   static double normTargetBookNumber_03 = 0.00;
   static double normTargetBookNumber_04 = 0.00;
   static double normTargetBookNumber_05 = 0.00;
   static double normPredictBookNumber_01 = 0.00;
   static double normPredictBookNumber_02 = 0.00;
   static double normPredictBookNumber_03 = 0.00;
   static double normPredictBookNumber_04 = 0.00;
   static double normPredictBookNumber_05 = 0.00;
   static double denormTargetBookNumber_01 = 0.00;
   static double denormTargetBookNumber_02 = 0.00;
   static double denormTargetBookNumber_03 = 0.00;
```

```
    static double denormTargetBookNumber_04 = 0.00;
    static double denormTargetBookNumber_05 = 0.00;
    static double denormPredictBookNumber_01 = 0.00;
    static double denormPredictBookNumber_02 = 0.00;
    static double denormPredictBookNumber_03 = 0.00;
    static double denormPredictBookNumber_04 = 0.00;
    static double denormPredictBookNumber_05 = 0.00;
    static double normDifferencePerc = 0.00;
    static double denormPredictXPointValue_01 = 0.00;
    static double denormPredictXPointValue_02 = 0.00;
    static double denormPredictXPointValue_03 = 0.00;
    static double denormPredictXPointValue_04 = 0.00;
    static double denormPredictXPointValue_05 = 0.00;
    static double valueDifference = 0.00;
    static int numberOfInputNeurons;
    static int numberOfOutputNeurons;
    static int intNumberOfRecordsInTestFile;
    static String trainFileName;
    static String priceFileName;
    static String testFileName;
    static String chartTrainFileName;
    static String chartTestFileName;
    static String networkFileName;
    static int workingMode;
    static String cvsSplitBy = ",";
    static int returnCode;

    static List<Double> xData = new ArrayList<Double>();
    static List<Double> yData1 = new ArrayList<Double>();
    static List<Double> yData2 = new ArrayList<Double>();

    static XYChart Chart;
```

```
@Override
public XYChart getChart()
 {

  // Create Chart
  Chart = new XYChartBuilder().width(900).height(500).title(getClass().
            getSimpleName()).xAxisTitle("x").yAxisTitle("y= f(x)").build();

  // Customize Chart
  Chart.getStyler().setPlotBackgroundColor(ChartColor.getAWTColor
  (ChartColor.GREY));
  Chart.getStyler().setPlotGridLinesColor(new Color(255, 255, 255));
  Chart.getStyler().setChartBackgroundColor(Color.WHITE);
  Chart.getStyler().setLegendBackgroundColor(Color.PINK);
  Chart.getStyler().setChartFontColor(Color.MAGENTA);
  Chart.getStyler().setChartTitleBoxBackgroundColor(new Color(0, 222, 0));
  Chart.getStyler().setChartTitleBoxVisible(true);
  Chart.getStyler().setChartTitleBoxBorderColor(Color.BLACK);
  Chart.getStyler().setPlotGridLinesVisible(true);
  Chart.getStyler().setAxisTickPadding(20);
  Chart.getStyler().setAxisTickMarkLength(15);
  Chart.getStyler().setPlotMargin(20);
  Chart.getStyler().setChartTitleVisible(false);
  Chart.getStyler().setChartTitleFont(new Font(Font.MONOSPACED,
  Font.BOLD, 24));
  Chart.getStyler().setLegendFont(new Font(Font.SERIF, Font.PLAIN, 18));
  Chart.getStyler().setLegendPosition(LegendPosition.InsideSE);
  Chart.getStyler().setLegendSeriesLineLength(12);
  Chart.getStyler().setAxisTitleFont(new Font(Font.SANS_SERIF,
  Font.ITALIC, 18));
  Chart.getStyler().setAxisTickLabelsFont(new Font(Font.SERIF,
  Font.PLAIN, 11));
  Chart.getStyler().setDatePattern("yyyy-MM");
  Chart.getStyler().setDecimalPattern("#0.00");
```

```
    // Interval to normalize data
    Nh =  1;
    Nl = -1;

   // Normalization parameters for workBook number
   double minWordNumberDl = 1.00;
   double maxWordNumberDh = 60.00;

   // Normalization parameters for target values
   minTargetValueDl = 0.00;
   maxTargetValueDh = 1.00;

   // Configuration

   // Set the mode to run the program

   workingMode = 1; // Training mode

   if(workingMode == 1)
    {
       // Training mode
       intNumberOfRecordsInTestFile = 31;
       trainFileName = "C:/My_Neural_Network_Book/Book_Examples/Sample6_
       Norm_Train_File.csv";

     File file1 = new File(chartTrainFileName);
     File file2 = new File(networkFileName);

     if(file1.exists())
       file1.delete();

     if(file2.exists())
       file2.delete();

     returnCode = 0;    // Clear the return code variable

     do
      {
        returnCode = trainValidateSaveNetwork();
      } while (returnCode > 0);
    }   // End the training mode
```

```
else
 {
   // Testing mode
   intNumberOfRecordsInTestFile = 16;
     testFileName = "C:/My_Neural_Network_Book/Book_Examples/Sample6_
     Norm_Test_File.csv";
     networkFileName =
        "C:/My_Neural_Network_Book/Book_Examples/Sample6_Saved_Network_
        File.csv";
     numberOfInputNeurons = 3;
     numberOfOutputNeurons = 5;

     loadAndTestNetwork();
 }

 Encog.getInstance().shutdown();

 return Chart;

}  // End of the method

// =======================================================
// Load CSV to memory.
// @return The loaded dataset.
// =======================================================
public static MLDataSet loadCSV2Memory(String filename, int input,
int ideal, boolean headers, CSVFormat format, boolean significance)
  {
     DataSetCODEC codec = new CSVDataCODEC(new File(filename), format,
       headers, input, ideal, significance);
     MemoryDataLoader load = new MemoryDataLoader(codec);
     MLDataSet dataset = load.external2Memory();
     return dataset;
  }
```

```
// =======================================================
//  The main method.
//  @param Command line arguments. No arguments are used.
// =======================================================
public static void main(String[] args)
 {
   ExampleChart<XYChart> exampleChart = new Sample6();
   XYChart Chart = exampleChart.getChart();
   new SwingWrapper<XYChart>(Chart).displayChart();
 } // End of the main method

//=======================================================================
// This method trains, validates, and saves the trained network file on disk
//=======================================================================
static public int trainValidateSaveNetwork()
 {
   // Load the training CSV file in memory
   MLDataSet trainingSet =
     loadCSV2Memory(trainFileName,numberOfInputNeurons,
     numberOfOutputNeurons,true,CSVFormat.ENGLISH,false);

   // create a neural network
   BasicNetwork network = new BasicNetwork();

   // Input layer
   network.addLayer(new BasicLayer(null,true,3));

   // Hidden layer
   network.addLayer(new BasicLayer(new ActivationTANH(),true,7));
   network.addLayer(new BasicLayer(new ActivationTANH(),true,7));
   network.addLayer(new BasicLayer(new ActivationTANH(),true,7));
   network.addLayer(new BasicLayer(new ActivationTANH(),true,7));
   network.addLayer(new BasicLayer(new ActivationTANH(),true,7));
   network.addLayer(new BasicLayer(new ActivationTANH(),true,7));
```

```
// Output layer
network.addLayer(new BasicLayer(new ActivationTANH(),false,5));

network.getStructure().finalizeStructure();
network.reset();

// train the neural network
final ResilientPropagation train = new ResilientPropagation(network,
trainingSet);

int epoch = 1;

do
 {
    train.iteration();
    System.out.println("Epoch #" + epoch + " Error:" +
    train.getError());

   epoch++;

   if (epoch >= 1000 && network.calculateError(trainingSet) >
   0.0000000000000012)
       {
        returnCode = 1;

        System.out.println("Try again");
        return returnCode;
       }

 } while (network.calculateError(trainingSet) > 0.0000000000000011);

// Save the network file
EncogDirectoryPersistence.saveObject(new File(networkFileName),
network);

System.out.println("Neural Network Results:");

int m = 0;

for(MLDataPair pair: trainingSet)
  {
       m++;
```

```
final MLData output = network.compute(pair.getInput());

MLData inputData = pair.getInput();
MLData actualData = pair.getIdeal();
MLData predictData = network.compute(inputData);

// Calculate and print the results

normInputWordNumber_01 = inputData.getData(0);
normInputWordNumber_02 = inputData.getData(1);
normInputWordNumber_03 = inputData.getData(2);

normTargetBookNumber_01 = actualData.getData(0);
normTargetBookNumber_02 = actualData.getData(1);
normTargetBookNumber_03 = actualData.getData(2);
normTargetBookNumber_04 = actualData.getData(3);
normTargetBookNumber_05 = actualData.getData(4);

normPredictBookNumber_01 = predictData.getData(0);
normPredictBookNumber_02 = predictData.getData(1);
normPredictBookNumber_03 = predictData.getData(2);
normPredictBookNumber_04 = predictData.getData(3);
normPredictBookNumber_05 = predictData.getData(4);

// De-normalize the results
denormInputWordNumber_01 = ((minWordNumberDl -
maxWordNumberDh)*normInputWordNumber_01 - Nh*minWordNumberDl +
maxWordNumberDh *Nl)/(Nl - Nh);

denormInputWordNumber_02 = ((minWordNumberDl -
maxWordNumberDh)*normInputWordNumber_02 - Nh*minWordNumberDl +
maxWordNumberDh *Nl)/(Nl - Nh);

denormInputWordNumber_03 = ((minWordNumberDl -
maxWordNumberDh)*normInputWordNumber_03 - Nh*minWordNumberDl +
maxWordNumberDh *Nl)/(Nl - Nh);

denormTargetBookNumber_01 = ((minTargetValueDl -
maxTargetValueDh)*normTargetBookNumber_01 -
Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);
```

```
denormTargetBookNumber_02 = ((minTargetValueDl -
maxTargetValueDh)*normTargetBookNumber_02 -
Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

denormTargetBookNumber_03 = ((minTargetValueDl -
maxTargetValueDh)*normTargetBookNumber_03 -
Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

denormTargetBookNumber_04 = ((minTargetValueDl -
maxTargetValueDh)*normTargetBookNumber_04 -
Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

denormTargetBookNumber_05 = ((minTargetValueDl -
maxTargetValueDh)*normTargetBookNumber_05 -
 Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

denormPredictBookNumber_01 =((minTargetValueDl -
maxTargetValueDh)*normPredictBookNumber_01 -
Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

denormPredictBookNumber_02 =((minTargetValueDl -
maxTargetValueDh)*normPredictBookNumber_02 -
Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

denormPredictBookNumber_03 =((minTargetValueDl -
maxTargetValueDh)*normPredictBookNumber_03 -
Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

denormPredictBookNumber_04 =((minTargetValueDl -
maxTargetValueDh)*normPredictBookNumber_04 -
Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

denormPredictBookNumber_05 =((minTargetValueDl -
maxTargetValueDh)*normPredictBookNumber_05 -
Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

System.out.println ("RecordNumber = " + m);

System.out.println ("denormTargetBookNumber_01 = " +
denormTargetBookNumber_01 + "denormPredictBookNumber_01 = " +
denormPredictBookNumber_01);
```

```
System.out.println ("denormTargetBookNumber_02 = " +
denormTargetBookNumber_02 + "denormPredictBookNumber_02 = " +
denormPredictBookNumber_02);

System.out.println ("denormTargetBookNumber_03 = " +
denormTargetBookNumber_03 + "denormPredictBookNumber_03 = " +
denormPredictBookNumber_03);

System.out.println ("denormTargetBookNumber_04 = " +
denormTargetBookNumber_04 + "denormPredictBookNumber_04 = " +
denormPredictBookNumber_04);

System.out.println ("denormTargetBookNumber_05 = " +
denormTargetBookNumber_05 + "denormPredictBookNumber_05 = " +
denormPredictBookNumber_05);

//System.out.println (" ");

// Print the classification results
if(Math.abs(denormPredictBookNumber_01) > 0.85)
  if(Math.abs(denormPredictBookNumber_01) > 0.85  &
     Math.abs(denormPredictBookNumber_02) < 0.2   &
     Math.abs(denormPredictBookNumber_03) < 0.2   &
     Math.abs(denormPredictBookNumber_04) < 0.2   &
     Math.abs(denormPredictBookNumber_05) < 0.2)
      {
        System.out.println ("Record 1 belongs to book 1");
        System.out.println (" ");
      }
  else
   {
     System.out.println ("Wrong results for record 1");
     System.out.println (" ");
   }

 if(Math.abs(denormPredictBookNumber_02) > 0.85)
  if(Math.abs(denormPredictBookNumber_01) < 0.2  &
     Math.abs(denormPredictBookNumber_02) > 0.85 &
     Math.abs(denormPredictBookNumber_03) < 0.2  &
     Math.abs(denormPredictBookNumber_04) < 0.2  &
```

```
    Math.abs(denormPredictBookNumber_05) < 0.2)
     {
       System.out.println ("Record 2 belongs to book 2");
       System.out.println (" ");
     }
  else
   {
     System.out.println ("Wrong results for record 2");
     System.out.println (" ");
   }
if(Math.abs(denormPredictBookNumber_03) > 0.85)
 if(Math.abs(denormPredictBookNumber_01) < 0.2 &
    Math.abs(denormPredictBookNumber_02) < 0.2  &
    Math.abs(denormPredictBookNumber_03) > 0.85 &
    Math.abs(denormPredictBookNumber_04) < 0.2  &
    Math.abs(denormPredictBookNumber_05) < 0.2)
     {
       System.out.println ("Record 3 belongs to book 3");
       System.out.println (" ");
     }
 else
  {
    System.out.println ("Wrong results for record 3");
    System.out.println (" ");
  }
if(Math.abs(denormPredictBookNumber_04) > 0.85)
 if(Math.abs(denormPredictBookNumber_01) < 0.2  &
    Math.abs(denormPredictBookNumber_02) < 0.2  &
    Math.abs(denormPredictBookNumber_03) < 0.2  &
    Math.abs(denormPredictBookNumber_04) > 0.85 &
    Math.abs(denormPredictBookNumber_05) < 0.2)
     {
       System.out.println ("Record 4 belongs to book 4");
       System.out.println (" ");
     }
```

```
            else
             {
               System.out.println ("Wrong results for record 4");
               System.out.println (" ");
             }

           if(Math.abs(denormPredictBookNumber_05) > 0.85)
            if(Math.abs(denormPredictBookNumber_01) < 0.2  &
               Math.abs(denormPredictBookNumber_02) < 0.2  &
               Math.abs(denormPredictBookNumber_03) < 0.2  &
               Math.abs(denormPredictBookNumber_04) < 0.2 &
               Math.abs(denormPredictBookNumber_05) > 0.85)
                {
                  System.out.println ("Record 5 belongs to book 5");
                  System.out.println (" ");
                }
            else
             {
               System.out.println ("Wrong results for record 5");
               System.out.println (" ");
             }

      }   // End for pair loop

      returnCode = 0;
      return returnCode;

  }   // End of the method

//=====================================================================
// Load and test the trained network at non-trainable points
//=====================================================================
static public void loadAndTestNetwork()
 {
  System.out.println("Testing the networks results");

  List<Double> xData = new ArrayList<Double>();
  List<Double> yData1 = new ArrayList<Double>();
```

```
List<Double> yData2 = new ArrayList<Double>();

double targetToPredictPercent = 0;
double maxGlobalResultDiff = 0.00;
double averGlobalResultDiff = 0.00;
double sumGlobalResultDiff = 0.00;
double normInputWordNumberFromRecord = 0.00;
double normTargetBookNumberFromRecord = 0.00;
double normPredictXPointValueFromRecord = 0.00;
BasicNetwork network;
maxGlobalResultDiff = 0.00;
averGlobalResultDiff = 0.00;
sumGlobalResultDiff = 0.00;

// Load the test dataset into memory
MLDataSet testingSet =
loadCSV2Memory(testFileName,numberOfInputNeurons,
numberOfOutputNeurons,true,CSVFormat.ENGLISH,false);

// Load the saved trained network
network =
 (BasicNetwork)EncogDirectoryPersistence.loadObject(new
 File(networkFileName));

int i = 0;

for (MLDataPair pair:  testingSet)
 {
      i++;

      MLData inputData = pair.getInput();
      MLData actualData = pair.getIdeal();
      MLData predictData = network.compute(inputData);

      // These values are Normalized as the whole input is
      normInputWordNumberFromRecord = inputData.getData(0);
      normTargetBookNumberFromRecord = actualData.getData(0);
      normPredictXPointValueFromRecord = predictData.getData(0);
```

```
denormInputWordNumber_01 = ((minWordNumberDl -maxWordNumberDh)*
normInputWordNumber_01 - Nh*minWordNumberDl + maxWordNumberDh
*Nl)/(Nl - Nh);

  denormInputWordNumber_02 = ((minWordNumberDl - maxWordNumberDh)*
  normInputWordNumber_02 - Nh*minWordNumberDl + maxWordNumberDh
  *Nl)/(Nl - Nh);

  denormInputWordNumber_03 = ((minWordNumberDl -
  maxWordNumberDh)*normInputWordNumber_03 - Nh*minWordNumberDl +
  maxWordNumberDh *Nl)/(Nl - Nh);

denormTargetBookNumber_01 = ((minTargetValueDl - maxTargetValueDh)*
normTargetBookNumber_01 - Nh*minTargetValueDl +
maxTargetValueDh*Nl)/(Nl - Nh);

  denormTargetBookNumber_02 = ((minTargetValueDl -
  maxTargetValueDh)*normTargetBookNumber_02 -
  Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

  denormTargetBookNumber_03 = ((minTargetValueDl -
  maxTargetValueDh)*normTargetBookNumber_03 - Nh*minTargetValueDl +
  maxTargetValueDh*Nl)/(Nl - Nh);

  denormTargetBookNumber_04 = ((minTargetValueDl - maxTarget
  ValueDh)*normTargetBookNumber_04 - Nh*minTargetValueDl +
  maxTargetValueDh*Nl)/(Nl - Nh);

  denormTargetBookNumber_05 = ((minTargetValueDl - maxTarget
  ValueDh)*normTargetBookNumber_05 - Nh*minTargetValueDl +
  maxTargetValueDh*Nl)/(Nl - Nh);

  denormPredictBookNumber_01 =((minTargetValueDl - maxTarget
  ValueDh)*normPredictBookNumber_01 - Nh*minTargetValueDl +
  maxTargetValueDh*Nl)/(Nl - Nh);

  denormPredictBookNumber_02 =((minTargetValueDl - maxTarget
  ValueDh)*normPredictBookNumber_02 -Nh*minTargetValueDl +
  maxTargetValueDh*Nl)/(Nl - Nh);
```

```
denormPredictBookNumber_03 =((minTargetValueDl - maxTarget
ValueDh)*normPredictBookNumber_03 - Nh*minTargetValueDl +
maxTargetValueDh*Nl)/(Nl - Nh);

denormPredictBookNumber_04 =((minTargetValueDl - maxTarget
ValueDh)*normPredictBookNumber_04 - Nh*minTargetValueDl +
maxTargetValueDh*Nl)/(Nl - Nh);

denormPredictBookNumber_05 =((minTargetValueDl - maxTarget
ValueDh)*normPredictBookNumber_05 -Nh*minTargetValueDl +
maxTargetValueDh*Nl)/(Nl - Nh);

System.out.println ("RecordNumber = " + i);

System.out.println ("denormTargetBookNumber_01 = " +
denormTargetBookNumber_01 + "denormPredictBookNumber_01 = " +
denormPredictBookNumber_01);

System.out.println ("denormTargetBookNumber_02 = " +
denormTargetBookNumber_02 + "denormPredictBookNumber_02 = " +
denormPredictBookNumber_02);

System.out.println ("denormTargetBookNumber_03 = " +
denormTargetBookNumber_03 + "denormPredictBookNumber_03 = " +
denormPredictBookNumber_03);

System.out.println ("denormTargetBookNumber_04 = " +
denormTargetBookNumber_04 + "denormPredictBookNumber_04 = " +
denormPredictBookNumber_04);

System.out.println ("denormTargetBookNumber_05 = " +
denormTargetBookNumber_05 + "denormPredictBookNumber_05 = " +
denormPredictBookNumber_05);

//System.out.println (" ");

if(Math.abs(denormPredictBookNumber_01) > 0.85  &
  Math.abs(denormPredictBookNumber_02) < 0.2   &
  Math.abs(denormPredictBookNumber_03) < 0.2   &
```

```
    Math.abs(denormPredictBookNumber_04) < 0.2   &
    Math.abs(denormPredictBookNumber_05) < 0.2
    ||
    Math.abs(denormPredictBookNumber_01) < 0.2  &
       Math.abs(denormPredictBookNumber_02) > 0.85 &
       Math.abs(denormPredictBookNumber_03) < 0.2  &
       Math.abs(denormPredictBookNumber_04) < 0.2  &
       Math.abs(denormPredictBookNumber_05) < 0.2
    |
   Math.abs(denormPredictBookNumber_01) < 0.2  &
       Math.abs(denormPredictBookNumber_02) > 0.85 &
       Math.abs(denormPredictBookNumber_03) < 0.2  &
       Math.abs(denormPredictBookNumber_04) < 0.2  &
       Math.abs(denormPredictBookNumber_05) < 0.2
  ||
    Math.abs(denormPredictBookNumber_01) < 0.2 &
       Math.abs(denormPredictBookNumber_02) < 0.2  &
       Math.abs(denormPredictBookNumber_03) > 0.85 &
       Math.abs(denormPredictBookNumber_04) < 0.2  &
       Math.abs(denormPredictBookNumber_05) < 0.2
    ||
   Math.abs(denormPredictBookNumber_01) < 0.2  &
       Math.abs(denormPredictBookNumber_02) < 0.2  &
       Math.abs(denormPredictBookNumber_03) < 0.2  &
       Math.abs(denormPredictBookNumber_04) > 0.85 &
       Math.abs(denormPredictBookNumber_05) < 0.2
   ||
  Math.abs(denormPredictBookNumber_01) < 0.2  &
       Math.abs(denormPredictBookNumber_02) < 0.2  &
       Math.abs(denormPredictBookNumber_03) < 0.2  &
       Math.abs(denormPredictBookNumber_04) < 0.2 &
       Math.abs(denormPredictBookNumber_05) > 0.85)
```

```
                {
                    System.out.println ("Record belong to some book");
                    System.out.println (" ");
                }
            else
             {
                 System.out.println ("Unknown book");
                 System.out.println (" ");
             }

        }  // End for pair loop

    } // End of the method

} // End of the class
```

Listing 10-4 shows the code fragment of the training method.

Listing 10-4. Code Fragment of the Training Method

```
static public int trainValidateSaveNetwork()
  {
      // Load the training CSV file in memory
      MLDataSet trainingSet =
        loadCSV2Memory(trainFileName,numberOfInputNeurons,
        numberOfOutputNeurons,true,CSVFormat.ENGLISH,false);

      // create a neural network
      BasicNetwork network = new BasicNetwork();

      // Input layer
      network.addLayer(new BasicLayer(null,true,3));

      // Hidden layer
      network.addLayer(new BasicLayer(new ActivationTANH(),true,7));
      network.addLayer(new BasicLayer(new ActivationTANH(),true,7));
      network.addLayer(new BasicLayer(new ActivationTANH(),true,7));
      network.addLayer(new BasicLayer(new ActivationTANH(),true,7));
```

```
network.addLayer(new BasicLayer(new ActivationTANH(),true,7));
network.addLayer(new BasicLayer(new ActivationTANH(),true,7));

// Output layer
network.addLayer(new BasicLayer(new ActivationTANH(),false,5));

network.getStructure().finalizeStructure();
network.reset();

//Train the neural network
final ResilientPropagation train = new ResilientPropagation(network,
trainingSet);

int epoch = 1;

do
 {
      train.iteration();
      System.out.println("Epoch #" + epoch + " Error:" +
      train.getError());

    epoch++;

   if (epoch >= 1000 && network.calculateError(trainingSet) >
   0.0000000000000012)
       {
        returnCode = 1;
        System.out.println("Try again");
        return returnCode;
       }

} while (network.calculateError(trainingSet) > 0.0000000000000011);

// Save the network file
EncogDirectoryPersistence.saveObject(new File(networkFileName),network);

System.out.println("Neural Network Results:");

double sumNormDifferencePerc = 0.00;
double averNormDifferencePerc = 0.00;
double maxNormDifferencePerc = 0.00;
```

```
int m = 0;

for(MLDataPair pair: trainingSet)
  {
        m++;

       final MLData output = network.compute(pair.getInput());

       MLData inputData = pair.getInput();
       MLData actualData = pair.getIdeal();
       MLData predictData = network.compute(inputData);

       // Calculate and print the results

       normInputWordNumber_01 = inputData.getData(0);
       normInputWordNumber_02 = inputData.getData(1);
       normInputWordNumber_03 = inputData.getData(2);

       normTargetBookNumber_01 = actualData.getData(0);
       normTargetBookNumber_02 = actualData.getData(1);
       normTargetBookNumber_03 = actualData.getData(2);
       normTargetBookNumber_04 = actualData.getData(3);
       normTargetBookNumber_05 = actualData.getData(4);

       normPredictBookNumber_01 = predictData.getData(0);
       normPredictBookNumber_02 = predictData.getData(1);
       normPredictBookNumber_03 = predictData.getData(2);
       normPredictBookNumber_04 = predictData.getData(3);
       normPredictBookNumber_05 = predictData.getData(4);

       denormInputWordNumber_01 = ((minWordNumberDl -maxWordNumberDh)*
       normInputWordNumber_01 - Nh*minWordNumberDl +
       maxWordNumberDh *Nl)/(Nl - Nh);

       denormInputWordNumber_02 = ((minWordNumberDl -maxWordNumberDh)*
       normInputWordNumber_02 - Nh*minWordNumberDl +
       maxWordNumberDh *Nl)/(Nl - Nh);

       denormInputWordNumber_03 = ((minWordNumberDl -maxWordNumberDh)*
       normInputWordNumber_03 - Nh*minWordNumberDl +
       maxWordNumberDh *Nl)/(Nl - Nh);
```

```
denormTargetBookNumber_01 = ((minTargetValueDl -
maxTargetValueDh)*normTargetBookNumber_01 -
Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

denormTargetBookNumber_02 = ((minTargetValueDl -
maxTargetValueDh)*normTargetBookNumber_02 -
Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

denormTargetBookNumber_03 = ((minTargetValueDl -
maxTargetValueDh)*normTargetBookNumber_03 -
Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

denormTargetBookNumber_04 = ((minTargetValueDl -
maxTargetValueDh)*normTargetBookNumber_04 -
Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

denormTargetBookNumber_05 = ((minTargetValueDl -
maxTargetValueDh)*normTargetBookNumber_05 -
Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

denormPredictBookNumber_01 =((minTargetValueDl -
maxTargetValueDh)*normPredictBookNumber_01 -
Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

denormPredictBookNumber_02 =((minTargetValueDl -
maxTargetValueDh)*normPredictBookNumber_02 -
Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

denormPredictBookNumber_03 =((minTargetValueDl -
maxTargetValueDh)*normPredictBookNumber_03 -
Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

denormPredictBookNumber_04 =((minTargetValueDl -
maxTargetValueDh)*normPredictBookNumber_04 -
Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

denormPredictBookNumber_05 =((minTargetValueDl -
maxTargetValueDh)*normPredictBookNumber_05 -
Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);
```

```
 System.out.println ("RecordNumber = " + m);

 System.out.println ("denormTargetBookNumber_01 = " +
 denormTargetBookNumber_01 + "denormPredictBookNumber_01 = " +
 denormPredictBookNumber_01);

 System.out.println ("denormTargetBookNumber_02 = " +
 denormTargetBookNumber_02 + "denormPredictBookNumber_02 = " +
 denormPredictBookNumber_02);

 System.out.println ("denormTargetBookNumber_03 = " +
 denormTargetBookNumber_03 + "denormPredictBookNumber_03 = " +
 denormPredictBookNumber_03);

 System.out.println ("denormTargetBookNumber_04 = " +
 denormTargetBookNumber_04 + "denormPredictBookNumber_04 = " +
 denormPredictBookNumber_04);

 System.out.println ("denormTargetBookNumber_05 = " +
 denormTargetBookNumber_05 + "denormPredictBookNumber_05 = " +
 denormPredictBookNumber_05);

//System.out.println (" ");

// Print the classification results in the log
if(Math.abs(denormPredictBookNumber_01) > 0.85)
  if(Math.abs(denormPredictBookNumber_01) > 0.85  &
     Math.abs(denormPredictBookNumber_02) < 0.2   &
     Math.abs(denormPredictBookNumber_03) < 0.2   &
     Math.abs(denormPredictBookNumber_04) < 0.2   &
     Math.abs(denormPredictBookNumber_05) < 0.2)
      {
        System.out.println ("Record 1 belongs to book 1");
        System.out.println (" ");
      }
  else
   {
     System.out.println ("Wrong results for record 1");
     System.out.println (" ");
   }
```

```
if(Math.abs(denormPredictBookNumber_02) > 0.85)
 if(Math.abs(denormPredictBookNumber_01) < 0.2  &
    Math.abs(denormPredictBookNumber_02) > 0.85 &
    Math.abs(denormPredictBookNumber_03) < 0.2  &
    Math.abs(denormPredictBookNumber_04) < 0.2  &
    Math.abs(denormPredictBookNumber_05) < 0.2)
     {
       System.out.println ("Record 2 belongs to book 2");
       System.out.println (" ");
     }
  else
   {
     System.out.println ("Wrong results for record 2");
     System.out.println (" ");
   }

if(Math.abs(denormPredictBookNumber_03) > 0.85)
 if(Math.abs(denormPredictBookNumber_01) < 0.2 &
    Math.abs(denormPredictBookNumber_02) < 0.2  &
    Math.abs(denormPredictBookNumber_03) > 0.85 &
    Math.abs(denormPredictBookNumber_04) < 0.2  &
    Math.abs(denormPredictBookNumber_05) < 0.2)
     {
       System.out.println ("Record 3 belongs to book 3");
       System.out.println (" ");
     }
 else
  {
    System.out.println ("Wrong results for record 3");
    System.out.println (" ");
  }

if(Math.abs(denormPredictBookNumber_04) > 0.85)
 if(Math.abs(denormPredictBookNumber_01) < 0.2  &
    Math.abs(denormPredictBookNumber_02) < 0.2  &
    Math.abs(denormPredictBookNumber_03) < 0.2  &
    Math.abs(denormPredictBookNumber_04) > 0.85 &
```

```
            Math.abs(denormPredictBookNumber_05) < 0.2)
             {
               System.out.println ("Record 4 belongs to book 4");
               System.out.println (" ");
             }
          else
           {
             System.out.println ("Wrong results for record 4");
             System.out.println (" ");
           }
         if(Math.abs(denormPredictBookNumber_05) > 0.85)
          if(Math.abs(denormPredictBookNumber_01) < 0.2  &
             Math.abs(denormPredictBookNumber_02) < 0.2  &
             Math.abs(denormPredictBookNumber_03) < 0.2  &
             Math.abs(denormPredictBookNumber_04) < 0.2 &
             Math.abs(denormPredictBookNumber_05) > 0.85)
              {
               System.out.println ("Record 5 belongs to book 5");
               System.out.println (" ");
              }
          else
           {
             System.out.println ("Wrong results for record 5");
             System.out.println (" ");
           }

    }   // End for pair loop

    returnCode = 0;
    return returnCode;

 }   // End of the method
```

Listing 10-5 shows the code fragment of the testing method.

Here, you load the test data set and the previously saved trained network in memory. Next, you loop over the pair data set and retrieve for each record three input book numbers and five target book numbers. You denormalize the obtained values and then check whether the record belongs to one of the five books.

Listing 10-5. Code Fragment of the Testing Method

```
 // Load the test dataset into memory
 MLDataSet testingSet =
     loadCSV2Memory(testFileName,numberOfInputNeurons,numberOfOutputNeurons,
       true,CSVFormat.ENGLISH,false);

 // Load the saved trained network
 network =
      (BasicNetwork)EncogDirectoryPersistence.loadObject(new
File(networkFileName));

int i = 0;

for (MLDataPair pair:  testingSet)
    {
           i++;

           MLData inputData = pair.getInput();
           MLData actualData = pair.getIdeal();
           MLData predictData = network.compute(inputData);

           // These values are Normalized as the whole input is
           normInputWordNumberFromRecord = inputData.getData(0);
           normTargetBookNumberFromRecord = actualData.getData(0);
           normPredictXPointValueFromRecord = predictData.getData(0);

           denormInputWordNumber_01 = ((minWordNumberDl -maxWordNumberDh)*
           normInputWordNumber_01 - Nh*minWordNumberDl +
           maxWordNumberDh *Nl)/(Nl - Nh);

             denormInputWordNumber_02 = ((minWordNumberDl -
             maxWordNumberDh)*normInputWordNumber_02 - Nh*minWordNumberDl +
             maxWordNumberDh *Nl)/(Nl - Nh);

             denormInputWordNumber_03 = ((minWordNumberDl -
             maxWordNumberDh)*normInputWordNumber_03 - Nh*minWordNumberDl +
             maxWordNumberDh *Nl)/(Nl - Nh);
```

```
denormTargetBookNumber_01 = ((minTargetValueDl -
maxTargetValueDh)*normTargetBookNumber_01 -
Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

  denormTargetBookNumber_02 = ((minTargetValueDl -
  maxTargetValueDh)*normTargetBookNumber_02 -
  Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

  denormTargetBookNumber_03 = ((minTargetValueDl -
  maxTargetValueDh)*normTargetBookNumber_03 -
  Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

  denormTargetBookNumber_04 = ((minTargetValueDl -
  maxTargetValueDh)*normTargetBookNumber_04 -
  Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

  denormTargetBookNumber_05 = ((minTargetValueDl -
  maxTargetValueDh)*normTargetBookNumber_05 -
  Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

  denormPredictBookNumber_01 =((minTargetValueDl -
  maxTargetValueDh)*normPredictBookNumber_01 -
  Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

  denormPredictBookNumber_02 =((minTargetValueDl -
  maxTargetValueDh)*normPredictBookNumber_02 -
  Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

  denormPredictBookNumber_03 =((minTargetValueDl -
  maxTargetValueDh)*normPredictBookNumber_03 -
  Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

  denormPredictBookNumber_04 =((minTargetValueDl -
  maxTargetValueDh)*normPredictBookNumber_04 -
  Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

  denormPredictBookNumber_05 =((minTargetValueDl -
  maxTargetValueDh)*normPredictBookNumber_05 -
  Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);
```

```
System.out.println ("RecordNumber = " + i);

System.out.println ("denormTargetBookNumber_01 = " +
denormTargetBookNumber_01 + "denormPredictBookNumber_01 = " +
denormPredictBookNumber_01);

System.out.println ("denormTargetBookNumber_02 = " +
denormTargetBookNumber_02 + "denormPredictBookNumber_02 = " +
denormPredictBookNumber_02);

System.out.println ("denormTargetBookNumber_03 = " +
denormTargetBookNumber_03 + "denormPredictBookNumber_03 = " +
denormPredictBookNumber_03);

System.out.println ("denormTargetBookNumber_04 = " +
denormTargetBookNumber_04 + "denormPredictBookNumber_04 = " +
denormPredictBookNumber_04);

System.out.println ("denormTargetBookNumber_05 = " +
denormTargetBookNumber_05 + "denormPredictBookNumber_05 = " +
denormPredictBookNumber_05);

//System.out.println (" ");

if(Math.abs(denormPredictBookNumber_01) > 0.85  &
  Math.abs(denormPredictBookNumber_02) < 0.2   &
  Math.abs(denormPredictBookNumber_03) < 0.2   &
  Math.abs(denormPredictBookNumber_04) < 0.2   &
  Math.abs(denormPredictBookNumber_05) < 0.2
  |
  Math.abs(denormPredictBookNumber_01) < 0.2  &
     Math.abs(denormPredictBookNumber_02) > 0.85 &
     Math.abs(denormPredictBookNumber_03) < 0.2  &
     Math.abs(denormPredictBookNumber_04) < 0.2  &
     Math.abs(denormPredictBookNumber_05) < 0.2

 |
 Math.abs(denormPredictBookNumber_01) < 0.2  &
     Math.abs(denormPredictBookNumber_02) > 0.85 &
     Math.abs(denormPredictBookNumber_03) < 0.2  &
```

```
                Math.abs(denormPredictBookNumber_04) < 0.2  &
                Math.abs(denormPredictBookNumber_05) < 0.2
              |
               Math.abs(denormPredictBookNumber_01) < 0.2 &
                Math.abs(denormPredictBookNumber_02) < 0.2  &
                Math.abs(denormPredictBookNumber_03) > 0.85 &
                Math.abs(denormPredictBookNumber_04) < 0.2  &
                Math.abs(denormPredictBookNumber_05) < 0.2
             |
              Math.abs(denormPredictBookNumber_01) < 0.2  &
                Math.abs(denormPredictBookNumber_02) < 0.2  &
                Math.abs(denormPredictBookNumber_03) < 0.2  &
                Math.abs(denormPredictBookNumber_04) > 0.85 &
                Math.abs(denormPredictBookNumber_05) < 0.2
            |
            Math.abs(denormPredictBookNumber_01) < 0.2  &
                Math.abs(denormPredictBookNumber_02) < 0.2  &
                Math.abs(denormPredictBookNumber_03) < 0.2  &
                Math.abs(denormPredictBookNumber_04) < 0.2 &
                Math.abs(denormPredictBookNumber_05) > 0.85)
                {
                   System.out.println ("Record belong to some book");
                   System.out.println (" ");
                }
            else
             {
                System.out.println ("Unknown book");
                System.out.println (" ");
             }
    }  // End for pair loop

} // End of the method
```

Training Results

Listing 10-6 shows the training/validation results.

Listing 10-6. Training/Validation Results

```
RecordNumber = 1
denormTargetBookNumber_01 = 1.0   denormPredictBookNumber_01 = 1.0
denormTargetBookNumber_02 = -0.0  denormPredictBookNumber_02 =
3.6221384780432686E-9
denormTargetBookNumber_03 = -0.0  denormPredictBookNumber_03 = -0.0
denormTargetBookNumber_04 = -0.0  denormPredictBookNumber_04 =
1.3178162894256218E-8
denormTargetBookNumber_05 = -0.0  denormPredictBookNumber_05 =
2.220446049250313E-16
    Record 1 belongs to book 1

RecordNumber = 2
denormTargetBookNumber_01 = 1.0   denormPredictBookNumber_01 = 1.0
denormTargetBookNumber_02 = -0.0  denormPredictBookNumber_02 =
3.6687665128098956E-9
denormTargetBookNumber_03 = -0.0  denormPredictBookNumber_03 = -0.0
denormTargetBookNumber_04 = -0.0  denormPredictBookNumber_04 =
1.0430401597982808E-8
denormTargetBookNumber_05 = -0.0  denormPredictBookNumber_05 =
2.220446049250313E-16
    Record 1 belongs to book 1

RecordNumber = 3
denormTargetBookNumber_01 = 1.0   denormPredictBookNumber_01 = 1.0
denormTargetBookNumber_02 = -0.0  denormPredictBookNumber_02 =
4.35402175424926E-9
denormTargetBookNumber_03 = -0.0  denormPredictBookNumber_03 = -0.0
denormTargetBookNumber_04 = -0.0  denormPredictBookNumber_04 =
9.684705759571699E-9
denormTargetBookNumber_05 = -0.0  denormPredictBookNumber_05 =
2.220446049250313E-16
```

```
    Record 1 belongs to book 1

RecordNumber = 4
denormTargetBookNumber_01 = 1.0   denormPredictBookNumber_01 = 1.0
denormTargetBookNumber_02 = -0.0  denormPredictBookNumber_02 =
6.477930192261283E-9
denormTargetBookNumber_03 = -0.0  denormPredictBookNumber_03 = -0.0
denormTargetBookNumber_04 = -0.0  denormPredictBookNumber_04 =
4.863816960298806E-9
denormTargetBookNumber_05 = -0.0  denormPredictBookNumber_05 =
2.220446049250313E-16
    Record 1 belongs to book 1

RecordNumber = 5
denormTargetBookNumber_01 = 1.0   denormPredictBookNumber_01 = 1.0
denormTargetBookNumber_02 = -0.0  denormPredictBookNumber_02 =
1.7098276960947345E-8
denormTargetBookNumber_03 = -0.0  denormPredictBookNumber_03 = -0.0
denormTargetBookNumber_04 = -0.0  denormPredictBookNumber_04 =
4.196660130517671E-9
denormTargetBookNumber_05 = -0.0  denormPredictBookNumber_05 =
2.220446049250313E-16
    Record 1 belongs to book 1

RecordNumber = 6
denormTargetBookNumber_01 = 1.0   denormPredictBookNumber_01 = 1.0
denormTargetBookNumber_02 = -0.0  denormPredictBookNumber_02 =
9.261896322110275E-8
denormTargetBookNumber_03 = -0.0  denormPredictBookNumber_03 = -0.0
denormTargetBookNumber_04 = -0.0  denormPredictBookNumber_04 =
2.6307949707593536E-9
denormTargetBookNumber_05 = -0.0  denormPredictBookNumber_05 =
2.7755575615628914E-16
    Record 1 belongs to book 1
```

```
RecordNumber = 7
denormTargetBookNumber_01 = -0.0  denormPredictBookNumber_01 =
5.686340287525127E-12
denormTargetBookNumber_02 = 1.0   denormPredictBookNumber_02 =
0.9999999586267019
denormTargetBookNumber_03 = -0.0  denormPredictBookNumber_03 = -0.0
denormTargetBookNumber_04 = -0.0  denormPredictBookNumber_04 =
1.1329661653292078E-9
denormTargetBookNumber_05 = -0.0  denormPredictBookNumber_05 =
9.43689570931383E-16
    Record 2 belongs to book 2

RecordNumber = 8
denormTargetBookNumber_01 = -0.0  denormPredictBookNumber_01 = -0.0
denormTargetBookNumber_02 = 1.0   denormPredictBookNumber_02 =
0.9999999999998506
denormTargetBookNumber_03 = -0.0  denormPredictBookNumber_03 = -0.0
denormTargetBookNumber_04 = -0.0  denormPredictBookNumber_04 =
1.091398971198032E-9
denormTargetBookNumber_05 = -0.0  denormPredictBookNumber_05 =
2.6645352591003757E-15
    Record 2 belongs to book 2

RecordNumber = 9
denormTargetBookNumber_01 = -0.0  denormPredictBookNumber_01 = -0.0
denormTargetBookNumber_02 = 1.0   denormPredictBookNumber_02 =
0.9999999999999962
denormTargetBookNumber_03 = -0.0  denormPredictBookNumber_03 = -0.0
denormTargetBookNumber_04 = -0.0  denormPredictBookNumber_04 =
1.0686406759496947E-9
denormTargetBookNumber_05 = -0.0  denormPredictBookNumber_05 =
3.7192471324942744E-15
```

```
    Record 2 belongs to book 2

RecordNumber = 10
denormTargetBookNumber_01 = -0.0  denormPredictBookNumber_01 = -0.0
denormTargetBookNumber_02 = 1.0   denormPredictBookNumber_02 =
0.9999999999999798
denormTargetBookNumber_03 = -0.0  denormPredictBookNumber_03 =
2.2352120154778277E-12
denormTargetBookNumber_04 = -0.0  denormPredictBookNumber_04 =
7.627692921730045E-10
denormTargetBookNumber_05 = -0.0  denormPredictBookNumber_05 =
1.9817480989559044E-14
    Record 2 belongs to book 2

RecordNumber = 11
denormTargetBookNumber_01 = -0.0  denormPredictBookNumber_01 = -0.0
denormTargetBookNumber_02 = 1.0   denormPredictBookNumber_02 =
0.9999999999999603
denormTargetBookNumber_03 = -0.0  denormPredictBookNumber_03 =
1.2451872866137137E-11
denormTargetBookNumber_04 = -0.0  denormPredictBookNumber_04 =
7.404629132068408E-10
denormTargetBookNumber_05 = -0.0  denormPredictBookNumber_05 =
2.298161660974074E-14
    Record 2 belongs to book 2

RecordNumber = 12
denormTargetBookNumber_01 = -0.0  denormPredictBookNumber_01 = -0.0
denormTargetBookNumber_02 = 1.0   denormPredictBookNumber_02 =
0.9999999999856213
denormTargetBookNumber_03 = -0.0  denormPredictBookNumber_03 =
7.48775297876314E-8
denormTargetBookNumber_04 = -0.0  denormPredictBookNumber_04 =
6.947271091739537E-10
denormTargetBookNumber_05 = -0.0  denormPredictBookNumber_05 =
4.801714581503802E-14
    Record 2 belongs to book 2
```

```
RecordNumber = 13
denormTargetBookNumber_01 = -0.0  denormPredictBookNumber_01 = -0.0
denormTargetBookNumber_02 = -0.0  denormPredictBookNumber_02 =
7.471272545078733E-9
denormTargetBookNumber_03 = 1.0   denormPredictBookNumber_03 =
0.9999999419988991
denormTargetBookNumber_04 = -0.0  denormPredictBookNumber_04 =
2.5249974888730264E-9
denormTargetBookNumber_05 = -0.0  denormPredictBookNumber_05 =
2.027711332175386E-12
    Record 3 belongs to book 3

RecordNumber = 14
denormTargetBookNumber_01 = -0.0  denormPredictBookNumber_01 = -0.0
denormTargetBookNumber_02 = -0.0  denormPredictBookNumber_02 =
2.295386103412511E-13
denormTargetBookNumber_03 = 1.0   denormPredictBookNumber_03 =
0.9999999999379154
denormTargetBookNumber_04 = -0.0  denormPredictBookNumber_04 =
4.873732140087128E-9
denormTargetBookNumber_05 = -0.0  denormPredictBookNumber_05 =
4.987454893523591E-12
Record 3 belongs to book 3

RecordNumber = 15
denormTargetBookNumber_01 = -0.0  denormPredictBookNumber_01 = -0.0
denormTargetBookNumber_02 = -0.0  denormPredictBookNumber_02 =
2.692845946228317E-13
denormTargetBookNumber_03 = 1.0   denormPredictBookNumber_03 =
0.9999999998630087
denormTargetBookNumber_04 = -0.0  denormPredictBookNumber_04 =
4.701179112664988E-9
denormTargetBookNumber_05 = -0.0  denormPredictBookNumber_05 =
4.707678691318051E-12
    Record 3 belongs to book 3
```

```
RecordNumber = 16
denormTargetBookNumber_01 = -0.0  denormPredictBookNumber_01 = -0.0
denormTargetBookNumber_02 = -0.0  denormPredictBookNumber_02 = -0.0
denormTargetBookNumber_03 = 1.0   denormPredictBookNumber_03 =
0.9999999999999996
denormTargetBookNumber_04 = -0.0  denormPredictBookNumber_04 =
2.0469307360215794E-8
denormTargetBookNumber_05 = -0.0  denormPredictBookNumber_05 =
2.843247859374287E-11
     Record 3 belongs to book 3

RecordNumber = 17
denormTargetBookNumber_01 = -0.0  denormPredictBookNumber_01 = -0.0
denormTargetBookNumber_02 = -0.0  denormPredictBookNumber_02 = -0.0
denormTargetBookNumber_03 = 1.0   denormPredictBookNumber_03 =
0.9999999999999987
denormTargetBookNumber_04 = -0.0  denormPredictBookNumber_04 =
1.977055869017974E-8
denormTargetBookNumber_05 = -0.0  denormPredictBookNumber_05 =
2.68162714256448E-11
     Record 3 belongs to book 3

RecordNumber = 18
denormTargetBookNumber_01 = -0.0  denormPredictBookNumber_01 = -0.0
denormTargetBookNumber_02 = -0.0  denormPredictBookNumber_02 = -0.0
denormTargetBookNumber_03 = 1.0   denormPredictBookNumber_03 =
0.9999999885142061
denormTargetBookNumber_04 = -0.0  denormPredictBookNumber_04 =
2.6820915488556807E-8
denormTargetBookNumber_05 = -0.0  denormPredictBookNumber_05 =
7.056188966458876E-12
    Record 3 belongs to book 3
```

```
RecordNumber = 19
denormTargetBookNumber_01 = -0.0  denormPredictBookNumber_01 = -0.0
denormTargetBookNumber_02 = -0.0  denormPredictBookNumber_02 = -0.0
denormTargetBookNumber_03 = -0.0  denormPredictBookNumber_03 =
2.983344798979104E-8
denormTargetBookNumber_04 = 1.0   denormPredictBookNumber_04 =
0.9999999789933758
denormTargetBookNumber_05 = -0.0  denormPredictBookNumber_05 =
1.7987472622493783E-10
    Record 4 belongs to book 4

RecordNumber = 20
denormTargetBookNumber_01 = -0.0  denormPredictBookNumber_01 = -0.0
denormTargetBookNumber_02 = -0.0  denormPredictBookNumber_02 = -0.0
denormTargetBookNumber_03 = -0.0  denormPredictBookNumber_03 =
1.0003242317813132E-7
denormTargetBookNumber_04 = 1.0   denormPredictBookNumber_04 =
0.9999999812213116
denormTargetBookNumber_05 = -0.0  denormPredictBookNumber_05 =
2.2566659652056842E-10
    Record 4 belongs to book 4

RecordNumber = 21
denormTargetBookNumber_01 = -0.0  denormPredictBookNumber_01 = -0.0
denormTargetBookNumber_02 = -0.0  denormPredictBookNumber_02 = -0.0
denormTargetBookNumber_03 = -0.0  denormPredictBookNumber_03 =
1.4262971415046621E-8
denormTargetBookNumber_04 = 1.0   denormPredictBookNumber_04 =
0.9999999812440078
denormTargetBookNumber_05 = -0.0  denormPredictBookNumber_05 =
2.079504346497174E-10
    Record 4 belongs to book 4
```

```
RecordNumber = 22
denormTargetBookNumber_01 = -0.0  denormPredictBookNumber_01 = -0.0
denormTargetBookNumber_02 = -0.0  denormPredictBookNumber_02 = -0.0
denormTargetBookNumber_03 = -0.0  denormPredictBookNumber_03 =
5.790115659154438E-8
denormTargetBookNumber_04 = 1.0   denormPredictBookNumber_04 =
0.9999999845075942
denormTargetBookNumber_05 = -0.0  denormPredictBookNumber_05 =
2.9504404475133583E-10
    Record 4 belongs to book 4

RecordNumber = 23
denormTargetBookNumber_01 = -0.0  denormPredictBookNumber_01 = -0.0
denormTargetBookNumber_02 = -0.0  denormPredictBookNumber_02 = -0.0
denormTargetBookNumber_03 = -0.0  denormPredictBookNumber_03 =
6.890162551620449E-9
denormTargetBookNumber_04 = 1.0   denormPredictBookNumber_04 =
0.999999984526581
denormTargetBookNumber_05 = -0.0  denormPredictBookNumber_05 =
2.6966767707747863E-10
    Record 4 belongs to book 4

RecordNumber = 24
denormTargetBookNumber_01 = -0.0  denormPredictBookNumber_01 = -0.0
denormTargetBookNumber_02 = -0.0  denormPredictBookNumber_02 = -0.0
denormTargetBookNumber_03 = -0.0  denormPredictBookNumber_03 =
9.975842318876715E-9
denormTargetBookNumber_04 = 1.0   denormPredictBookNumber_04 =
0.9999999856956441
denormTargetBookNumber_05 = -0.0  denormPredictBookNumber_05 =
3.077177401777931E-10
    Record 4 belongs to book 4
```

```
RecordNumber = 25
denormTargetBookNumber_01 = -0.0  denormPredictBookNumber_01 = -0.0
denormTargetBookNumber_02 = -0.0  denormPredictBookNumber_02 = -0.0
denormTargetBookNumber_03 = -0.0  denormPredictBookNumber_03 =
3.569367024169878E-14
denormTargetBookNumber_04 = -0.0  denormPredictBookNumber_04 =
1.8838704707313525E-8
denormTargetBookNumber_05 = 1.0   denormPredictBookNumber_05 =
0.9999999996959972
    Record 5 belongs to book 5

RecordNumber = 26
denormTargetBookNumber_01 = -0.0  denormPredictBookNumber_01 = -0.0
denormTargetBookNumber_02 = -0.0  denormPredictBookNumber_02 = -0.0
denormTargetBookNumber_03 = -0.0  denormPredictBookNumber_03 =
4.929390229335695E-14
denormTargetBookNumber_04 = -0.0  denormPredictBookNumber_04 =
1.943621164013365E-8
denormTargetBookNumber_05 = 1.0   denormPredictBookNumber_05 =
0.9999999997119369
    Record 5 belongs to book 5

RecordNumber = 27
denormTargetBookNumber_01 = -0.0  denormPredictBookNumber_01 = -0.0
denormTargetBookNumber_02 = -0.0  denormPredictBookNumber_02 = -0.0
denormTargetBookNumber_03 = -0.0  denormPredictBookNumber_03 =
1.532107773982716E-14
denormTargetBookNumber_04 = -0.0  denormPredictBookNumber_04 =
1.926626319592728E-8
denormTargetBookNumber_05 = 1.0   denormPredictBookNumber_05 =
0.9999999996935514
    Record 5 belongs to book 5
```

```
RecordNumber = 28
denormTargetBookNumber_01 = -0.0  denormPredictBookNumber_01 = -0.0
denormTargetBookNumber_02 = -0.0  denormPredictBookNumber_02 = -0.0
denormTargetBookNumber_03 = -0.0  denormPredictBookNumber_03 =
3.2862601528904634E-14
denormTargetBookNumber_04 = -0.0  denormPredictBookNumber_04 =
2.034116280968945E-8
denormTargetBookNumber_05 = 1.0   denormPredictBookNumber_05 =
0.9999999997226772
    Record 5 belongs to book 5

RecordNumber = 29
denormTargetBookNumber_01 = -0.0  denormPredictBookNumber_01 = -0.0
denormTargetBookNumber_02 = -0.0  denormPredictBookNumber_02 = -0.0
denormTargetBookNumber_03 = -0.0  denormPredictBookNumber_03 =
1.27675647831893E-14
denormTargetBookNumber_04 = -0.0  denormPredictBookNumber_04 =
2.014738198496957E-8
denormTargetBookNumber_05 = 1.0   denormPredictBookNumber_05 =
0.9999999997076233
    Record 5 belongs to book 5

RecordNumber = 30
denormTargetBookNumber_01 = -0.0  denormPredictBookNumber_01 = -0.0
denormTargetBookNumber_02 = -0.0  denormPredictBookNumber_02 = -0.0
denormTargetBookNumber_03 = -0.0  denormPredictBookNumber_03 =
2.0039525594484076E-14
denormTargetBookNumber_04 = -0.0  denormPredictBookNumber_04 =
2.0630209485172912E-8
denormTargetBookNumber_05 = 1.0  denormPredictBookNumber_05 =
0.9999999997212032
    Record 5 belongs to book 5
```

As shown in the log, the program correctly identified the book numbers to which all the records belong .

Testing Results

Listing 10-7 shows the testing results.

Listing 10-7. Testing Results

```
RecordNumber = 1
   Unknown book

RecordNumber = 2
   Unknown book

RecordNumber = 3
   Unknown book

RecordNumber = 4
   Unknown book

RecordNumber = 5
   Unknown book

RecordNumber = 6
   Unknown book

RecordNumber = 7
   Unknown book

RecordNumber = 8
   Unknown book

RecordNumber = 9
   Unknown book

RecordNumber = 10
   Unknown book
RecordNumber = 11
   Unknown book

RecordNumber = 12
   Unknown book
```

```
RecordNumber = 13
   Unknown book

RecordNumber = 14
   Unknown book

RecordNumber = 15
   Unknown book
```

The testing process correctly classified the objects by determining that all the processed records don't belong to any of the five books.

Summary

The chapter explained how to use neural networks to classify objects. Specifically, the example in this chapter showed how a neural network was able to determine to which book each testing record belongs. In the next chapter, you will learn the importance of selecting the correct processing model.

CHAPTER 11

The Importance of Selecting the Correct Model

The example discussed in this chapter will end up showing a negative result. However, you can learn a lot from mistakes like this.

Example 7: Predicting Next Month's Stock Market Price

In this example, you will try to predict next month's price of the SPY exchange-traded fund (ETF); this is the ETF that mimics the S&P 500 stock market index. Someone's rational for developing such a project could be something like this:

> *"We know that market prices are random, jumping daily up and down and reacting to different news. However, we are using the monthly prices, which tend to be more stable. In addition, the market often experiences conditions that are similar to past situations, so people (in general) should react approximately the same with the same conditions. Therefore, by knowing how the market reacted in the past, we should be able to closely predict the market behavior for the next month."*

In this example, you will use the ten-year historic monthly prices for the SPY ETF and will attempt to predict next month's price. Of course, using the historical SPY data from a longer duration would positively contribute to the accuracy of the prediction; however, this is an example, so let's keep it reasonable small. The input data set contains data for

I. Livshin, *Artificial Neural Networks with Java*, https://doi.org/10.1007/978-1-4842-4421-0_11

ten years (120 months), from January 2000 until January 2009, and you want to predict the SPY price at the end of February 2009. Table 11-1 shows the historical monthly SPY prices for that period.

Table 11-1. *Historical Monthly SPY ETF Prices*

Date	Price	Date	Price
200001	1394.46	200501	1181.27
200002	1366.42	200502	1203.6
200003	1498.58	200503	1180.59
200004	1452.43	200504	1156.85
200005	1420.6	200505	1191.5
200006	1454.6	200506	1191.33
200007	1430.83	200507	1234.18
200008	1517.68	200508	1220.33
200009	1436.51	200509	1228.81
200010	1429.4	200510	1207.01
200011	1314.95	200511	1249.48
200012	1320.28	200512	1248.29
200101	1366.01	200601	1280.08
200102	1239.94	200602	1280.66
200103	1160.33	200603	1294.87
200104	1249.46	200604	1310.61
200105	1255.82	200605	1270.09
200106	1224.38	200606	1270.2
200107	1211.23	200607	1276.66
200108	1133.58	200608	1303.82
200109	1040.94	200609	1335.85
200110	1059.78	200610	1377.94

(*continued*)

Table 11-1. (*continued*)

Date	Price	Date	Price
200111	1139.45	200611	1400.63
200112	1148.08	200612	1418.3
200201	1130.2	200701	1438.24
200202	1106.73	200702	1406.82
200203	1147.39	200703	1420.86
200204	1076.92	200704	1482.37
200205	1067.14	200705	1530.62
200206	989.82	200706	1503.35
200207	911.62	200707	1455.27
200208	916.07	200708	1473.99
200209	815.28	200709	1526.75
200210	885.76	200710	1549.38
200211	936.31	200711	1481.14
200212	879.82	200712	1468.36
200301	855.7	200801	1378.55
200302	841.15	200802	1330.63
200303	848.18	200803	1322.7
200304	916.92	200804	1385.59
200305	963.59	200805	1400.38
200306	974.5	200806	1280
200307	990.31	200807	1267.38
200308	1008.01	200808	1282.83
200309	995.97	200809	1166.36
200310	1050.71	200810	968.75
200311	1058.2	200811	896.24

(*continued*)

Table 11-1. (*continued*)

Date	Price	Date	Price
200312	1111.92	200812	903.25
200401	1131.13	200901	825.88
200402	1144.94	200902	735.09
200403	1126.21	200903	797.87
200404	1107.3	200904	872.81
200405	1120.68	200905	919.14
200406	1140.84	200906	919.32
200407	1101.72	200907	987.48
200408	1104.24	200908	1020.62
200409	1114.58	200909	1057.08
200410	1130.2	200910	1036.19
200411	1173.82	200911	1095.63
200412	1211.92	200912	1115.1

Figure 11-1 shows the chart of the historical monthly SPY prices.

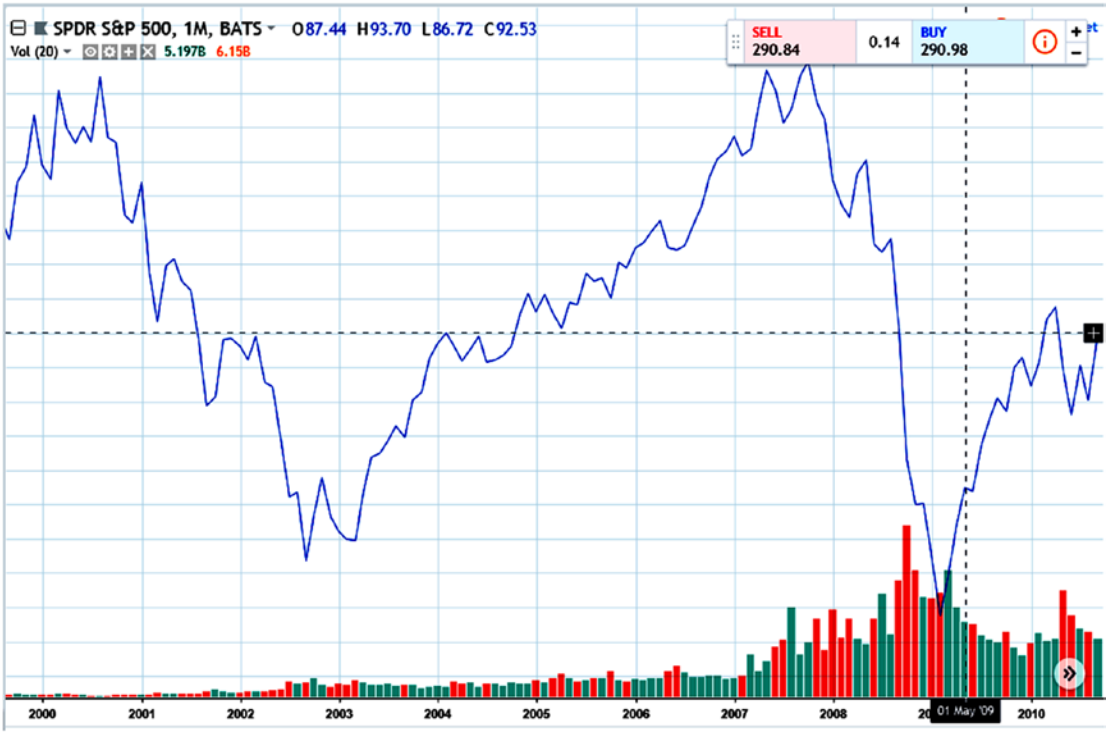


Figure 11-1. *SPY monthly chart for the interval [2000/01 – 2009/01]*

Notice that the input data set includes the market prices during two market crashes, so the network should be able to learn about the market's behavior during those crashes. You already learned from the previous examples that to make predictions outside the training range you need to transform the original data to a format that will allow you to do this. So, as part of this transformation, you create the price difference data set with records that include these two fields:

- *Field 1*: Percent difference between the current and previous month prices
- *Field 2*: Percent difference between the next and current month prices

Table 11-2 shows the fragment of the transformed price difference data set.

Table 11-2. *Fragment of the Price Difference Data Set*

Field 1	Field 2		
priceDiffPerc	**targetPriceDiffPerc**	**Date**	**InputPrice**
-5.090352221	-2.010814222	200001	1394.46
-2.010814222	9.671989579	200002	1366.42
9.671989579	-3.079582004	200003	1498.58
-3.079582004	-2.191499762	200004	1452.43
-2.191499762	2.39335492	200005	1420.6
2.39335492	-1.63412622	200006	1454.6
-1.63412622	6.069903483	200007	1430.83
6.069903483	-5.348294766	200008	1517.68
-5.348294766	-0.494949565	200009	1436.51
-0.494949565	-8.006856024	200010	1429.4
-8.006856024	0.405338606	200011	1314.95
0.405338606	3.463659224	200012	1320.28
3.463659224	-9.229068601	200101	1366.01
-9.229068601	-6.420471958	200102	1239.94
-6.420471958	7.681435454	200103	1160.33
7.681435454	0.509019897	200104	1249.46
0.509019897	-2.503543501	200105	1255.82
-2.503543501	-1.07401297	200106	1224.38
-1.07401297	-6.410838569	200107	1211.23
-6.410838569	-8.172338962	200108	1133.58
-8.172338962	1.809902588	200109	1040.94
1.809902588	7.517597992	200110	1059.78
7.517597992	0.757382948	200111	1139.45
0.757382948	-1.557382761	200112	1148.08
-1.557382761	-2.076623606	200201	1130.2

(*continued*)

Table 11-2. (*continued*)

Field 1	Field 2		
priceDiffPerc	**targetPriceDiffPerc**	**Date**	**InputPrice**
-2.076623606	3.673886133	200202	1106.73
3.673886133	-6.141765224	200203	1147.39
-6.141765224	-0.908145452	200204	1076.92
-0.908145452	-7.245534794	200205	1067.14
-7.245534794	-7.90042634	200206	989.82
-7.90042634	0.488141989	200207	911.62
0.488141989	-11.00243431	200208	916.07
-11.00243431	8.64488274	200209	815.28
8.64488274	5.706963512	200210	885.76
5.706963512	-6.033258216	200211	936.31
-6.033258216	-2.741469846	200212	879.82
-2.741469846	-1.700362276	200301	855.7
-1.700362276	0.835760566	200302	841.15
0.835760566	8.104411799	200303	848.18
8.104411799	5.089866073	200304	916.92
5.089866073	1.132224286	200305	963.59

Columns 3 and 4 were included to facilitate the calculation of columns 1 and 2, but they are ignored during processing. As always, you normalize this data set on the interval [-1, 1]. Table 11-3 shows the normalized data set.

Table 11-3. *Fragment of the Normalized Price Difference Data Set*

priceDiffPerc	targetPriceDiffPerc	Date	inputPrice
-0.006023481	0.199279052	200001	1394.46
0.199279052	0.978132639	200002	1366.42
0.978132639	0.128027866	200003	1498.58
0.128027866	0.187233349	200004	1452.43
0.187233349	0.492890328	200005	1420.6
0.492890328	0.224391585	200006	1454.6
0.224391585	0.737993566	200007	1430.83
0.737993566	-0.023219651	200008	1517.68
-0.023219651	0.300336696	200009	1436.51
0.300336696	-0.200457068	200010	1429.4
-0.200457068	0.360355907	200011	1314.95
0.360355907	0.564243948	200012	1320.28
0.564243948	-0.281937907	200101	1366.01
-0.281937907	-0.094698131	200102	1239.94
-0.094698131	0.84542903	200103	1160.33
0.84542903	0.367267993	200104	1249.46
0.367267993	0.166430433	200105	1255.82
0.166430433	0.261732469	200106	1224.38
0.261732469	-0.094055905	200107	1211.23
-0.094055905	-0.211489264	200108	1133.58
-0.211489264	0.453993506	200109	1040.94
0.453993506	0.834506533	200110	1059.78
0.834506533	0.38382553	200111	1139.45
0.38382553	0.229507816	200112	1148.08
0.229507816	0.19489176	200201	1130.2

(*continued*)

Table 11-3. (*continued*)

priceDiffPerc	targetPriceDiffPerc	Date	inputPrice
0.19489176	0.578259076	200202	1106.73
0.578259076	-0.076117682	200203	1147.39
-0.076117682	0.272790303	200204	1076.92
0.272790303	-0.14970232	200205	1067.14
-0.14970232	-0.193361756	200206	989.82
-0.193361756	0.365876133	200207	911.62
0.365876133	-0.400162287	200208	916.07
-0.400162287	0.909658849	200209	815.28
0.909658849	0.713797567	200210	885.76
0.713797567	-0.068883881	200211	936.31
-0.068883881	0.150568677	200212	879.82

Again, ignore columns 3 and 4. They are used here for the convention of preparing this data set, but they are not processed.

Including Function Topology in the Data Set

Next, you will include information about the function topology in the data set because it allows you to match not only a single Field 1 value but the set of 12 Field 1 values (which means matching one year worth of data). To do this, you build the training file with the sliding window records. Each sliding window record consists of 12 `inputPriceDiffPerc` fields from 12 original records plus the `targetPriceDiffPerc` field from the next original record (the record that follows the original record's 12). Table 11-4 shows the fragment of the resulting data set.

Table 11-4. *Fragment of the Training Data Set That Consists of Sliding Window Records*

Sliding Windows												
0.591	0.55	0.165	0.459	0.206	0.199	0.533	0.332	0.573	0.259	0.38	0.215	0.327
0.55	0.165	0.459	0.206	0.199	0.533	0.332	0.573	0.259	0.38	0.215	0.568	0.503
0.165	0.459	0.206	0.199	0.533	0.332	0.573	0.259	0.38	0.215	0.568	0.327	0.336
0.459	0.206	0.199	0.533	0.332	0.573	0.259	0.38	0.215	0.568	0.327	0.503	0.407
0.206	0.199	0.533	0.332	0.573	0.259	0.38	0.215	0.568	0.327	0.503	0.336	0.414
0.199	0.533	0.332	0.573	0.259	0.38	0.215	0.568	0.327	0.503	0.336	0.407	0.127
0.533	0.332	0.573	0.259	0.38	0.215	0.568	0.327	0.503	0.336	0.407	0.414	0.334
0.332	0.573	0.259	0.38	0.215	0.568	0.327	0.503	0.336	0.407	0.414	0.127	0.367
0.573	0.259	0.38	0.215	0.568	0.327	0.503	0.336	0.407	0.414	0.127	0.334	0.475
0.259	0.38	0.215	0.568	0.327	0.503	0.336	0.407	0.414	0.127	0.334	0.367	0.497
0.38	0.215	0.568	0.327	0.503	0.336	0.407	0.414	0.127	0.334	0.367	0.475	0.543
0.215	0.568	0.327	0.503	0.336	0.407	0.414	0.127	0.334	0.367	0.475	0.497	0.443
0.568	0.327	0.503	0.336	0.407	0.414	0.127	0.334	0.367	0.475	0.497	0.543	0.417
0.327	0.503	0.336	0.407	0.414	0.127	0.334	0.367	0.475	0.497	0.543	0.443	0.427
0.503	0.336	0.407	0.414	0.127	0.334	0.367	0.475	0.497	0.543	0.443	0.417	0.188
0.336	0.407	0.414	0.127	0.334	0.367	0.475	0.497	0.543	0.443	0.417	0.427	0.400
0.407	0.414	0.127	0.334	0.367	0.475	0.497	0.543	0.443	0.417	0.427	0.188	0.622
0.414	0.127	0.334	0.367	0.475	0.497	0.543	0.443	0.417	0.427	0.188	0.400	0.55
0.127	0.334	0.367	0.475	0.497	0.543	0.443	0.417	0.427	0.188	0.4	0.622	0.215
0.334	0.367	0.475	0.497	0.543	0.443	0.417	0.427	0.188	0.4	0.622	0.55	0.12
0.367	0.475	0.497	0.543	0.443	0.417	0.427	0.188	0.400	0.622	0.55	0.215	0.419
0.475	0.497	0.543	0.443	0.417	0.427	0.188	0.400	0.622	0.55	0.215	0.12	0.572
0.497	0.543	0.443	0.417	0.427	0.188	0.400	0.622	0.55	0.215	0.12	0.419	0.432
0.543	0.443	0.417	0.427	0.188	0.4	0.622	0.55	0.215	0.12	0.419	0.572	0.04

(*continued*)

Table 11-4. (*continued*)

Sliding Windows												
0.443	0.417	0.427	0.188	0.400	0.622	0.55	0.215	0.12	0.419	0.572	0.432	0.276
0.417	0.427	0.188	0.400	0.622	0.550	0.215	0.12	0.419	0.572	0.432	0.04	-0.074
0.427	0.188	0.400	0.622	0.55	0.215	0.12	0.419	0.572	0.432	0.040	0.276	0.102
0.188	0.400	0.622	0.55	0.215	0.12	0.419	0.572	0.432	0.04	0.276	-0.074	0.294
0.400	0.622	0.55	0.215	0.12	0.419	0.572	0.432	0.04	0.276	-0.07	0.102	0.650

Because the function is noncontinuous, you break this data set into micro-batches (single-month records).

Building Micro-Batch Files

Listing 11-1 shows the program code that builds the micro-batch files from the normalized sliding window data set.

Listing 11-1. Program Code That Builds the Micro-Batch File

```
// ============================================================================
// Build micro-batch files from the normalized sliding windows file.
// Each micro-batch dataset should consists of 12 inputPriceDiffPerc fields
// taken from 12 records in the original file plus a single
   targetPriceDiffPerc
// value taken from the next month record. Each micro-batch includes the label
// record.
// ============================================================================

package sample7_build_microbatches;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.PrintWriter;
```

```
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.*;
import java.util.Properties;

public class Sample7_Build_MicroBatches
 {

    // Config for Training
    static int numberOfRowsInInputFile = 121;
    static int numberOfRowsInBatch = 13;
    static String  strInputFileName =
      "C:/My_Neural_Network_Book/Book_Examples/Sample7_SlidWindows_
      Train.csv";
    static String  strOutputFileNameBase =
      "C:/My_Neural_Network_Book/Temp_Files/Sample7_Microbatches_
      Train_Batch_";

    // Config for Testing
    //static int numberOfRowsInInputFile = 122;
    //static int numberOfRowsInBatch = 13;
    //static String  strInputFileName =
    //    "C:/My_Neural_Network_Book/Book_Examples/Sample7_SlidWindows_
          Test.csv";
    //static String  strOutputFileNameBase =
    //    "C:/My_Neural_Network_Book/Temp_Files/Sample7_Microbatches_
          Test_Batch_";

    static InputStream input = null;

    // ================================================================
    // Main method
    // ================================================================
    public static void main(String[] args)
```

```
{
  BufferedReader br;
  PrintWriter out;
  String cvsSplitBy = ",";
  String line = "";
  String lineLabel = "";
  String[] strOutputFileNames = new String[1070];
  String iString;
  String strOutputFileName;
  String[] strArrLine = new String[1086];

  int i;
  int r;

  // Read the original data and break it into batches

  try
   {
     // Delete all output file if they exist

     for (i = 0; i < numberOfRowsInInputFile; i++)
      {
        iString = Integer.toString(i);

        if(i < 10)
           strOutputFileName = strOutputFileNameBase + "00" +
           iString + ".csv";
        else
           if (i >= 10 && i < 100)
              strOutputFileName = strOutputFileNameBase + "0" +
              iString + ".csv";
            else
              strOutputFileName = strOutputFileNameBase + iString +
              ".csv";

        Files.deleteIfExists(Paths.get(strOutputFileName));
      }
```

```
i = -1;   // Input line number
r = -2;   // index to write in the memory
br = new BufferedReader(new FileReader(strInputFileName));

// Load all input recodes into memory
while ((line = br.readLine()) != null)
 {
   i++;
   r++;
   if (i == 0)
    {
      // Save the label line
      lineLabel = line;
    }
   else
    {
      // Save the data in memory
      strArrLine[r] = line;
    }

 }  // End of WHILE

br.close();

// Build batches
br = new BufferedReader(new FileReader(strInputFileName));

for (i = 0; i < numberOfRowsInInputFile - 1; i++)
  {
    iString = Integer.toString(i);

    // Construct the mini-batch
     if(i < 10)
       strOutputFileName = strOutputFileNameBase + "00" +
       iString + ".csv";
    else
```

```
                if (i >= 10 && i < 100)
                   strOutputFileName = strOutputFileNameBase + "0" +
                   iString + ".csv";
                 else
                   strOutputFileName = strOutputFileNameBase + iString +
                   ".csv";

              out = new PrintWriter(new BufferedWriter(new FileWriter
              (strOutputFileName)));

              // write the header line as it is
              out.println(lineLabel);
              out.println(strArrLine[i]);

              out.close();

            }  // End of FOR i loop

          }  // End of TRY
         catch (IOException io)
          {
              io.printStackTrace();
          }

     }  // End of the Main method

}  // End of the class
```

This program breaks the sliding window data set into micro-batch files. Figure 11-2 shows a fragment of the list of micro-batch files.

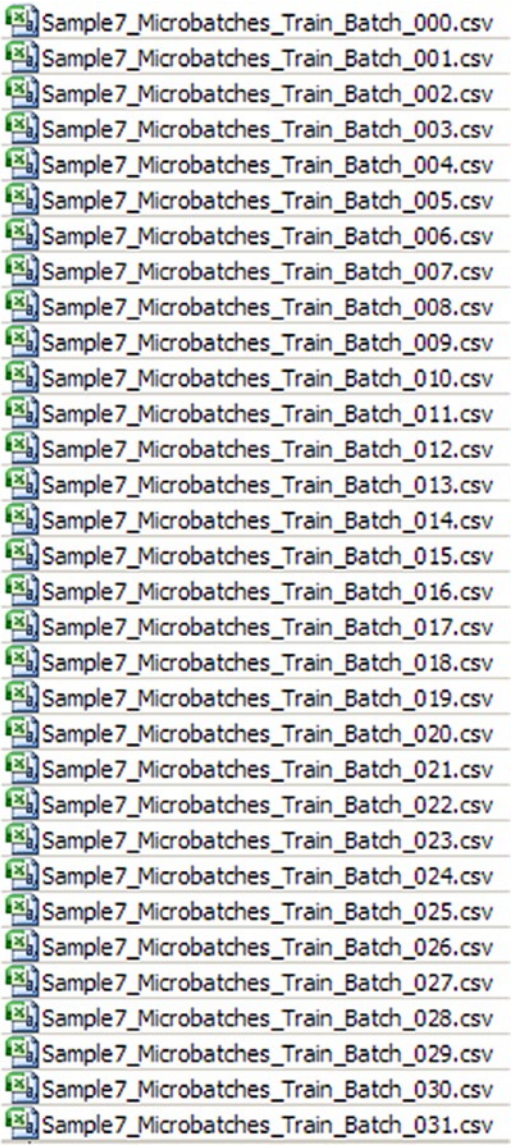


Figure 11-2. *Fragment of the list of micro-batch files*

Listing 11-2 shows how each micro-batch data set looks when it is opened.

Listing 11-2. Sample of the Micro-Batch File

```
Sliding window micro-batch record
-0.006023481 0.199279052 0.978132639 0.128027866 0.187233349 0.492890328
0.224391585 0.737993566 -0.023219651 0.300336696 -0.200457068 0.360355907
-0.281937907

   Micro-batch files are the training files to be processed by the network.
```

Network Architecture

Figure 11-3 shows the network architecture for this example. The network has 12 input neurons, seven hidden layers (each with 25 neurons), and an output layer with a single neuron.

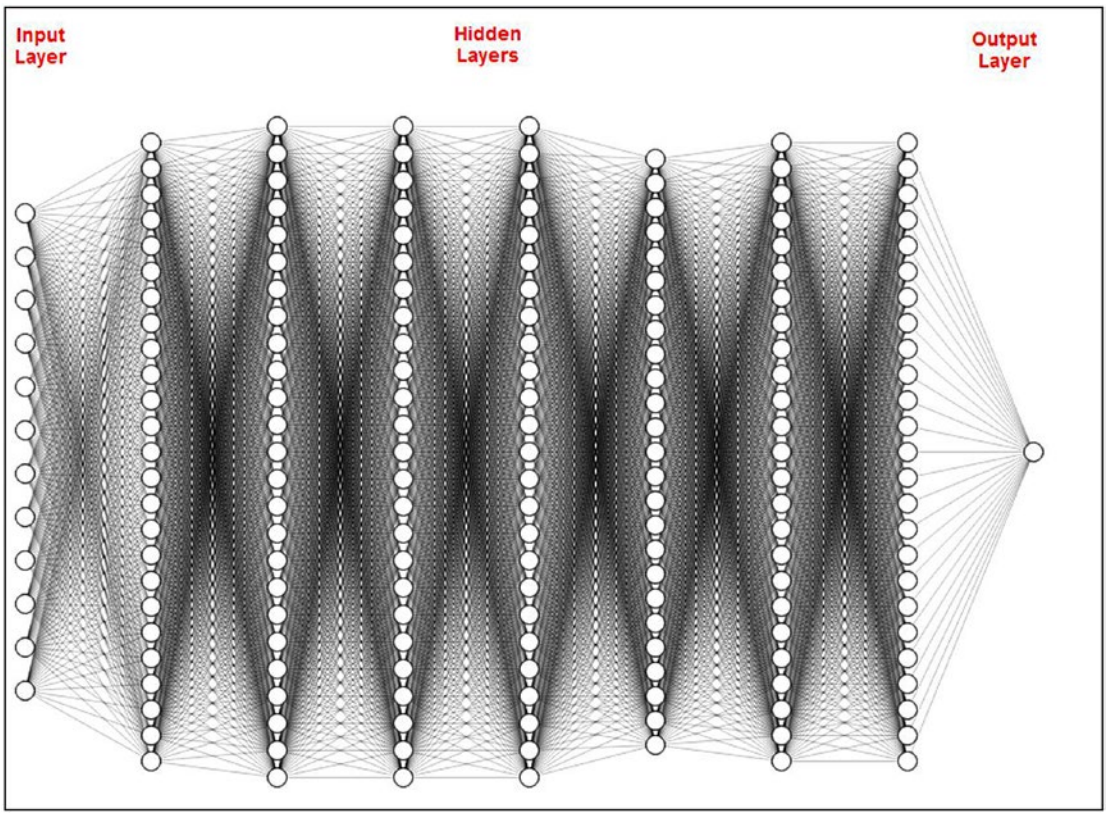


Figure 11-3. *Network architecture*

Now you are ready to build the network processing program.

Program Code

Listing 11-3 shows the program code.

Listing 11-3. Code of the Neural Network Processing Program

```
// ========================================================================
// Approximate the SPY prices function using the micro-batch method.
// Each micro-batch file includes the label record and the data record.
// The data record contains 12 inputPriceDiffPerc fields plus one
// targetPriceDiffPerc field.
//
// The number of input Layer neurons is 12
// The number of output Layer neurons is 1
// ========================================================================

package sample7;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.PrintWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.*;
import java.util.Properties;
import java.time.YearMonth;
import java.awt.Color;
import java.awt.Font;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.text.DateFormat;
```

```
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.LocalDate;
import java.time.Month;
import java.time.ZoneId;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.List;
import java.util.Locale;
import java.util.Properties;
import org.encog.Encog;
import org.encog.engine.network.activation.ActivationTANH;
import org.encog.engine.network.activation.ActivationReLU;
import org.encog.ml.data.MLData;
import org.encog.ml.data.MLDataPair;
import org.encog.ml.data.MLDataSet;
import org.encog.ml.data.buffer.MemoryDataLoader;
import org.encog.ml.data.buffer.codec.CSVDataCODEC;
import org.encog.ml.data.buffer.codec.DataSetCODEC;
import org.encog.neural.networks.BasicNetwork;
import org.encog.neural.networks.layers.BasicLayer;
import org.encog.neural.networks.training.propagation.resilient.
ResilientPropagation;
import org.encog.persist.EncogDirectoryPersistence;
import org.encog.util.csv.CSVFormat;

import org.knowm.xchart.SwingWrapper;
import org.knowm.xchart.XYChart;
import org.knowm.xchart.XYChartBuilder;
import org.knowm.xchart.XYSeries;
import org.knowm.xchart.demo.charts.ExampleChart;
import org.knowm.xchart.style.Styler.LegendPosition;
import org.knowm.xchart.style.colors.ChartColor;
import org.knowm.xchart.style.colors.XChartSeriesColors;
import org.knowm.xchart.style.lines.SeriesLines;
```

```
import org.knowm.xchart.style.markers.SeriesMarkers;
import org.knowm.xchart.BitmapEncoder;
import org.knowm.xchart.BitmapEncoder.BitmapFormat;
import org.knowm.xchart.QuickChart;
import org.knowm.xchart.SwingWrapper;

public class Sample7 implements ExampleChart<XYChart>
{
   // Normalization parameters

   // Normalizing interval
   static double Nh =  1;
   static double Nl = -1;

   // inputPriceDiffPerc
   static double inputPriceDiffPercDh = 10.00;
   static double inputPriceDiffPercDl = -20.00;

   // targetPriceDiffPerc
   static double targetPriceDiffPercDh = 10.00;
   static double targetPriceDiffPercDl = -20.00;

   static String cvsSplitBy = ",";
   static Properties prop = null;
   static Date workDate = null;
   static int paramErrorCode = 0;
   static int paramBatchNumber = 0;
   static int paramDayNumber = 0;
   static String strWorkingMode;
   static String strNumberOfBatchesToProcess;
   static String strNumberOfRowsInInputFile;
   static String strNumberOfRowsInBatches;
   static String strIputNeuronNumber;
   static String strOutputNeuronNumber;
   static String strNumberOfRecordsInTestFile;
   static String strInputFileNameBase;
   static String strTestFileNameBase;
   static String strSaveNetworkFileNameBase;
```

```
static String strTrainFileName;
static String strValidateFileName;
static String strChartFileName;
static String strDatesTrainFileName;
static String strPricesFileName;
static int intWorkingMode;
static int intNumberOfBatchesToProcess;
static int intNumberOfRowsInBatches;
static int intInputNeuronNumber;
static int intOutputNeuronNumber;
static String strOutputFileName;
static String strSaveNetworkFileName;
static String strNumberOfMonths;
static String strYearMonth;
static XYChart Chart;
static String iString;
static double inputPriceFromFile;

static List<Double> xData = new ArrayList<Double>();
static List<Double> yData1 = new ArrayList<Double>();
static List<Double> yData2 = new ArrayList<Double>();

// These arrays is where the two Date files are loaded
static Date[] yearDateTraining = new Date[150];
static String[] strTrainingFileNames = new String[150];
static String[] strTestingFileNames = new String[150];
static String[] strSaveNetworkFileNames = new String[150];

static BufferedReader br3;

static double recordNormInputPriceDiffPerc_00 = 0.00;
static double recordNormInputPriceDiffPerc_01 = 0.00;
static double recordNormInputPriceDiffPerc_02 = 0.00;
static double recordNormInputPriceDiffPerc_03 = 0.00;
static double recordNormInputPriceDiffPerc_04 = 0.00;
static double recordNormInputPriceDiffPerc_05 = 0.00;
static double recordNormInputPriceDiffPerc_06 = 0.00;
```

```
    static double recordNormInputPriceDiffPerc_07 = 0.00;
    static double recordNormInputPriceDiffPerc_08 = 0.00;
    static double recordNormInputPriceDiffPerc_09 = 0.00;
    static double recordNormInputPriceDiffPerc_10 = 0.00;
    static double recordNormInputPriceDiffPerc_11 = 0.00;

    static double recordNormTargetPriceDiffPerc = 0.00;
    static double tempMonth = 0.00;
    static int intNumberOfSavedNetworks = 0;

    static double[] linkToSaveInputPriceDiffPerc_00 = new double[150];
    static double[] linkToSaveInputPriceDiffPerc_01 = new double[150];
    static double[] linkToSaveInputPriceDiffPerc_02 = new double[150];
    static double[] linkToSaveInputPriceDiffPerc_03 = new double[150];
    static double[] linkToSaveInputPriceDiffPerc_04 = new double[150];
    static double[] linkToSaveInputPriceDiffPerc_05 = new double[150];
    static double[] linkToSaveInputPriceDiffPerc_06 = new double[150];
    static double[] linkToSaveInputPriceDiffPerc_07 = new double[150];
    static double[] linkToSaveInputPriceDiffPerc_08 = new double[150];
    static double[] linkToSaveInputPriceDiffPerc_09 = new double[150];
    static double[] linkToSaveInputPriceDiffPerc_10 = new double[150];
    static double[] linkToSaveInputPriceDiffPerc_11 = new double[150];

    static int[] returnCodes  = new int[3];
    static int intDayNumber = 0;
    static File file2 = null;
    static double[] linkToSaveTargetPriceDiffPerc = new double[150];
    static double[] arrPrices = new double[150];

  @Override
  public XYChart getChart()
   {

    // Create Chart

    Chart = new XYChartBuilder().width(900).height(500).title(getClass().
    getSimpleName()).xAxisTitle("Month").yAxisTitle("Price").build();
```

```
// Customize Chart
Chart.getStyler().setPlotBackgroundColor(ChartColor.getAWTColor
(ChartColor.GREY));
Chart.getStyler().setPlotGridLinesColor(new Color(255, 255, 255));
Chart.getStyler().setChartBackgroundColor(Color.WHITE);
Chart.getStyler().setLegendBackgroundColor(Color.PINK);
Chart.getStyler().setChartFontColor(Color.MAGENTA);
Chart.getStyler().setChartTitleBoxBackgroundColor(new Color(0, 222, 0));
Chart.getStyler().setChartTitleBoxVisible(true);
Chart.getStyler().setChartTitleBoxBorderColor(Color.BLACK);
Chart.getStyler().setPlotGridLinesVisible(true);
Chart.getStyler().setAxisTickPadding(20);
Chart.getStyler().setAxisTickMarkLength(15);
Chart.getStyler().setPlotMargin(20);
Chart.getStyler().setChartTitleVisible(false);
Chart.getStyler().setChartTitleFont(new Font(Font.MONOSPACED,
Font.BOLD, 24));
Chart.getStyler().setLegendFont(new Font(Font.SERIF, Font.PLAIN, 18));
// Chart.getStyler().setLegendPosition(LegendPosition.InsideSE);
Chart.getStyler().setLegendPosition(LegendPosition.OutsideE);
Chart.getStyler().setLegendSeriesLineLength(12);
Chart.getStyler().setAxisTitleFont(new Font(Font.SANS_SERIF,
Font.ITALIC, 18));
Chart.getStyler().setAxisTickLabelsFont(new Font(Font.SERIF,
Font.PLAIN, 11));
Chart.getStyler().setDatePattern("yyyy-MM");
Chart.getStyler().setDecimalPattern("#0.00");

// Configuration

 // Set the mode of running this program
 intWorkingMode = 1; // Training mode

 if(intWorkingMode == 1)
   {
      // Training mode
      intNumberOfBatchesToProcess = 120;
```

```
            strInputFileNameBase =
               "C:/My_Neural_Network_Book/Temp_Files/Sample7_Microbatches_
               Train_Batch_";
            strSaveNetworkFileNameBase =
               "C:/My_Neural_Network_Book/Temp_Files/Sample7_Save_Network_
               Batch_";
            strChartFileName = "C:/My_Neural_Network_Book/Temp_Files/Sample7_
            XYLineChart_Train.jpg";
            strDatesTrainFileName =
                "C:/My_Neural_Network_Book/Book_Examples/Sample7_Dates_Real_
                SP500_3000.csv";
             strPricesFileName = "C:/My_Neural_Network_Book/Book_Examples/
                  Sample7_InputPrice_SP500_200001_200901.csv";
        }
    else
        {
            // Testing mode
            intNumberOfBatchesToProcess = 121;
            intNumberOfSavedNetworks = 120;
            strInputFileNameBase =
              "C:/My_Neural_Network_Book/Temp_Files/Sample7_Microbatches_
              Test_Batch_";
           strSaveNetworkFileNameBase =
             "C:/My_Neural_Network_Book/Temp_Files/Sample7_Save_Network_
             Batch_";
           strChartFileName =
             "C:/My_Neural_Network_Book/Book_Examples/Sample7_XYLineChart_
             Test.jpg";
           strDatesTrainFileName =
             "C:/My_Neural_Network_Book/Book_Examples/Sample7_Dates_Real_
             SP500_3000.csv";
          trPricesFileName = "C:/My_Neural_Network_Book/Book_Examples/
              Sample7_InputPrice_SP500_200001_200901.csv";
      }
```

```
// Common configuration
intNumberOfRowsInBatches = 1;
intInputNeuronNumber = 12;
intOutputNeuronNumber = 1;

// Generate training batch file names and the corresponding Save
   Network file names and
// save them arrays
for (int i = 0; i < intNumberOfBatchesToProcess; i++)
 {
   iString = Integer.toString(i);

   // Construct the training batch names
   if (i < 10)
    {
      strOutputFileName = strInputFileNameBase + "00" + iString +
      ".csv";
      strSaveNetworkFileName = strSaveNetworkFileNameBase + "00" +
      iString + ".csv";
    }
   else
    {
      if(i >=10 && i < 100)
       {
        strOutputFileName = strInputFileNameBase + "0" + iString + ".csv";
        strSaveNetworkFileName = strSaveNetworkFileNameBase + "0" +
        iString + ".csv";
       }
      else
       {
         strOutputFileName = strInputFileNameBase + iString + ".csv";
         strSaveNetworkFileName = strSaveNetworkFileNameBase +
         iString + ".csv";
       }

    }
```

```
      strSaveNetworkFileNames[i] = strSaveNetworkFileName;

      if(intWorkingMode == 1)
       {
        strTrainingFileNames[i] = strOutputFileName;

        File file1 = new File(strSaveNetworkFileNames[i]);

        if(file1.exists())
            file1.delete();
       }
      else
       strTestingFileNames[i] = strOutputFileName;

     }  // End the FOR loop

      // Build the array linkToSaveInputPriceDiffPerc_01
      String tempLine = null;
      String[] tempWorkFields = null;

      recordNormInputPriceDiffPerc_00 = 0.00;
      recordNormInputPriceDiffPerc_01 = 0.00;
      recordNormInputPriceDiffPerc_02 = 0.00;
      recordNormInputPriceDiffPerc_03 = 0.00;
      recordNormInputPriceDiffPerc_04 = 0.00;
      recordNormInputPriceDiffPerc_05 = 0.00;
      recordNormInputPriceDiffPerc_06 = 0.00;
      recordNormInputPriceDiffPerc_07 = 0.00;
      recordNormInputPriceDiffPerc_08 = 0.00;
      recordNormInputPriceDiffPerc_09 = 0.00;
      recordNormInputPriceDiffPerc_10 = 0.00;
      recordNormInputPriceDiffPerc_11 = 0.00;

      double recordNormTargetPriceDiffPerc = 0.00;

      try
       {
          for (int m = 0; m < intNumberOfBatchesToProcess; m++)
```

```
{
   if(intWorkingMode == 1)
     br3 = new BufferedReader(new FileReader(strTraining
     FileNames[m]));
   else
     br3 = new BufferedReader(new FileReader(strTesting
     FileNames[m]));

   // Skip the label record
   tempLine = br3.readLine();
   tempLine = br3.readLine();

   // Break the line using comma as separator
   tempWorkFields = tempLine.split(cvsSplitBy);

   recordNormInputPriceDiffPerc_00 = Double.parseDouble
   (tempWorkFields[0]);
   recordNormInputPriceDiffPerc_01 = Double.parseDouble
   (tempWorkFields[1]);
   recordNormInputPriceDiffPerc_02 = Double.parseDouble
   (tempWorkFields[2]);
   recordNormInputPriceDiffPerc_03 = Double.parseDouble
   (tempWorkFields[3]);
   recordNormInputPriceDiffPerc_04 = Double.parseDouble
   (tempWorkFields[4]);
   recordNormInputPriceDiffPerc_05 = Double.parseDouble
   (tempWorkFields[5]);
   recordNormInputPriceDiffPerc_06 = Double.parseDouble
   (tempWorkFields[6]);
   recordNormInputPriceDiffPerc_07 = Double.parseDouble
   (tempWorkFields[7]);
   recordNormInputPriceDiffPerc_08 = Double.parseDouble
   (tempWorkFields[8]);
   recordNormInputPriceDiffPerc_09 = Double.parseDouble
   (tempWorkFields[9]);
   recordNormInputPriceDiffPerc_10 = Double.parseDouble
   (tempWorkFields[10]);
```

```
            recordNormInputPriceDiffPerc_11 = Double.parseDouble
            (tempWorkFields[11]);

            recordNormTargetPriceDiffPerc = Double.parseDouble
            (tempWorkFields[12]);

            linkToSaveInputPriceDiffPerc_00[m] = recordNormInputPrice
            DiffPerc_00;
            linkToSaveInputPriceDiffPerc_01[m] = recordNormInputPrice
            DiffPerc_01;
            linkToSaveInputPriceDiffPerc_02[m] = recordNormInputPrice
            DiffPerc_02;
            linkToSaveInputPriceDiffPerc_03[m] = recordNormInputPrice
            DiffPerc_03;
            linkToSaveInputPriceDiffPerc_04[m] = recordNormInputPrice
            DiffPerc_04;
            linkToSaveInputPriceDiffPerc_05[m] = recordNormInputPrice
            DiffPerc_05;
            linkToSaveInputPriceDiffPerc_06[m] = recordNormInputPrice
            DiffPerc_06;
            linkToSaveInputPriceDiffPerc_07[m] = recordNormInputPrice
            DiffPerc_07;
            linkToSaveInputPriceDiffPerc_08[m] = recordNormInputPrice
            DiffPerc_08;
            linkToSaveInputPriceDiffPerc_09[m] = recordNormInputPrice
            DiffPerc_09;
            linkToSaveInputPriceDiffPerc_10[m] = recordNormInputPrice
            DiffPerc_10;
            linkToSaveInputPriceDiffPerc_11[m] = recordNormInputPrice
            DiffPerc_11;

            linkToSaveTargetPriceDiffPerc[m] = recordNormTargetPrice
            DiffPerc;

        }  // End the FOR loop

    // Load dates into memory
    loadDatesInMemory();
```

```
  // Load Prices into memory
  loadPriceFileInMemory();

  file2 = new File(strChartFileName);

  if(file2.exists())
    file2.delete();

  // Test the working mode
  if(intWorkingMode == 1)
    {
      // Train batches and save the trained networks
      int  paramBatchNumber;

      returnCodes[0] = 0;    // Clear the error Code
      returnCodes[1] = 0;    // Set the initial batch Number to 1;
      returnCodes[2] = 0;    // Set the initial day number;

      do
       {
         paramErrorCode = returnCodes[0];
         paramBatchNumber = returnCodes[1];
         paramDayNumber = returnCodes[2];

        returnCodes =
          trainBatches(paramErrorCode,paramBatchNumber,paramDayNumber);
       } while (returnCodes[0] > 0);

    }   // End the train logic
  else
   {
       // Load and test the network logic
        loadAndTestNetwork();

   }  // End of ELSE

}     // End of Try
```

```
    catch (Exception e1)
     {
       e1.printStackTrace();
     }

    Encog.getInstance().shutdown();

    return Chart;

  }  // End of method

  // ================================================================
  // Load CSV to memory.
  // @return The loaded dataset.
  // ================================================================
  public static MLDataSet loadCSV2Memory(String filename, int input,
  int ideal,
    Boolean headers, CSVFormat format, Boolean significance)
    {
       DataSetCODEC codec = new CSVDataCODEC(new File(filename), format,
       headers, input, ideal, significance);
       MemoryDataLoader load = new MemoryDataLoader(codec);
       MLDataSet dataset = load.external2Memory();
       return dataset;
    }

  // ================================================================
  //  The main method.
  //  @param Command line arguments. No arguments are used.
  // ================================================================
  public static void main(String[] args)
   {
     ExampleChart<XYChart> exampleChart = new Sample7();
     XYChart Chart = exampleChart.getChart();
     new SwingWrapper<XYChart>(Chart).displayChart();
   } // End of the main method
```

```
//=====================================================================
// Mode 0. Train batches as individual networks, saving them in separate
   files on disk.
//=====================================================================
static public int[] trainBatches(int paramErrorCode,int paramBatch
Number,
      int paramDayNumber)
  {
    int rBatchNumber;

    double realDenormTargetToPredictPricePerc = 0;
    double maxGlobalResultDiff = 0.00;
    double averGlobalResultDiff = 0.00;
    double sumGlobalResultDiff = 0.00;
    double normTargetPriceDiffPerc = 0.00;
    double normPredictPriceDiffPerc = 0.00;
    double normInputPriceDiffPercFromRecord = 0.00;
    double denormTargetPriceDiffPerc;
    double denormPredictPriceDiffPerc;
    double denormInputPriceDiffPercFromRecord;
    double workNormInputPrice;
    Date tempDate;
    double trainError;
    double realDenormPredictPrice;
    double realDenormTargetPrice;

    // Build the network
    BasicNetwork network = new BasicNetwork();

    // Input layer
   network.addLayer(new BasicLayer(null,true,intInputNeuronNumber));

   // Hidden layer.
   network.addLayer(new BasicLayer(new ActivationTANH(),true,25));
   network.addLayer(new BasicLayer(new ActivationTANH(),true,25));
   network.addLayer(new BasicLayer(new ActivationTANH(),true,25));
   network.addLayer(new BasicLayer(new ActivationTANH(),true,25));
```

```
network.addLayer(new BasicLayer(new ActivationTANH(),true,25));
network.addLayer(new BasicLayer(new ActivationTANH(),true,25));
network.addLayer(new BasicLayer(new ActivationTANH(),true,25));

// Output layer
network.addLayer(new BasicLayer(new ActivationTANH(),false,intOutputN
euronNumber));

network.getStructure().finalizeStructure();
network.reset();

// Loop over batches
intDayNumber = paramDayNumber;  // Day number for the chart

for (rBatchNumber = paramBatchNumber; rBatchNumber < intNumberOf
BatchesToProcess;
         rBatchNumber++)
 {
   intDayNumber++;

   //if(rBatchNumber == 201)
   // rBatchNumber = rBatchNumber;

   // Load the training CVS file for the current batch in memory
   MLDataSet trainingSet = loadCSV2Memory(strTrainingFileNames
   [rBatchNumber],
     intInputNeuronNumber,intOutputNeuronNumber,true,CSVFormat.
     ENGLISH,false);

   // train the neural network
   ResilientPropagation train = new ResilientPropagation(network,
   trainingSet);

   int epoch = 1;
   double tempLastErrorPerc = 0.00;

   do
     {
       train.iteration();

       epoch++;
```

```
 for (MLDataPair pair1:  trainingSet)
  {
    MLData inputData = pair1.getInput();
    MLData actualData = pair1.getIdeal();
    MLData predictData = network.compute(inputData);

    // These values are normalized
    normTargetPriceDiffPerc = actualData.getData(0);
    normPredictPriceDiffPerc = predictData.getData(0);

    // De-normalize these values
    denormTargetPriceDiffPerc = ((targetPriceDiffPercDl - target
    PriceDiffPercDh)*normTargetPriceDiffPerc - Nh*targetPrice
    DiffPercDl + targetPriceDiffPercDh*Nl)/(Nl - Nh);

    denormPredictPriceDiffPerc =((targetPriceDiffPercDl - target
    PriceDiffPercDh)*normPredictPriceDiffPerc - Nh*target
    PriceDiffPercDl + targetPriceDiffPercDh*Nl)/(Nl - Nh);

    inputPriceFromFile = arrPrices[rBatchNumber+12];

    realDenormTargetPrice = inputPriceFromFile + inputPriceFrom
    File*denormTargetPriceDiffPerc/100;

    realDenormPredictPrice = inputPriceFromFile + inputPriceFrom
    File*denormPredictPriceDiffPerc/100;

    realDenormTargetToPredictPricePerc = (Math.abs(realDenorm
    TargetPrice - realDenormPredictPrice)/realDenormTarget
    Price)*100;

  }
if (epoch >= 500 && realDenormTargetToPredictPricePerc > 0.00091)
  {
   returnCodes[0] = 1;
   returnCodes[1] = rBatchNumber;
   returnCodes[2] = intDayNumber-1;
```

```
            //System.out.println("Try again");
            return returnCodes;
           }

          //System.out.println(realDenormTargetToPredictPricePerc);
       } while(realDenormTargetToPredictPricePerc >  0.0009);

    // This batch is optimized

    // Save the network for the current batch
    EncogDirectoryPersistence.saveObject(newFile(strSaveNetworkFileNames
    [rBatchNumber]),network);

    // Print the trained neural network results for the batch
    //System.out.println("Trained Neural Network Results");

    // Get the results after the network optimization
    int i = - 1; // Index of the array to get results

   maxGlobalResultDiff = 0.00;
   averGlobalResultDiff = 0.00;
   sumGlobalResultDiff = 0.00;

  //if (rBatchNumber == 857)
  //    i = i;

  // Validation
  for (MLDataPair pair:  trainingSet)
    {
      i++;

      MLData inputData = pair.getInput();
      MLData actualData = pair.getIdeal();
      MLData predictData = network.compute(inputData);

      // These values are Normalized as the whole input is
      normTargetPriceDiffPerc = actualData.getData(0);
      normPredictPriceDiffPerc = predictData.getData(0);
      //normInputPriceDiffPercFromRecord[i] = inputData.getData(0);
      normInputPriceDiffPercFromRecord = inputData.getData(0);
```

```
// De-normalize this data to show the real result value
denormTargetPriceDiffPerc = ((targetPriceDiffPercDl - targetPrice
DiffPercDh)*normTargetPriceDiffPerc - Nh*targetPriceDiffPercDl +
targetPriceDiffPercDh*Nl)/(Nl - Nh);

denormPredictPriceDiffPerc =((targetPriceDiffPercDl - targetPrice
DiffPercDh)*normPredictPriceDiffPerc - Nh*targetPriceDiffPercDl +
targetPriceDiffPercDh*Nl)/(Nl - Nh);

denormInputPriceDiffPercFromRecord = ((inputPriceDiffPercDl - input
PriceDiffPercDh)*normInputPriceDiffPercFromRecord - Nh*input
PriceDiffPercDl + inputPriceDiffPercDh*Nl)/(Nl - Nh);

// Get the price of the 12th element of the row
inputPriceFromFile = arrPrices[rBatchNumber+12];

// Convert denormPredictPriceDiffPerc and denormTargetPriceDiffPerc
// to real de-normalized prices

realDenormTargetPrice = inputPriceFromFile + inputPriceFromFile*
(denormTargetPriceDiffPerc/100);
realDenormPredictPrice = inputPriceFromFile + inputPriceFromFile*
(denormPredictPriceDiffPerc/100);
realDenormTargetToPredictPricePerc = (Math.abs(realDenormTarget
Price - realDenormPredictPrice)/realDenormTargetPrice)*100;

System.out.println("Month = " + (rBatchNumber+1) + "  targetPrice = " +
realDenormTargetPrice + "  predictPrice = " + realDenormPredict
Price + "  diff = " + realDenormTargetToPredictPricePerc);

 if (realDenormTargetToPredictPricePerc > maxGlobalResultDiff)
   {
     maxGlobalResultDiff = realDenormTargetToPredictPricePerc;
   }

sumGlobalResultDiff = sumGlobalResultDiff + realDenormTargetTo
PredictPricePerc;
```

```
        // Populate chart elements
        tempDate = yearDateTraining[rBatchNumber+14];
        //xData.add(tempDate);
        tempMonth = (double) rBatchNumber+14;
        xData.add(tempMonth);
        yData1.add(realDenormTargetPrice);
        yData2.add(realDenormPredictPrice);

      }  // End for Price pair loop

    }  // End of the loop over batches

    XYSeries series1 = Chart.addSeries("Actual price", xData, yData1);
    XYSeries series2 = Chart.addSeries("Predicted price", xData, yData2);

    series1.setLineColor(XChartSeriesColors.BLUE);
    series2.setMarkerColor(Color.ORANGE);
    series1.setLineStyle(SeriesLines.SOLID);
    series2.setLineStyle(SeriesLines.SOLID);

     // Print the max and average results

     averGlobalResultDiff = sumGlobalResultDiff/intNumberOfBatchesToProcess;

     System.out.println(" ");
     System.out.println("maxGlobalResultDiff = " + maxGlobalResultDiff);
     System.out.println("averGlobalResultDiff = " + averGlobalResultDiff);
     System.out.println(" ");

     // Save the chart image
     try
      {
        BitmapEncoder.saveBitmapWithDPI(Chart, strChartFileName,
        BitmapFormat.JPG, 100);
      }
     catch (Exception bt)
      {
        bt.printStackTrace();
      }
```

```
    System.out.println ("Chart and Network have been saved");
    System.out.println("End of validating batches for training");

    returnCodes[0] = 0;
    returnCodes[1] = 0;
    returnCodes[2] = 0;

    return returnCodes;
 }  // End of method

//=====================================================================
// Mode 1. Load the previously saved trained network and process test
   mini-batches
//=====================================================================

static public void loadAndTestNetwork()
 {
   System.out.println("Testing the networks results");

   List<Double> xData = new ArrayList<Double>();
   List<Double> yData1 = new ArrayList<Double>();
   List<Double> yData2 = new ArrayList<Double>();

   double realDenormTargetToPredictPricePerc = 0;
   double maxGlobalResultDiff = 0.00;
   double averGlobalResultDiff = 0.00;
   double sumGlobalResultDiff = 0.00;
   double maxGlobalIndex = 0;

   recordNormInputPriceDiffPerc_00 = 0.00;
   recordNormInputPriceDiffPerc_01 = 0.00;
   recordNormInputPriceDiffPerc_02 = 0.00;
   recordNormInputPriceDiffPerc_03 = 0.00;
   recordNormInputPriceDiffPerc_04 = 0.00;
   recordNormInputPriceDiffPerc_05 = 0.00;
   recordNormInputPriceDiffPerc_06 = 0.00;
   recordNormInputPriceDiffPerc_07 = 0.00;
   recordNormInputPriceDiffPerc_08 = 0.00;
```

```
recordNormInputPriceDiffPerc_09 = 0.00;
recordNormInputPriceDiffPerc_10 = 0.00;
recordNormInputPriceDiffPerc_11 = 0.00;

double recordNormTargetPriceDiffPerc = 0.00;
double normTargetPriceDiffPerc;
double normPredictPriceDiffPerc;
double normInputPriceDiffPercFromRecord;
double denormTargetPriceDiffPerc;
double denormPredictPriceDiffPerc;
double denormInputPriceDiffPercFromRecord;
double realDenormTargetPrice = 0.00;
double realDenormPredictPrice = 0.00;
double minVectorValue = 0.00;
String tempLine;
String[] tempWorkFields;
int tempMinIndex = 0;
double rTempPriceDiffPerc = 0.00;
double rTempKey = 0.00;
double vectorForNetworkRecord = 0.00;
double r_00 = 0.00;
double r_01 = 0.00;
double r_02 = 0.00;
double r_03 = 0.00;
double r_04 = 0.00;
double r_05 = 0.00;
double r_06 = 0.00;
double r_07 = 0.00;
double r_08 = 0.00;
double r_09 = 0.00;
double r_10 = 0.00;
double r_11 = 0.00;
double vectorDiff;
double r1 = 0.00;
double r2 = 0.00;
double vectorForRecord = 0.00;
```

```
int k1 = 0;
int k3 = 0;

BufferedReader br4;
BasicNetwork network;

try
 {
   maxGlobalResultDiff = 0.00;
   averGlobalResultDiff = 0.00;
   sumGlobalResultDiff = 0.00;

   for (k1 = 0; k1 < intNumberOfBatchesToProcess; k1++)
    {
       br4 = new BufferedReader(new FileReader(strTestingFileNames[k1]));
       tempLine = br4.readLine();

       // Skip the label record
       tempLine = br4.readLine();

       // Break the line using comma as separator
       tempWorkFields = tempLine.split(cvsSplitBy);

       recordNormInputPriceDiffPerc_00 = Double.parseDouble(tempWork
       Fields[0]);
       recordNormInputPriceDiffPerc_01 = Double.parseDouble(tempWork
       Fields[1]);
       recordNormInputPriceDiffPerc_02 = Double.parseDouble(tempWork
       Fields[2]);
       recordNormInputPriceDiffPerc_03 = Double.parseDouble(tempWork
       Fields[3]);
       recordNormInputPriceDiffPerc_04 = Double.parseDouble(tempWork
       Fields[4]);
       recordNormInputPriceDiffPerc_05 = Double.parseDouble(tempWork
       Fields[5]);
       recordNormInputPriceDiffPerc_06 = Double.parseDouble(tempWork
       Fields[6]);
       recordNormInputPriceDiffPerc_07 = Double.parseDouble(tempWork
       Fields[7]);
```

```
recordNormInputPriceDiffPerc_08 = Double.parseDouble(tempWork
Fields[8]);
recordNormInputPriceDiffPerc_09 = Double.parseDouble(tempWork
Fields[9]);
recordNormInputPriceDiffPerc_10 = Double.parseDouble(tempWork
Fields[10]);
recordNormInputPriceDiffPerc_11 = Double.parseDouble(tempWork
Fields[11]);

recordNormTargetPriceDiffPerc = Double.parseDouble(tempWork
Fields[12]);

if(k1 < 120)
 {
   // Load the network for the current record
   network = (BasicNetwork)EncogDirectoryPersistence.loadObject
     (newFile(strSaveNetworkFileNames[k1]));

 // Load the training file record
 MLDataSet testingSet = loadCSV2Memory(strTestingFileNames[k1],
 intInputNeuronNumber, intOutputNeuronNumber,true,
 CSVFormat.ENGLISH,false);

// Get the results from the loaded previously saved networks
 int i = - 1;

 for (MLDataPair pair:  testingSet)
  {
     i++;

     MLData inputData = pair.getInput();
     MLData actualData = pair.getIdeal();
     MLData predictData = network.compute(inputData);

     // These values are Normalized as the whole input is
     normTargetPriceDiffPerc = actualData.getData(0);
     normPredictPriceDiffPerc = predictData.getData(0);
     normInputPriceDiffPercFromRecord = inputData.getData(11);
```

```
    // De-normalize this data
    denormTargetPriceDiffPerc = ((targetPriceDiffPercDl -
    targetPriceDiffPercDh)*normTargetPriceDiffPerc - Nh*target
    PriceDiffPercDl + targetPriceDiffPercDh*Nl)/(Nl - Nh);
    denormPredictPriceDiffPerc =((targetPriceDiffPercDl -
    targetPriceDiffPercDh)*normPredictPriceDiffPerc - Nh*
    targetPriceDiffPercDl + targetPriceDiffPercDh*Nl)/(Nl - Nh);

    denormInputPriceDiffPercFromRecord = ((inputPriceDiff
    PercDl - inputPriceDiffPercDh)*normInputPriceDiffPercFrom
    Record - Nh*inputPriceDiffPercDl + inputPriceDiff
    PercDh*Nl)/(Nl - Nh);

    inputPriceFromFile = arrPrices[k1+12];

    // Convert denormPredictPriceDiffPerc and denormTarget
       PriceDiffPerc to a real
    // de-normalize price
    realDenormTargetPrice = inputPriceFromFile + inputPrice
    FromFile*(denormTargetPriceDiffPerc/100);
    realDenormPredictPrice = inputPriceFromFile + inputPrice
    FromFile*(denormPredictPriceDiffPerc/100);

    realDenormTargetToPredictPricePerc = (Math.abs(realDenorm
    TargetPrice - realDenormPredictPrice)/realDenormTarget
    Price)*100;

    System.out.println("Month = " + (k1+1) + " targetPrice = " +
    realDenormTargetPrice + " predictPrice = " + real
    DenormPredictPrice + " diff = " + realDenormTargetTo
    PredictPricePerc);

  }  // End for pair loop

} // End for IF
```

```
else
 {

   vectorForRecord = Math.sqrt(
     Math.pow(recordNormInputPriceDiffPerc_00,2) +
     Math.pow(recordNormInputPriceDiffPerc_01,2) +
     Math.pow(recordNormInputPriceDiffPerc_02,2) +
     Math.pow(recordNormInputPriceDiffPerc_03,2) +
     Math.pow(recordNormInputPriceDiffPerc_04,2) +
     Math.pow(recordNormInputPriceDiffPerc_05,2) +
     Math.pow(recordNormInputPriceDiffPerc_06,2) +
     Math.pow(recordNormInputPriceDiffPerc_07,2) +
     Math.pow(recordNormInputPriceDiffPerc_08,2) +
     Math.pow(recordNormInputPriceDiffPerc_09,2) +
     Math.pow(recordNormInputPriceDiffPerc_10,2) +
     Math.pow(recordNormInputPriceDiffPerc_11,2));

     // Look for the network of previous days that closely
        matches
     // the value of vectorForRecord

     minVectorValue = 999.99;

     for (k3 = 0; k3 < intNumberOfSavedNetworks; k3++)
        {
          r_00 = linkToSaveInputPriceDiffPerc_00[k3];
          r_01 = linkToSaveInputPriceDiffPerc_01[k3];
          r_02 = linkToSaveInputPriceDiffPerc_02[k3];
          r_03 = linkToSaveInputPriceDiffPerc_03[k3];
          r_04 = linkToSaveInputPriceDiffPerc_04[k3];
          r_05 = linkToSaveInputPriceDiffPerc_05[k3];
          r_06 = linkToSaveInputPriceDiffPerc_06[k3];
          r_07 = linkToSaveInputPriceDiffPerc_07[k3];
          r_08 = linkToSaveInputPriceDiffPerc_08[k3];
          r_09 = linkToSaveInputPriceDiffPerc_09[k3];
          r_10 = linkToSaveInputPriceDiffPerc_10[k3];
          r_11 = linkToSaveInputPriceDiffPerc_11[k3];
```

```
            r2 = linkToSaveTargetPriceDiffPerc[k3];

            vectorForNetworkRecord = Math.sqrt(
            Math.pow(r_00,2) +
            Math.pow(r_01,2) +
            Math.pow(r_02,2) +
            Math.pow(r_03,2) +
            Math.pow(r_04,2) +
            Math.pow(r_05,2) +
            Math.pow(r_06,2) +
            Math.pow(r_07,2) +
            Math.pow(r_08,2) +
            Math.pow(r_09,2) +
            Math.pow(r_10,2) +
            Math.pow(r_11,2));

            vectorDiff = Math.abs(vectorForRecord - vectorFor
            NetworkRecord);

            if(vectorDiff < minVectorValue)
             {
               minVectorValue = vectorDiff;

               // Save this network record attributes
               rTempKey = r_00;
               rTempPriceDiffPerc = r2;
               tempMinIndex = k3;
             }

}  // End  FOR k3 loop

network = (BasicNetwork)EncogDirectoryPersistence.loadObject
(newFile(strSaveNetworkFileNames[tempMinIndex]));

 // Now, tempMinIndex points to the corresponding saved network
 // Load this network
 MLDataSet testingSet = loadCSV2Memory(strTestingFileNames[k1],
 intInputNeuronNumber,intOutputNeuronNumber,true,CSVFormat.
 ENGLISH,false);
```

```
// Get the results from the previously saved and  now loaded
   network
int i = - 1;

for (MLDataPair pair:  testingSet)
 {
    i++;

    MLData inputData = pair.getInput();
    MLData actualData = pair.getIdeal();
    MLData predictData = network.compute(inputData);

    // These values are Normalized as the whole input is
    normTargetPriceDiffPerc = actualData.getData(0);
    normPredictPriceDiffPerc = predictData.getData(0);
    normInputPriceDiffPercFromRecord = inputData.getData(11);

    // Renormalize this data to show the real result value
    denormTargetPriceDiffPerc = ((targetPriceDiffPercDl -
    targetPriceDiffPercDh)*normTargetPriceDiffPerc - Nh*
    targetPriceDiffPercDl + targetPriceDiffPercDh*Nl)/
    (Nl - Nh);

    denormPredictPriceDiffPerc =((targetPriceDiffPercDl -
    targetPriceDiffPercDh)*normPredictPriceDiffPerc - Nh*
    targetPriceDiffPercDl + targetPriceDiffPercDh*Nl)/
    (Nl - Nh);

    denormInputPriceDiffPercFromRecord = ((inputPriceDiff
    PercDl - inputPriceDiffPercDh)*normInputPriceDiffPerc
    FromRecord - Nh*inputPriceDiffPercDl + inputPriceDiff
    PercDh*Nl)/(Nl - Nh);

    inputPriceFromFile = arrPrices[k1+12];

    // Convert denormPredictPriceDiffPerc and
       denormTargetPriceDiffPerc to a real
    // demoralize prices
    realDenormTargetPrice = inputPriceFromFile +
    inputPriceFromFile*(denormTargetPriceDiffPerc/100);
```

```
            realDenormPredictPrice = inputPriceFromFile + inputPriceFrom
            File*(denormPredictPriceDiffPerc/100);

            realDenormTargetToPredictPricePerc = (Math.abs(realDenorm
            TargetPrice - realDenormPredictPrice)/realDenormTarget
            Price)*100;

            System.out.println("Month = " + (k1+1) +  "  targetPrice =
            " + realDenormTargetPrice + "  predictPrice = " + real
            DenormPredictPrice + "   diff = " + realDenormTargetTo
            PredictPricePerc);

            if (realDenormTargetToPredictPricePerc > maxGlobal
            ResultDiff)
             {
               maxGlobalResultDiff = realDenormTargetToPredict
               PricePerc;
             }

            sumGlobalResultDiff = sumGlobalResultDiff + realDenorm
            TargetToPredictPricePerc;

          } // End of IF

        }  // End for pair loop

        // Populate chart elements

        tempMonth = (double) k1+14;
        xData.add(tempMonth);
        yData1.add(realDenormTargetPrice);
        yData2.add(realDenormPredictPrice);

     }   // End of loop K1

    // Print the max and average results

    System.out.println(" ");
    System.out.println(" ");
    System.out.println("Results of processing testing batches");
```

```
        averGlobalResultDiff = sumGlobalResultDiff/intNumberOfBatches
        ToProcess;

        System.out.println("maxGlobalResultDiff = " + maxGlobalResultDiff +
        "  i = " + maxGlobalIndex);
        System.out.println("averGlobalResultDiff = " + averGlobalResult
        Diff);
        System.out.println(" ");
        System.out.println(" ");

      }     // End of TRY
     catch (IOException e1)
      {
            e1.printStackTrace();
      }

    // All testing batch files have been processed
      XYSeries series1 = Chart.addSeries("Actual Price", xData, yData1);
      XYSeries series2 = Chart.addSeries("Forecasted Price", xData,
      yData2);

      series1.setLineColor(XChartSeriesColors.BLUE);
      series2.setMarkerColor(Color.ORANGE);
      series1.setLineStyle(SeriesLines.SOLID);
      series2.setLineStyle(SeriesLines.SOLID);

      // Save the chart image
      try
       {
         BitmapEncoder.saveBitmapWithDPI(Chart, strChartFileName,
         BitmapFormat.JPG, 100);
       }
      catch (Exception bt)
       {
         bt.printStackTrace();
       }
```

```
   System.out.println ("The Chart has been saved");
   System.out.println("End of testing for mini-batches training");

} // End of the method

//======================================================================
// Load training dates file in memory
//======================================================================
public static void loadDatesInMemory()
 {
   BufferedReader br1 = null;

   DateFormat sdf = new SimpleDateFormat("yyyy-MM");

   Date dateTemporateDate = null;
   String strTempKeyorateDate;
   int intTemporateDate;

   String line = "";
   String cvsSplitBy = ",";

    try
      {
        br1 = new BufferedReader(new FileReader(strDatesTrainFileName));

       int i = -1;
       int r = -2;

       while ((line = br1.readLine()) != null)
        {
          i++;
          r++;

         // Skip the header line
         if(i > 0)
           {
            // Break the line using comma as separator
            String[] workFields = line.split(cvsSplitBy);
```

```
                strTempKeyorateDate = workFields[0];
                intTemporateDate = Integer.parseInt(strTempKeyorateDate);

                try
                  {
                    dateTemporateDate = convertIntegerToDate(intTemporateDate);
                  }
                catch (ParseException e)
                 {
                   e.printStackTrace();
                   System.exit(1);
                 }

                yearDateTraining[r] = dateTemporateDate;
              }

         }  // end of the while loop

          br1.close();

        }
       catch (IOException ex)
        {
          ex.printStackTrace();
          System.err.println("Error opening files = " + ex);
          System.exit(1);
        }

     }

   //=====================================================================
   // Convert the month date as integer to the Date variable
   //=====================================================================
   public static Date convertIntegerToDate(int denormInputDateI) throws
   ParseException
    {
```

```
    int numberOfYears = denormInputDateI/12;
    int numberOfMonths = denormInputDateI - numberOfYears*12;

    if (numberOfMonths == 0)
     {
       numberOfYears = numberOfYears - 1;
       numberOfMonths = 12;
     }

    String strNumberOfYears = Integer.toString(numberOfYears);

    if(numberOfMonths < 10)
      {
        strNumberOfMonths = Integer.toString(numberOfMonths);
        strNumberOfMonths = "0" + strNumberOfMonths;
      }
     else
      {
        strNumberOfMonths = Integer.toString(numberOfMonths);
      }

     //strYearMonth = "01-" + strNumberOfMonths + "-" + strNumberOfYears +
     "T09:00:00.000Z";
     strYearMonth = strNumberOfYears + "-" + strNumberOfMonths;

     DateFormat sdf = new SimpleDateFormat("yyyy-MM");

     try
      {
       workDate = sdf.parse(strYearMonth);
      }
    catch (ParseException e)
     {
       e.printStackTrace();
     }

   return workDate;

}  // End of method
```

```
//=====================================================================
// Convert the month date as integer to the string strDate variable
//=====================================================================
public static String convertIntegerToString(int denormInputDateI)
 {
    int numberOfYears = denormInputDateI/12;
    int numberOfMonths = denormInputDateI - numberOfYears*12;

    if (numberOfMonths == 0)
     {
       numberOfYears = numberOfYears - 1;
       numberOfMonths = 12;
     }

    String strNumberOfYears = Integer.toString(numberOfYears);

    if(numberOfMonths < 10)
      {
        strNumberOfMonths = Integer.toString(numberOfMonths);
        strNumberOfMonths = "0" + strNumberOfMonths;
      }
     else
      {
        strNumberOfMonths = Integer.toString(numberOfMonths);
      }

     strYearMonth = strNumberOfYears + "-" + strNumberOfMonths;

   return strYearMonth;

}  // End of method

//=====================================================================
// Load Prices file in memory
//=====================================================================
```

```
public static void loadPriceFileInMemory()
 {
   BufferedReader br1 = null;

   String line = "";
   String cvsSplitBy = ",";
   String strTempKeyPrice = "";
   double tempPrice = 0.00;

    try
      {
        br1 = new BufferedReader(new FileReader(strPricesFileName));

       int i = -1;
       int r = -2;

       while ((line = br1.readLine()) != null)
        {
          i++;
          r++;

         // Skip the header line
         if(i > 0)
           {
            // Break the line using comma as separator
            String[] workFields = line.split(cvsSplitBy);

            strTempKeyPrice = workFields[0];
            tempPrice = Double.parseDouble(strTempKeyPrice);
            arrPrices[r] = tempPrice;

         }

     }  // end of the while loop

      br1.close();

    }
```

```
        catch (IOException ex)
         {
             ex.printStackTrace();
             System.err.println("Error opening files = " + ex);
             System.exit(1);
         }

     }

 } // End of the  Encog class
```

Training Process

For the most part, the training method logic is similar to what was used in the preceding examples, so it does not need any explanation, with the exception of one part that I will discuss here.

Sometimes you have to deal with functions that have very small values, so the calculated errors are even smaller. For example, the network errors can reach microscopic values such as 14 or more zeros after the dot, as in 0.000000000000025. When you get such errors, you will start questioning the accuracy of the calculation. In this code, I have included an example of how to handle such a situation.

Instead of simply calling the `train.getError()` method to determine the network error, you use a pair data set to retrieve the input, actual, and predicted function values from the network for each epoch; denormalize those values; and calculate the error percent difference between the calculated and actual values. You then exit from the pair loop with a `returnCode` value of 0 when this difference is less than the error limit. This is shown in Listing 11-4.

Listing 11-4. Checking the Error Using the Actual Function Values

```
int epoch = 1;
double tempLastErrorPerc = 0.00;

 do
      {
          train.iteration();

          epoch++;
```

```
for (MLDataPair pair1:  trainingSet)
  {
        MLData inputData = pair1.getInput();
        MLData actualData = pair1.getIdeal();
        MLData predictData = network.compute(inputData);

          // These values are Normalized as the whole input is
          normTargetPriceDiffPerc = actualData.getData(0);
          normPredictPriceDiffPerc = predictData.getData(0);

          denormTargetPriceDiffPerc = ((targetPriceDiffPercDl -
          targetPriceDiffPercDh)*normTargetPriceDiffPerc - Nh*target
          PriceDiffPercDl + targetPriceDiffPercDh*Nl)/(Nl - Nh);

          denormPredictPriceDiffPerc =((targetPriceDiffPercDl -
          targetPriceDiffPercDh)*normPredictPriceDiffPerc  - Nh*
          targetPriceDiffPercDl + targetPriceDiffPercDh*Nl)/(Nl - Nh);

          inputPriceFromFile = arrPrices[rBatchNumber+12];

          realDenormTargetPrice = inputPriceFromFile + inputPriceFrom
          File*denormTargetPriceDiffPerc/100;

          realDenormPredictPrice = inputPriceFromFile + inputPriceFrom
          File*denormPredictPriceDiffPerc/100;

          realDenormTargetToPredictPricePerc = (Math.abs(realDenorm
          TargetPrice - realDenormPredictPrice)/realDenormTarget
          Price)*100;
       }

   if (epoch >= 500 && realDenormTargetToPredictPricePerc > 0.00091)
     {
        returnCodes[0] = 1;
        returnCodes[1] = rBatchNumber;
        returnCodes[2] = intDayNumber-1;

        return returnCodes;
     }

  } while(realDenormTargetToPredictPricePerc >  0.0009);
```

Training Results

Listing 11-5 shows the training results.

Listing 11-5. Training Results

```
Month =  1  targetPrice = 1239.94000  predictPrice = 1239.93074
diff = 7.46675E-4
Month =  2  targetPrice = 1160.33000  predictPrice = 1160.32905
diff = 8.14930E-5
Month =  3  targetPrice = 1249.46000  predictPrice = 1249.44897
diff = 8.82808E-4
Month =  4  targetPrice = 1255.82000  predictPrice = 1255.81679
diff = 2.55914E-4
Month =  5  targetPrice = 1224.38000  predictPrice = 1224.37483
diff = 4.21901E-4
Month =  6  targetPrice = 1211.23000  predictPrice = 1211.23758
diff = 6.25530E-4
Month =  7  targetPrice = 1133.58000  predictPrice = 1133.59013
diff = 8.94046E-4
Month =  8  targetPrice = 1040.94000  predictPrice = 1040.94164
diff = 1.57184E-4
Month =  9  targetPrice = 1059.78000  predictPrice = 1059.78951
diff = 8.97819E-4
Month = 10  targetPrice = 1139.45000  predictPrice = 1139.45977
diff = 8.51147E-4
Month = 11  targetPrice = 1148.08000  predictPrice = 1148.07912
diff = 7.66679E-5
Month = 12  targetPrice = 1130.20000  predictPrice = 1130.20593
diff = 5.24564E-4
Month = 13  targetPrice = 1106.73000  predictPrice = 1106.72654
diff = 3.12787E-4
Month = 14  targetPrice = 1147.39000  predictPrice = 1147.39283
diff = 2.46409E-4
Month = 15  targetPrice = 1076.92000  predictPrice = 1076.92461
diff = 4.28291E-4
```

```
Month = 16  targetPrice = 1067.14000  predictPrice = 1067.14948
diff = 8.88156E-4
Month = 17  targetPrice = 989.819999  predictPrice = 989.811316
diff = 8.77328E-4
Month = 18  targetPrice = 911.620000  predictPrice = 911.625389
diff = 5.91142E-4
Month = 19  targetPrice = 916.070000  predictPrice = 916.071216
diff = 1.32725E-4
Month = 20  targetPrice = 815.280000  predictPrice = 815.286704
diff = 8.22304E-4
Month = 21  targetPrice = 885.760000  predictPrice = 885.767730
diff = 8.72729E-4
Month = 22  targetPrice = 936.310000  predictPrice = 936.307290
diff = 2.89468E-4
Month = 23  targetPrice = 879.820000  predictPrice = 879.812595
diff = 8.41647E-4
Month = 24  targetPrice = 855.700000  predictPrice = 855.700307
diff = 3.58321E-5
Month = 25  targetPrice = 841.150000  predictPrice = 841.157407
diff = 8.80559E-4
Month = 26  targetPrice = 848.180000  predictPrice = 848.177279
diff = 3.22296E-4
Month = 27  targetPrice = 916.920000  predictPrice = 916.914394
diff = 6.11352E-4
Month = 28  targetPrice = 963.590000  predictPrice = 963.591678
diff = 1.74172E-4
Month = 29  targetPrice = 974.500000  predictPrice = 974.505665
diff = 5.81287E-4
Month = 30  targetPrice = 990.310000  predictPrice = 990.302895
diff = 7.17406E-4
Month = 31  targetPrice = 1008.01000  predictPrice = 1008.00861
diff = 1.37856E-4
Month = 32  targetPrice = 995.970000  predictPrice = 995.961734
diff = 8.29902E-4
Month = 33  targetPrice = 1050.71000  predictPrice = 1050.70954
diff = 4.42062E-5
```

```
Month = 34  targetPrice = 1058.20000  predictPrice = 1058.19690
diff = 2.93192E-4
Month = 35  targetPrice = 1111.92000  predictPrice = 1111.91406
diff = 5.34581E-4
Month = 36  targetPrice = 1131.13000  predictPrice = 1131.12351
diff = 5.73549E-4
Month = 37  targetPrice = 1144.94000  predictPrice = 1144.94240
diff = 2.09638E-4
Month = 38  targetPrice = 1126.21000  predictPrice = 1126.21747
diff = 6.63273E-4
Month = 39  targetPrice = 1107.30000  predictPrice = 1107.30139
diff = 1.25932E-4
Month = 40  targetPrice = 1120.68000  predictPrice = 1120.67926
diff = 6.62989E-5
Month = 41  targetPrice = 1140.84000  predictPrice = 1140.83145
diff = 7.49212E-4
Month = 42  targetPrice = 1101.72000  predictPrice = 1101.72597
diff = 5.42328E-4
Month = 43  targetPrice = 1104.24000  predictPrice = 1104.23914
diff = 7.77377E-5
Month = 44  targetPrice = 1114.58000  predictPrice = 1114.58307
diff = 2.75127E-4
Month = 45  targetPrice = 1130.20000  predictPrice = 1130.19238
diff = 6.74391E-4
Month = 46  targetPrice = 1173.82000  predictPrice = 1173.82891
diff = 7.58801E-4
Month = 47  targetPrice = 1211.92000  predictPrice = 1211.92000
diff = 4.97593E-7
Month = 48  targetPrice = 1181.27000  predictPrice = 1181.27454
diff = 3.84576E-4
Month = 49  targetPrice = 1203.60000  predictPrice = 1203.60934
diff = 7.75922E-4
Month = 50  targetPrice = 1180.59000  predictPrice = 1180.60006
diff = 8.51986E-4
Month = 51  targetPrice = 1156.85000  predictPrice = 1156.85795
diff = 6.87168E-4
```

```
Month = 52  targetPrice = 1191.50000  predictPrice = 1191.50082
diff = 6.89121E-5
Month = 53  targetPrice = 1191.32000  predictPrice = 1191.32780
diff = 1.84938E-4
Month = 54  targetPrice = 1234.18000  predictPrice = 1234.18141
diff = 1.14272E-4
Month = 55  targetPrice = 1220.33000  predictPrice = 1220.33276
diff = 2.26146E-4
Month = 56  targetPrice = 1228.81000  predictPrice = 1228.80612
diff = 3.15986E-4
Month = 57  targetPrice = 1207.01000  predictPrice = 1207.00419
diff = 4.81617E-4
Month = 58  targetPrice = 1249.48000  predictPrice = 1249.48941
diff = 7.52722E-4
Month = 59  targetPrice = 1248.29000  predictPrice = 1248.28153
diff = 6.78199E-4
Month = 60  targetPrice = 1280.08000  predictPrice = 1280.07984
diff = 1.22483E-5
Month = 61  targetPrice = 1280.66000  predictPrice = 1280.66951
diff = 7.42312E-4
Month = 62  targetPrice = 1294.87000  predictPrice = 1294.86026
diff = 7.51869E-4
Month = 63  targetPrice = 1310.61000  predictPrice = 1310.60544
diff = 3.48001E-4
Month = 64  targetPrice = 1270.09000  predictPrice = 1270.08691
diff = 2.43538E-4
Month = 65  targetPrice = 1270.20000  predictPrice = 1270.19896
diff = 8.21560E-5
Month = 66  targetPrice = 1276.66000  predictPrice = 1276.66042
diff = 3.26854E-5
Month = 67  targetPrice = 1303.82000  predictPrice = 1303.82874
diff = 6.70418E-4
Month = 68  targetPrice = 1335.85000  predictPrice = 1335.84632
diff = 2.75638E-4
Month = 69  targetPrice = 1377.94000  predictPrice = 1377.94691
diff = 5.01556E-4
```

```
Month = 70  targetPrice = 1400.63000  predictPrice = 1400.63379
diff = 2.70408E-4
Month = 71  targetPrice = 1418.30000  predictPrice = 1418.31183
diff = 8.34099E-4
Month = 72  targetPrice = 1438.24000  predictPrice = 1438.24710
diff = 4.93547E-4
Month = 73  targetPrice = 1406.82000  predictPrice = 1406.81500
diff = 3.56083E-4
Month = 74  targetPrice = 1420.86000  predictPrice = 1420.86304
diff = 2.13861E-4
Month = 75  targetPrice = 1482.37000  predictPrice = 1482.37807
diff = 5.44135E-4
Month = 76  targetPrice = 1530.62000  predictPrice = 1530.60780
diff = 7.96965E-4
Month = 77  targetPrice = 1503.35000  predictPrice = 1503.35969
diff = 6.44500E-4
Month = 78  targetPrice = 1455.27000  predictPrice = 1455.25870
diff = 7.77012E-4
Month = 79  targetPrice = 1473.99000  predictPrice = 1474.00301
diff = 8.82764E-4
Month = 80  targetPrice = 1526.75000  predictPrice = 1526.74507
diff = 3.23149E-4
Month = 81  targetPrice = 1549.38000  predictPrice = 1549.38480
diff = 3.10035E-4
Month = 82  targetPrice = 1481.14000  predictPrice = 1481.14819
diff = 5.52989E-4
Month = 83  targetPrice = 1468.36000  predictPrice = 1468.34730
diff = 8.64876E-4
Month = 84  targetPrice = 1378.55000  predictPrice = 1378.53761
diff = 8.98605E-4
Month = 85  targetPrice = 1330.63000  predictPrice = 1330.64177
diff = 8.84310E-4
Month = 86  targetPrice = 1322.70000  predictPrice = 1322.71089
diff = 8.23113E-4
Month = 87  targetPrice = 1385.59000  predictPrice = 1385.58259
diff = 5.34831E-4
```

```
Month = 88  targetPrice = 1400.38000  predictPrice = 1400.36749
diff = 8.93019E-4
Month = 89  targetPrice = 1279.99999  predictPrice = 1279.98926
diff = 8.38844E-4
Month = 90  targetPrice = 1267.38     predictPrice = 1267.39112
diff = 8.77235E-4
Month = 91  targetPrice = 1282.83000  predictPrice = 1282.82564
diff = 3.40160E-4
Month = 92  targetPrice = 1166.36000  predictPrice = 1166.35838
diff = 1.38537E-4
Month = 93  targetPrice = 968.750000  predictPrice = 968.756639
diff = 6.85325E-4
Month = 94  targetPrice = 896.24000   predictPrice = 896.236238
diff = 4.19700E-4
Month = 95  targetPrice = 903.250006  predictPrice = 903.250891
diff = 9.86647E-5
Month = 96  targetPrice = 825.880000  predictPrice = 825.877467
diff = 3.06702E-4
Month = 97  targetPrice = 735.090000  predictPrice = 735.089888
diff = 1.51705E-5
Month = 98  targetPrice = 797.870000  predictPrice = 797.864377
diff = 7.04777E-4
Month = 99  targetPrice = 872.810000  predictPrice = 872.817137
diff = 8.17698E-4
Month = 100 targetPrice = 919.14000   predictPrice = 919.144707
diff = 5.12104E-4
Month = 101 targetPrice = 919.32000   predictPrice = 919.311948
diff = 8.75905E-4
Month = 102 targetPrice = 987.48000   predictPrice = 987.485732
diff = 5.80499E-4
Month = 103 targetPrice = 1020.6200   predictPrice = 1020.62163
diff = 1.60605E-4
Month = 104 targetPrice = 1057.0800   predictPrice = 1057.07122
diff = 8.30374E-4
Month = 105 targetPrice = 1036.1900   predictPrice = 1036.18940
diff = 5.79388E-5
```

```
Month = 106 targetPrice = 1095.6300   predictPrice = 1095.63936
diff = 8.54512E-4
Month = 107 targetPrice = 1115.1000   predictPrice = 1115.09792
diff = 1.86440E-4
Month = 108 targetPrice = 1073.8700   predictPrice = 1073.87962
diff = 8.95733E-4
Month = 109 targetPrice = 1104.4900   predictPrice = 1104.48105
diff = 8.10355E-4
Month = 110 targetPrice = 1169.4300   predictPrice = 1169.42384
diff = 5.26459E-4
Month = 111 targetPrice = 1186.6900   predictPrice = 1186.68972
diff = 2.39657E-5
Month = 112 targetPrice = 1089.4100   predictPrice = 1089.40111
diff = 8.16044E-4
Month = 113 targetPrice = 1030.7100   predictPrice = 1030.71574
diff = 5.57237E-4
Month = 114 targetPrice = 1101.6000   predictPrice = 1101.59105
diff = 8.12503E-4
Month = 115 targetPrice = 1049.3300   predictPrice = 1049.32154
diff = 8.06520E-4
Month = 116 targetPrice = 1141.2000   predictPrice = 1141.20704
diff = 6.1701E-4
Month = 117 targetPrice = 1183.2600   predictPrice = 1183.27030
diff = 8.705E-4
Month = 118 targetPrice = 1180.5500   predictPrice = 1180.54438
diff = 4.763E-4
Month = 119 targetPrice = 1257.6400   predictPrice = 1257.63292
diff = 5.628E-4
Month = 120 targetPrice = 1286.1200   predictPrice = 1286.11021
diff = 7.608E-4

maxErrorPerc = 7.607871107092592E-4
averErrorPerc = 6.339892589243827E-6
```

The log shows that because of the use of the micro-batch method, the approximation results for this noncontinuous function are pretty good.

```
maxErrorDifferencePerc <  0.000761% and averErrorDifferencePerc < 0.00000634%
```

Figure 11-4 shows the chart of the training/validating results.

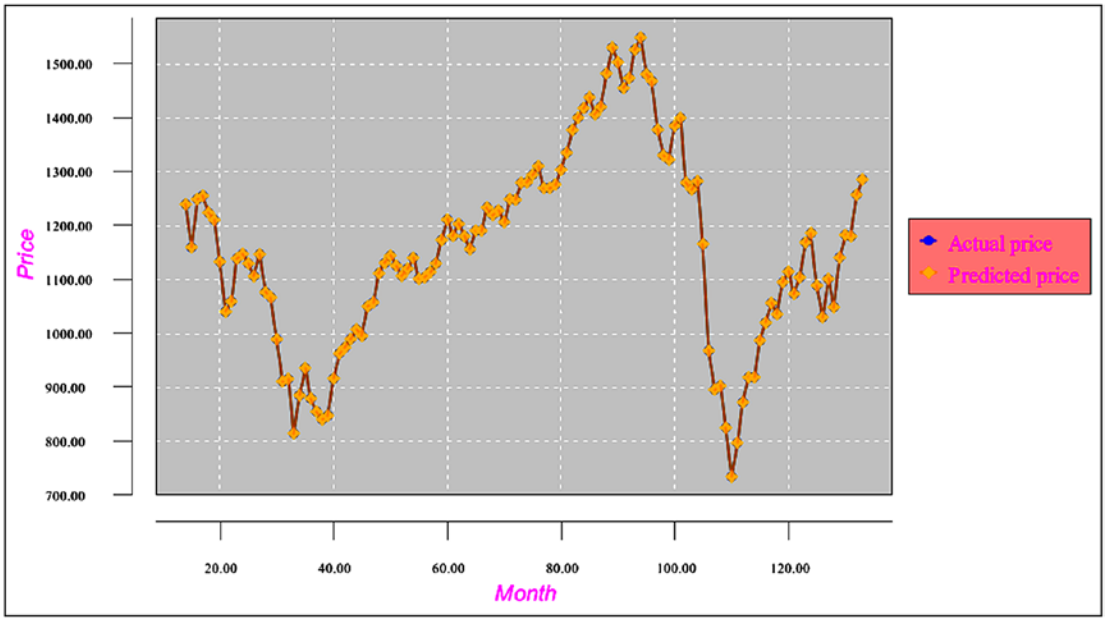


Figure 11-4. *Chart of the training results*

Testing Data Set

The testing data set has the same format as the training data set. As mentioned at the beginning of this example, the goal is to predict the market price for the next month, based on the ten-year historical data. Therefore, the testing data set is the same as the training data set, but it should include at the end one extra micro-batch record, which will be used for next month's price prediction (outside of the network training range). Table 11-5 shows a fragment of the price difference testing data set.

Table 11-5. *Fragment of the Price Difference Testing Data Set*

priceDiffPerc	targetPriceDiffPerc	Date	inputPrice
5.840553677	5.857688372	199704	801.34
5.857688372	4.345263356	199705	848.28
4.345263356	7.814583004	199706	885.14
7.814583004	-5.746560342	199707	954.31
-5.746560342	5.315352374	199708	899.47
5.315352374	-3.447766236	199709	947.28
-3.447766236	4.458682294	199710	914.62
4.458682294	1.573163073	199711	955.4
1.573163073	1.015013963	199712	970.43
1.015013963	7.04492594	199801	980.28
7.04492594	4.994568014	199802	1049.34
4.994568014	0.907646925	199803	1101.75
0.907646925	-1.882617495	199804	1111.75
-1.882617495	3.943822079	199805	1090.82
3.943822079	-1.161539547	199806	1133.84
-1.161539547	-14.57967109	199807	1120.67
-14.57967109	6.239553736	199808	957.28
6.239553736	8.029419573	199809	1017.01
8.029419573	5.91260342	199810	1098.67
5.91260342	5.63753083	199811	1163.63
5.63753083	4.10094124	199812	1229.23
4.10094124	-3.228251696	199901	1279.64
-3.228251696	3.879418249	199902	1238.33
3.879418249	3.79439819	199903	1286.37
3.79439819	-2.497041597	199904	1335.18

(*continued*)

Table 11-5. (*continued*)

priceDiffPerc	targetPriceDiffPerc	Date	inputPrice
-2.497041597	5.443833344	199905	1301.84
5.443833344	-3.204609859	199906	1372.71
-3.204609859	-0.625413932	199907	1328.72
-0.625413932	-2.855173772	199908	1320.41
-2.855173772	6.253946722	199909	1282.71

Table 11-6 shows a fragment of the normalized testing data set.

Table 11-6. *Fragment of the Normalized Testing Data Set*

priceDiffPerc	targetPriceDiffPerc	Date	inputPrice
0.722703578	0.723845891	199704	801.34
0.723845891	0.623017557	199705	848.28
0.623017557	0.854305534	199706	885.14
0.854305534	-0.049770689	199707	954.31
-0.049770689	0.687690158	199708	899.47
0.687690158	0.103482251	199709	947.28
0.103482251	0.63057882	199710	914.62
0.63057882	0.438210872	199711	955.4
0.438210872	0.401000931	199712	970.43
0.401000931	0.802995063	199801	980.28
0.802995063	0.666304534	199802	1049.34
0.666304534	0.393843128	199803	1101.75
0.393843128	0.2078255	199804	1111.75
0.2078255	0.596254805	199805	1090.82
0.596254805	0.255897364	199806	1133.84

(*continued*)

Table 11-6. (*continued*)

priceDiffPerc	targetPriceDiffPerc	Date	inputPrice
0.255897364	-0.638644739	199807	1120.67
-0.638644739	0.749303582	199808	957.28
0.749303582	0.868627972	199809	1017.01
0.868627972	0.727506895	199810	1098.67
0.727506895	0.709168722	199811	1163.63
0.709168722	0.606729416	199812	1229.23
0.606729416	0.118116554	199901	1279.64
0.118116554	0.591961217	199902	1238.33
0.591961217	0.586293213	199903	1286.37
0.586293213	0.166863894	199904	1335.18
0.166863894	0.696255556	199905	1301.84
0.696255556	0.119692676	199906	1372.71
0.119692676	0.291639071	199907	1328.72
0.291639071	0.142988415	199908	1320.41
0.142988415	0.750263115	199909	1282.71

Finally, Table 11-7 shows the sliding window testing data set. This is the data set used to test the trained network.

Table 11-7. *Fragment of the Sliding Window Testing Data Set*

Sliding Windows												
0.591	0.55	0.17	0.46	0.21	0.2	0.53	0.3	0.57	0.26	0.4	0.22	0.327
0.55	0.165	0.46	0.21	0.2	0.53	0.33	0.6	0.26	0.38	0.2	0.57	0.503
0.165	0.459	0.21	0.2	0.53	0.33	0.57	0.3	0.38	0.22	0.6	0.33	0.336
0.459	0.206	0.2	0.53	0.33	0.57	0.26	0.4	0.22	0.57	0.3	0.5	0.407

(*continued*)

Table 11-7. (*continued*)

Sliding Windows												
0.206	0.199	0.53	0.33	0.57	0.26	0.38	0.2	0.57	0.33	0.5	0.34	0.414
0.199	0.533	0.33	0.57	0.26	0.38	0.22	0.6	0.33	0.5	0.3	0.41	0.127
0.533	0.332	0.57	0.26	0.38	0.22	0.57	0.3	0.5	0.34	0.4	0.41	0.334
0.332	0.573	0.26	0.38	0.22	0.57	0.33	0.5	0.34	0.41	0.4	0.13	0.367
0.573	0.259	0.38	0.22	0.57	0.33	0.5	0.3	0.41	0.41	0.1	0.33	0.475
0.259	0.38	0.22	0.57	0.33	0.5	0.34	0.4	0.41	0.13	0.3	0.37	0.497
0.38	0.215	0.57	0.33	0.5	0.34	0.41	0.4	0.13	0.33	0.4	0.48	0.543
0.215	0.568	0.33	0.5	0.34	0.41	0.41	0.1	0.33	0.37	0.5	0.5	0.443
0.568	0.327	0.5	0.34	0.41	0.41	0.13	0.3	0.37	0.48	0.5	0.54	0.417
0.327	0.503	0.34	0.41	0.41	0.13	0.33	0.4	0.48	0.5	0.5	0.44	0.427
0.503	0.336	0.41	0.41	0.13	0.33	0.37	0.5	0.5	0.54	0.4	0.42	0.188
0.336	0.407	0.41	0.13	0.33	0.37	0.48	0.5	0.54	0.44	0.4	0.43	0.4
0.407	0.414	0.13	0.33	0.37	0.48	0.5	0.5	0.44	0.42	0.4	0.19	0.622
0.414	0.127	0.33	0.37	0.48	0.5	0.54	0.4	0.42	0.43	0.2	0.4	0.55
0.127	0.334	0.37	0.48	0.5	0.54	0.44	0.4	0.43	0.19	0.4	0.62	0.215
0.334	0.367	0.48	0.5	0.54	0.44	0.42	0.4	0.19	0.4	0.6	0.55	0.12
0.367	0.475	0.5	0.54	0.44	0.42	0.43	0.2	0.4	0.62	0.6	0.22	0.419
0.475	0.497	0.54	0.44	0.42	0.43	0.19	0.4	0.62	0.55	0.2	0.12	0.572
0.497	0.543	0.44	0.42	0.43	0.19	0.4	0.6	0.55	0.22	0.1	0.42	0.432
0.543	0.443	0.42	0.43	0.19	0.4	0.62	0.6	0.22	0.12	0.4	0.57	0.04
0.443	0.417	0.43	0.19	0.4	0.62	0.55	0.2	0.12	0.42	0.6	0.43	0.276
0.417	0.427	0.19	0.4	0.62	0.55	0.22	0.1	0.42	0.57	0.4	0.04	-0.074
0.427	0.188	0.4	0.62	0.55	0.22	0.12	0.4	0.57	0.43	0	0.28	0.102
0.188	0.4	0.62	0.55	0.22	0.12	0.42	0.6	0.43	0.04	0.3	-0.1	0.294
0.4	0.622	0.55	0.22	0.12	0.42	0.57	0.4	0.04	0.28	-0	0.1	0.65
0.622	0.55	0.22	0.12	0.42	0.57	0.43	0	0.28	-0.07	0.1	0.29	0.404

The sliding window testing data set is broken in micro-batch files. Figure 11-5 shows a fragment of the list of testing micro-batch files.

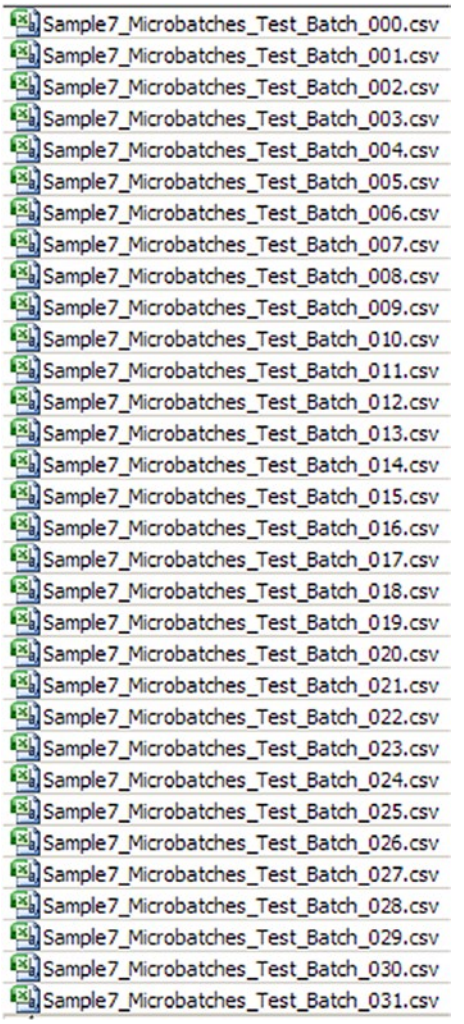


Figure 11-5. *Fragment of the list of testing micro-batch data sets*

Testing Logic

There are many new coding fragments in this method, so let's discuss them. You load the micro-batch data set and the corresponding saved network in a loop over the set of testing micro-batch data sets. Remember, you no longer process a single testing data set but a set of micro-batch testing data sets. Next, you obtain from the

network the input, actual, and predicted price values; normalize them; and calculate the actual and predicted prices. That is done for all test records for which the saved-network records exist.

However, there is no save-network file for the last micro-batch record in the test data set, simply because the network was not trained for that point. For this record you retrieve its 12 `inputPriceDiffPerc` fields, which are the keys used during network training. Next, you search the keys of all saved networks files that are located in the memory arrays called `linkToSaveInputPriceDiffPerc_00`, `linkToSaveInputPriceDiffPerc_01`, and so on.

Because there are 12 keys associated with each saved network, the search is done in the following way. For a micro-batch to be processed, you calculate the vector value in the 12D space using Euclidean geometry. For example, for the function of 12 variables y = f(x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12), the vector value is the square root of the sum of each x value powered to 2 (see 10-1).

$$\sqrt{x1^2 + x1^2 + x1^2 + x1^2 + x1^2 + x1^2} + x1^2 + x1^2 + x1^2 + x1^2 + x1^2 + x1^2 \qquad (11\text{-}1)$$

Then, for each set of network keys held in the `linkToSaveInputPriceDiffPerc` arrays, the vector value is also calculated. The network keys that closely match the set of keys from the processed record are selected and loaded into memory. Finally, you obtain from that network the input, active, and predicted values; denormalize them; and calculate the actual and predicted values. Listing 11-6 shows the code for this logic.

Listing 11-6. The Logic of Selecting the Saved-Network Record

```
static public void loadAndTestNetwork()
 {

    List<Double> xData = new ArrayList<Double>();
    List<Double> yData1 = new ArrayList<Double>();
    List<Double> yData2 = new ArrayList<Double>();

    int k1 = 0;
    int k3 = 0;

    BufferedReader br4;
    BasicNetwork network;

    try
```

```
{
  // Process testing batches

  maxGlobalResultDiff = 0.00;
  averGlobalResultDiff = 0.00;
  sumGlobalResultDiff = 0.00;

  for (k1 = 0; k1 < intNumberOfBatchesToProcess; k1++)
   {
      br4 = new BufferedReader(new FileReader(strTestingFile
      Names[k1]));
      tempLine = br4.readLine();

      // Skip the label record
      tempLine = br4.readLine();

      // Break the line using comma as separator
      tempWorkFields = tempLine.split(cvsSplitBy);

      recordNormInputPriceDiffPerc_00 = Double.parseDouble(tempWork
      Fields[0]);
      recordNormInputPriceDiffPerc_01 = Double.parseDouble(tempWork
      Fields[1]);
      recordNormInputPriceDiffPerc_02 = Double.parseDouble(tempWork
      Fields[2]);
      recordNormInputPriceDiffPerc_03 = Double.parseDouble(tempWork
      Fields[3]);
      recordNormInputPriceDiffPerc_04 = Double.parseDouble(tempWork
      Fields[4]);
      recordNormInputPriceDiffPerc_05 = Double.parseDouble(tempWork
      Fields[5]);
      recordNormInputPriceDiffPerc_06 = Double.parseDouble(tempWork
      Fields[6]);
      recordNormInputPriceDiffPerc_07 = Double.parseDouble(tempWork
      Fields[7]);
      recordNormInputPriceDiffPerc_08 = Double.parseDouble(tempWork
      Fields[8]);
```

```
recordNormInputPriceDiffPerc_09 = Double.parseDouble(tempWork
Fields[9]);
recordNormInputPriceDiffPerc_10 = Double.parseDouble(tempWork
Fields[10]);
recordNormInputPriceDiffPerc_11 = Double.parseDouble(tempWork
Fields[11]);

recordNormTargetPriceDiffPerc = Double.parseDouble(tempWork
Fields[12]);


if(k1 < 120)
 {
   // Load the network for the current record
   network = (BasicNetwork)EncogDirectoryPersistence. loadObject
   (newFile(strSaveNetworkFileNames[k1]));

 // Load the training file record
 MLDataSet testingSet = loadCSV2Memory(strTestingFileNames[k1],
 intInputNeuronNumber, intOutputNeuronNumber,true,
 CSVFormat.ENGLISH,false);

// Get the results from the loaded previously saved networks
 int i = - 1; // Index of the array to get results

 for (MLDataPair pair:  testingSet)
  {
     i++;

     MLData inputData = pair.getInput();
     MLData actualData = pair.getIdeal();
     MLData predictData = network.compute(inputData);

     // These values are Normalized as the whole input is
     normTargetPriceDiffPerc = actualData.getData(0);
     normPredictPriceDiffPerc = predictData.getData(0);
     normInputPriceDiffPercFromRecord = inputData.getData(11);
```

```
        // De-normalize this data to show the real result value
        denormTargetPriceDiffPerc = ((targetPriceDiffPercDl -
        targetPriceDiffPercDh)*normTargetPriceDiffPerc- Nh*target
        PriceDiffPercDl + targetPriceDiffPercDh*Nl)/(Nl - Nh);
        denormPredictPriceDiffPerc =((targetPriceDiffPercDl -
        targetPriceDiffPercDh)*normPredictPriceDiffPerc - Nh*
        targetPriceDiffPercDl + targetPriceDiffPercDh*Nl)/(Nl - Nh);

        denormInputPriceDiffPercFromRecord = ((inputPriceDiff
        PercDl - inputPriceDiffPercDh)*normInputPriceDiffPerc
        FromRecord - Nh*inputPriceDiffPercDl + inputPriceDiff
        PercDh*Nl)/(Nl - Nh);

        inputPriceFromFile = arrPrices[k1+12];

        // Convert denormPredictPriceDiffPerc and denormTarget
           PriceDiffPerc to real renormalized
        // price

        realDenormTargetPrice = inputPriceFromFile +
        inputPriceFromFile*(denormTargetPriceDiffPerc/100);
        realDenormPredictPrice = inputPriceFromFile +
        inputPriceFromFile*(denormPredictPriceDiffPerc/100);

        realDenormTargetToPredictPricePerc = (Math.abs(realDenorm
        TargetPrice - realDenormPredictPrice)/realDenorm
        TargetPrice)*100;

        System.out.println("Month = " + (k1+1) +  "  targetPrice =
        " + realDenormTargetPrice + "  predictPrice = " + real
        DenormPredictPrice + "   diff = " + realDenormTarget
        ToPredictPricePerc);

      }  // End of the for pair loop

    } // End for IF
```

```
else
 {
   vectorForRecord = Math.sqrt(
     Math.pow(recordNormInputPriceDiffPerc_00,2) +
     Math.pow(recordNormInputPriceDiffPerc_01,2) +
     Math.pow(recordNormInputPriceDiffPerc_02,2) +
     Math.pow(recordNormInputPriceDiffPerc_03,2) +
     Math.pow(recordNormInputPriceDiffPerc_04,2) +
     Math.pow(recordNormInputPriceDiffPerc_05,2) +
     Math.pow(recordNormInputPriceDiffPerc_06,2) +
     Math.pow(recordNormInputPriceDiffPerc_07,2) +
     Math.pow(recordNormInputPriceDiffPerc_08,2) +
     Math.pow(recordNormInputPriceDiffPerc_09,2) +
     Math.pow(recordNormInputPriceDiffPerc_10,2) +
     Math.pow(recordNormInputPriceDiffPerc_11,2));

     // Look for the network of previous months that closely
        match the
     // vectorForRecord value

     minVectorValue = 999.99;

     for (k3 = 0; k3 < intNumberOfSavedNetworks; k3++)
        {
          r_00 = linkToSaveInputPriceDiffPerc_00[k3];
          r_01 = linkToSaveInputPriceDiffPerc_01[k3];
          r_02 = linkToSaveInputPriceDiffPerc_02[k3];
          r_03 = linkToSaveInputPriceDiffPerc_03[k3];
          r_04 = linkToSaveInputPriceDiffPerc_04[k3];
          r_05 = linkToSaveInputPriceDiffPerc_05[k3];
          r_06 = linkToSaveInputPriceDiffPerc_06[k3];
          r_07 = linkToSaveInputPriceDiffPerc_07[k3];
          r_08 = linkToSaveInputPriceDiffPerc_08[k3];
          r_09 = linkToSaveInputPriceDiffPerc_09[k3];
          r_10 = linkToSaveInputPriceDiffPerc_10[k3];
          r_11 = linkToSaveInputPriceDiffPerc_11[k3];

          r2 = linkToSaveTargetPriceDiffPerc[k3];
```

```
                vectorForNetworkRecord = Math.sqrt(
                Math.pow(r_00,2) +
                Math.pow(r_01,2) +
                Math.pow(r_02,2) +
                Math.pow(r_03,2) +
                Math.pow(r_04,2) +
                Math.pow(r_05,2) +
                Math.pow(r_06,2) +
                Math.pow(r_07,2) +
                Math.pow(r_08,2) +
                Math.pow(r_09,2) +
                Math.pow(r_10,2) +
                Math.pow(r_11,2));

                vectorDiff = Math.abs(vectorForRecord - vectorFor
                NetworkRecord);

                if(vectorDiff < minVectorValue)
                 {
                   minVectorValue = vectorDiff;

                   // Save this network record attributes
                   rTempKey = r_00;
                   rTempPriceDiffPerc = r2;
                   tempMinIndex = k3;
                 }

        }  // End  FOR k3 loop

        network =
        (BasicNetwork)EncogDirectoryPersistence.loadObject(newFile
        (strSaveNetworkFileNames[tempMinIndex]));

         // Now, tempMinIndex points to the corresponding saved network
         // Load this network in memory
```

```
MLDataSet testingSet = loadCSV2Memory(strTestingFileNames[k1],
intInputNeuronNumber,intOutputNeuronNumber,true,
CSVFormat.ENGLISH,false);

// Get the results from the loaded network
int i = - 1;

for (MLDataPair pair:  testingSet)
 {
    i++;

    MLData inputData = pair.getInput();
    MLData actualData = pair.getIdeal();
    MLData predictData = network.compute(inputData);

    // These values are Normalized as the whole input is
    normTargetPriceDiffPerc = actualData.getData(0);
    normPredictPriceDiffPerc = predictData.getData(0);
    normInputPriceDiffPercFromRecord = inputData.getData(11);

    // Renormalize this data to show the real result value
    denormTargetPriceDiffPerc = ((targetPriceDiffPercDl -
    targetPriceDiffPercDh)*normTargetPriceDiffPerc - Nh*target
    PriceDiffPercDl + targetPriceDiffPercDh*Nl)/(Nl - Nh);

    denormPredictPriceDiffPerc =((targetPriceDiffPercDl -
    targetPriceDiffPercDh)* normPredictPriceDiffPerc - Nh*
    targetPriceDiffPercDl + targetPriceDiffPercDh*Nl)/(Nl - Nh);

    denormInputPriceDiffPercFromRecord = ((inputPriceDiffPercDl -
    inputPriceDiffPercDh)*normInputPriceDiffPercFromRecord -
    Nh*inputPriceDiffPercDl + inputPriceDiffPercDh*Nl)/(Nl - Nh);

    inputPriceFromFile = arrPrices[k1+12];

    // Convert denormPredictPriceDiffPerc and denormTarget
       PriceDiffPerc to a real
```

```
                //renormalized price
                 realDenormTargetPrice = inputPriceFromFile + inputPrice
                 FromFile*(denormTargetPriceDiffPerc/100);
                 realDenormPredictPrice = inputPriceFromFile + inputPrice
                 FromFile*(denormPredictPriceDiffPerc/100);

                 realDenormTargetToPredictPricePerc = (Math.abs(realDenorm
                 TargetPrice - realDenormPredictPrice)/realDenorm
                 TargetPrice)*100;

                 System.out.println("Month = " + (k1+1) +  "  targetPrice =
                 " + realDenormTargetPrice + " predictPrice = " + realDenorm
                 PredictPrice + "   diff = " + realDenormTargetToPredict
                 PricePerc);

                 if (realDenormTargetToPredictPricePerc > maxGlobal
                 ResultDiff)
                   maxGlobalResultDiff = realDenormTargetToPredict
                   PricePerc;

                 sumGlobalResultDiff = sumGlobalResultDiff + realDenorm
                 TargetToPredictPricePerc;

               } // End of IF

            }  // End for the pair loop

            // Populate chart elements

            tempMonth = (double) k1+14;
            xData.add(tempMonth);
            yData1.add(realDenormTargetPrice);
            yData2.add(realDenormPredictPrice);

         }   // End of loop K1

        // Print the max and average results

        System.out.println(" ");
        System.out.println(" ");
        System.out.println("Results of processing testing batches");
```

```
    averGlobalResultDiff = sumGlobalResultDiff/intNumberOfBatches
    ToProcess;

    System.out.println("maxGlobalResultDiff = " + maxGlobalResultDiff +
    "  i = " + maxGlobalIndex);
    System.out.println("averGlobalResultDiff = " + averGlobalResult
    Diff);
    System.out.println(" ");
    System.out.println(" ");

  }     // End of TRY
 catch (IOException e1)
  {
      e1.printStackTrace();
  }

// All testing batch files have been processed
  XYSeries series1 = Chart.addSeries("Actual Price", xData, yData1);
  XYSeries series2 = Chart.addSeries("Forecasted Price", xData, yData2);

  series1.setLineColor(XChartSeriesColors.BLUE);
  series2.setMarkerColor(Color.ORANGE);
  series1.setLineStyle(SeriesLines.SOLID);
  series2.setLineStyle(SeriesLines.SOLID);

  // Save the chart image
  try
   {
     BitmapEncoder.saveBitmapWithDPI(Chart, strChartFileName,
     BitmapFormat.JPG, 100);
   }
  catch (Exception bt)
   {
     bt.printStackTrace();
   }

  System.out.println ("The Chart has been saved");

} // End of the method
```

Testing Results

Listing 11-7 shows the fragment log of the testing results.

Listing 11-7. Testing Results

```
Month =   80  targetPrice = 1211.91999   predictPrice = 1211.91169
diff = 6.84919E-4
Month =   81  targetPrice = 1181.26999   predictPrice = 1181.26737
diff = 2.22043E-4
Month =   82  targetPrice = 1203.60000   predictPrice = 1203.60487
diff = 4.05172E-4
Month =   83  targetPrice = 1180.59000   predictPrice = 1180.59119
diff = 1.01641E-4
Month =   84  targetPrice = 1156.84999   predictPrice = 1156.84136
diff = 7.46683E-4
Month =   85  targetPrice = 1191.49999   predictPrice = 1191.49043
diff = 8.02666E-4
Month =   86  targetPrice = 1191.32999   predictPrice = 1191.31947
diff = 8.83502E-4
Month =   87  targetPrice = 1234.17999   predictPrice = 1234.17993
diff = 5.48814E-6
Month =   88  targetPrice = 1220.33000   predictPrice = 1220.31947
diff = 8.62680E-4
Month =   89  targetPrice = 1228.80999   predictPrice = 1228.82099
diff = 8.95176E-4
Month =   90  targetPrice = 1207.00999   predictPrice = 1207.00976
diff = 1.92764E-5
Month =   91  targetPrice = 1249.48000   predictPrice = 1249.48435
diff = 3.48523E-4
Month =   92  targetPrice = 1248.28999   predictPrice = 1248.27937
diff = 8.51313E-4
Month =   93  targetPrice = 1280.08000   predictPrice = 1280.08774
diff = 6.05221E-4
Month =   94  targetPrice = 1280.66000   predictPrice = 1280.66295
diff = 2.30633E-4
```

```
Month =   95  targetPrice = 1294.86999   predictPrice = 1294.85904
diff = 8.46250E-4
Month =   96  targetPrice = 1310.60999   predictPrice = 1310.61570
diff = 4.35072E-4
Month =   97  targetPrice = 1270.08999   predictPrice = 1270.08943
diff = 4.41920E-5
Month =   98  targetPrice = 1270.19999   predictPrice = 1270.21071
diff = 8.43473E-4
Month =   99  targetPrice = 1276.65999   predictPrice = 1276.65263
diff = 5.77178E-4
Month = 100  targetPrice = 1303.81999   predictPrice = 1303.82201
diff = 1.54506E-4
Month = 101  targetPrice = 1335.85000   predictPrice = 1335.83897
diff = 8.25569E-4
Month = 102  targetPrice = 1377.93999   predictPrice = 1377.94590
diff = 4.28478E-4
Month = 103  targetPrice = 1400.63000   predictPrice = 1400.62758
diff = 1.72417E-4
Month = 104  targetPrice = 1418.29999   predictPrice = 1418.31083
diff = 7.63732E-4
Month = 105  targetPrice = 1438.23999   predictPrice = 1438.23562
diff = 3.04495E-4
Month = 106  targetPrice = 1406.82000   predictPrice = 1406.83156
diff = 8.21893E-4
Month = 107  targetPrice = 1420.85999   predictPrice = 1420.86256
diff = 1.80566E-4
Month = 108  targetPrice = 1482.36999   predictPrice = 1482.35896
diff = 7.44717E-4
Month = 109  targetPrice = 1530.62000   predictPrice = 1530.62213
diff = 1.39221E-4
Month = 110  targetPrice = 1503.34999   predictPrice = 1503.33884
diff = 7.42204E-4
Month = 111  targetPrice = 1455.27000   predictPrice = 1455.27626
diff = 4.30791E-4
```

```
Month = 112  targetPrice = 1473.98999   predictPrice = 1473.97685
diff = 8.91560E-4
Month = 113  targetPrice = 1526.75000   predictPrice = 1526.76231
diff = 8.06578E-4
Month = 114  targetPrice = 1549.37999   predictPrice = 1549.39017
diff = 6.56917E-4
Month = 115  targetPrice = 1481.14000   predictPrice = 1481.15076
diff = 7.27101E-4
Month = 116  targetPrice = 1468.35999   predictPrice = 1468.35702
diff = 2.02886E-4
Month = 117  targetPrice = 1378.54999   predictPrice = 1378.55999
diff = 7.24775E-4
Month = 118  targetPrice = 1330.63000   predictPrice = 1330.61965
diff = 7.77501E-4
Month = 119  targetPrice = 1322.70000   predictPrice = 1322.69947
diff = 3.99053E-5
Month = 120  targetPrice = 1385.58999   predictPrice = 1385.60045
diff = 7.54811E-4
Month = 121  targetPrice = 1400.38000   predictPrice = 1162.09439
diff = 17.0157E-4

maxErrorPerc = 17.0157819794876
averErrorPerc = 0.14062629735113719
```

Figure 11-6 shows the chart of the testing results.

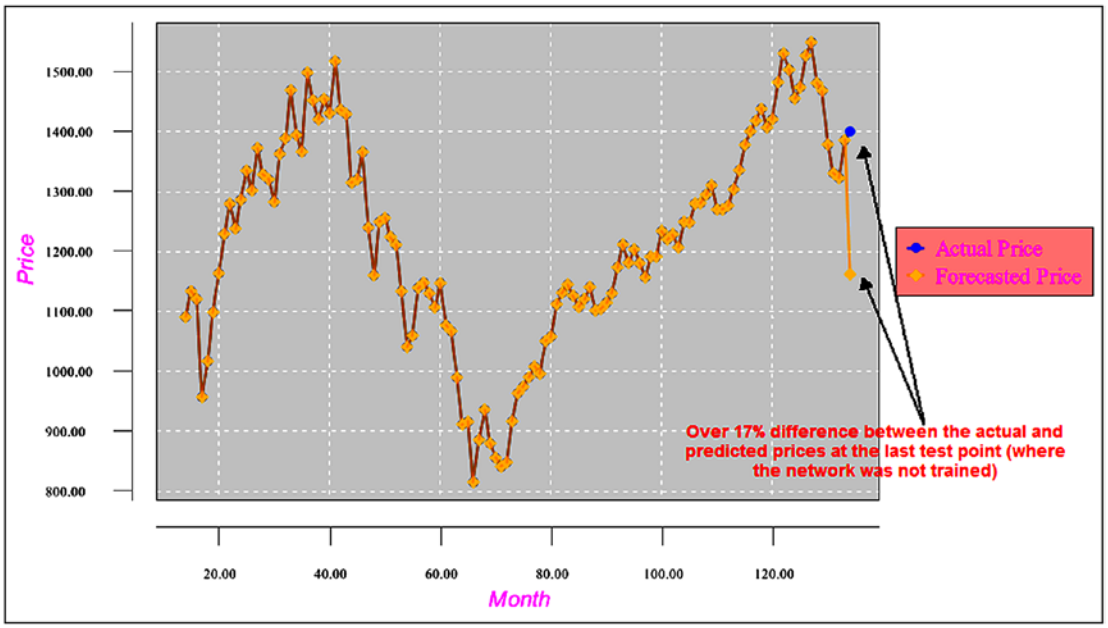


Figure 11-6. *Testing results chart*

Analyzing the Testing Results

At all points where the network was trained, the predicted price closely matches the actual price (the yellow and blue charts practically overlap). However, at the next month point (the point where the network was not trained), the predicted price differs from the actual price (which you happened to know) by more than 17 percent. Even the direction of the next month's predicted price (the difference from the previous month) is wrong. The actual price has slightly increased, while the predicted price has dropped considerably.

With the prices at these points being at around 1200 to 1300, the 17 percent difference represents an error of more than 200 points. This cannot be considered a prediction at all; the result is useless for traders/investors. So, what went wrong? We did not violate the restriction of predicting function values outside of the training range (by transforming the price function to be dependent on the price difference between months instead of sequential months). To answer this question, let's research the issue.

When you processed the last test record, you obtained the values of its first 12 fields from the 12 previous original records. They represent the price difference percent between the current and previous months. And the last field in the record is the percent

difference between the price value at the next month (record 13) and the price value at month 12. With all fields being normalized, the record is shown in Equation 11-2.

$$\begin{array}{l} 0.621937887 \quad 0.550328191 \quad 0.214557935 \quad 0.12012062 \quad 0.419090615 \\ 0.571960009 \quad 0.4321489 \quad 0.039710508 \quad 0.275810074 \quad -0.074423166 \\ 0.101592253 \quad 0.29360278 \quad 0.404494355 \end{array} \tag{11-2}$$

By knowing the price for micro-batch record 12 (which is 1385.95) and obtaining the network prediction as the `targetPriceDiffPerc` field (which is the percent difference between the next month and current month prices), you can calculate the next month's predicted price as shown in Equation 11-3.

$$\begin{aligned} \text{nextMonthPredictedPrice} = \text{record12ActualPrice} + \\ \text{record12ActualPrice}^{*}\text{predictedPriceDiffPerc} / 100.00 \end{aligned} \tag{11-3}$$

To get the network prediction for record 13 (`predictedPriceDiffPerc`), you feed the trained network the vector value of 12 `inputPriceDiffPerc` fields from the currently processed record (see 10-2). The network returns -16.129995719. Putting it all together, you receive the predicted price for the next month.

$$1385.59 - 1385.59^{*}16.12999 / 100.00 = 1{,}162.0943923170353$$

The predicted price for the next month is equal to 1,162.09, while the actual price is 1,400.38, with a difference of 17.02 percent. That's exactly the result shown in the processing log for the last record.

$$\begin{array}{l} \text{Month} = 121 \quad \text{targetPrice} = 1400.3800000016674 \\ \text{predictPrice} = 1162.0943923170353 \quad \text{diff} = 17.0157819794876 \end{array}$$

The calculated result for the next month's price is mathematically correct and is based on the sum of the price at the last training point and the price difference percent between the next and current points returned by the network.

The problem is that the historical stock market prices don't repeat themselves under the same or similar conditions. The price difference percent that the network returns for the calculated vector (10.1) of the last processed record is not correct for calculating the

predicted price for the next month. It is a problem with the model that you used in this example, which assumes that the price difference percent for the future month is similar to the price difference percent recorded for a same or close condition in the past.

This is an important lesson to learn. If the model is wrong, nothing will work. Before doing any neural network development, you need to prove that the chosen model works correctly. This will save you a lot of time and effort.

Some processes are random and unpredictable by definition. If the stock market became predictable, it would simply cease to exist because its premise is based on a difference of opinions. If everyone knew the future market's direction, all investors would be selling, and no one would be buying.

Summary

The chapter explained the importance of selecting the correct working model for the project. You should prove that the model works correctly for your project before starting any development. Failure to select the correct model will leave you with an incorrectly working application. The network will also produce the wrong results when it is used to predict the results of all sort of games (gambling, sports, and so on).

CHAPTER 12

Approximation of Functions in 3D Space

This chapter discusses how to approximate functions in 3D space. Such function values depend on two variables (instead of one variable, which was discussed in the preceding chapters). Everything discussed in this chapter is also correct for functions that depend on more than two variables. Figure 12-1 shows the chart of the 3D function considered in this chapter.

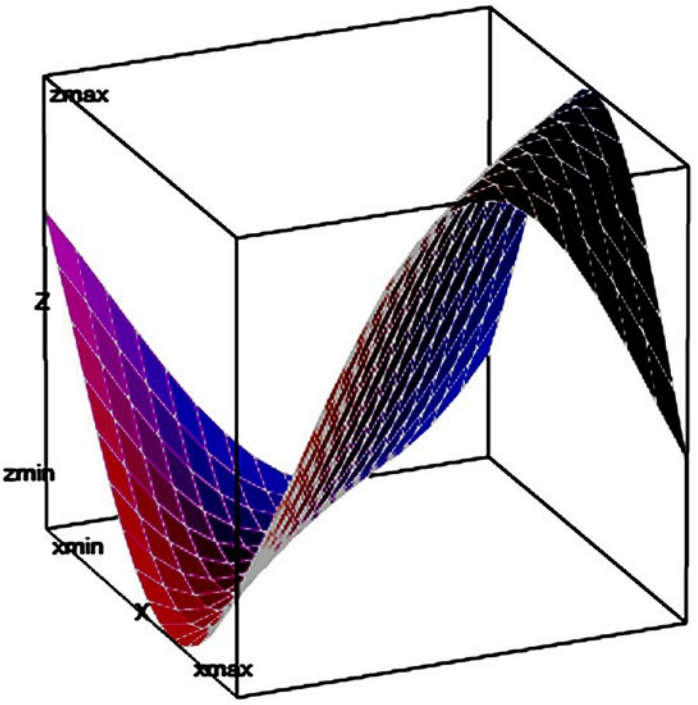


Figure 12-1. *Chart of the function in 3D space*

I. Livshin, *Artificial Neural Networks with Java*, https://doi.org/10.1007/978-1-4842-4421-0_12

Example 8: Approximation of Functions in 3D Space

The function formula is z(x, y) = 50.00 + sin(x*y), but again, let's pretend that the function formula is unknown and that the function is given to you by its values at certain points.

Data Preparation

The function values are given on the interval [3.00, 4.00] with the increment value 0.02 for both function arguments x and y. The starting point for the training data set is 3.00, and the starting point for the testing data set is 3.01. The increment for x and y values is 0.02. The training data set records consist of three fields.

The record structure of the training data set is as follows:

Field 1: The value of the x argument

Field 2: The value of the y argument

Field 3: The function value

Table 12-1 shows a fragment of the training data set. The training data set includes records for all possible combinations of x and y values.

Table 12-1. *Fragment of the Training Data Set*

x	y	z
3	3	50.41211849
3	3.02	50.35674187
3	3.04	50.30008138
3	3.06	50.24234091
3	3.08	50.18372828
3	3.1	50.12445442
3	3.12	50.06473267
3	3.14	50.00477794
3	3.16	49.94480602

(*continued*)

Table 12-1. (*continued*)

x	y	z
3	3.18	49.88503274
3	3.2	49.82567322
3	3.22	49.76694108
3	3.24	49.70904771
3	3.26	49.65220145
3	3.28	49.59660689
3	3.3	49.54246411
3	3.32	49.48996796
3	3.34	49.43930738
3	3.36	49.39066468
3	3.38	49.34421494
3	3.4	49.30012531
3	3.42	49.25855448
3	3.44	49.21965205
3	3.46	49.18355803
3	3.48	49.15040232
3	3.5	49.12030424
3	3.52	49.09337212
3	3.54	49.06970288
3	3.56	49.0493817
3	3.58	49.03248173
3	3.6	49.01906377
3	3.62	49.00917613
3	3.64	49.00285438
3	3.66	49.00012128

(*continued*)

Table 12-1. (*continued*)

x	y	z
3	3.68	49.00098666
3	3.7	49.00544741
3	3.72	49.01348748
3	3.74	49.02507793
3	3.76	49.04017704
3	3.78	49.05873048
3	3.8	49.08067147
3	3.82	49.10592106
3	3.84	49.13438836
3	3.86	49.16597093
3	3.88	49.2005551
3	3.9	49.23801642
3	3.92	49.27822005
3	3.94	49.3210213
3	3.96	49.36626615
3	3.98	49.41379176
3	4	49.46342708
3.02	3	50.35674187
3.02	3.02	50.29969979
3.02	3.04	50.24156468
3.02	3.06	50.18254857
3.02	3.08	50.1228667
3.02	3.1	50.06273673
3.02	3.12	50.00237796

(*continued*)

Table 12-1. (*continued*)

x	y	z
3.02	3.14	49.94201051
3.02	3.16	49.88185455
3.02	3.18	49.82212948
3.02	3.2	49.76305311

Table 12-2 shows the fragment of the testing data set. It has the same structure, but it includes the x and y points not used for the network training. Table 12-2 shows a fragment of the testing data set.

Table 12-2. *Fragment of the Testing Data Set*

x	y	z
3.01	3.01	50.35664845
3.01	3.03	50.29979519
3.01	3.05	50.24185578
3.01	3.07	50.18304015
3.01	3.09	50.12356137
3.01	3.11	50.06363494
3.01	3.13	50.00347795
3.01	3.15	49.94330837
3.01	3.17	49.88334418
3.01	3.19	49.82380263
3.01	3.21	49.76489943
3.01	3.23	49.70684798
3.01	3.25	49.64985862
3.01	3.27	49.59413779

(*continued*)

Table 12-2. (*continued*)

x	y	z
3.01	3.29	49.53988738
3.01	3.31	49.48730393
3.01	3.33	49.43657796
3.01	3.35	49.38789323
3.01	3.37	49.34142613
3.01	3.39	49.29734501
3.01	3.41	49.25580956
3.01	3.43	49.21697029
3.01	3.45	49.18096788
3.01	3.47	49.14793278
3.01	3.49	49.11798468
3.01	3.51	49.09123207
3.01	3.53	49.06777188
3.01	3.55	49.04768909
3.01	3.57	49.03105648
3.01	3.59	49.0179343
3.01	3.61	49.00837009
3.01	3.63	49.0023985
3.01	3.65	49.00004117
3.01	3.67	49.00130663
3.01	3.69	49.00619031
3.01	3.71	49.0146745
3.01	3.73	49.02672848
3.01	3.75	49.04230856

(*continued*)

Table 12-2. (*continued*)

x	y	z
3.01	3.77	49.06135831
3.01	3.79	49.08380871
3.01	3.81	49.10957841
3.01	3.83	49.13857407
3.01	3.85	49.17069063
3.01	3.87	49.20581173
3.01	3.89	49.24381013
3.01	3.91	49.28454816
3.01	3.93	49.32787824
3.01	3.95	49.37364338
3.01	3.97	49.42167777
3.01	3.99	49.47180739
3.03	3.01	50.29979519
3.03	3.03	50.24146764
3.03	3.05	50.18225361

Network Architecture

Figure 12-2 shows the network architecture. The function you'll process has two inputs (x and y); therefore, the network architecture has two inputs.

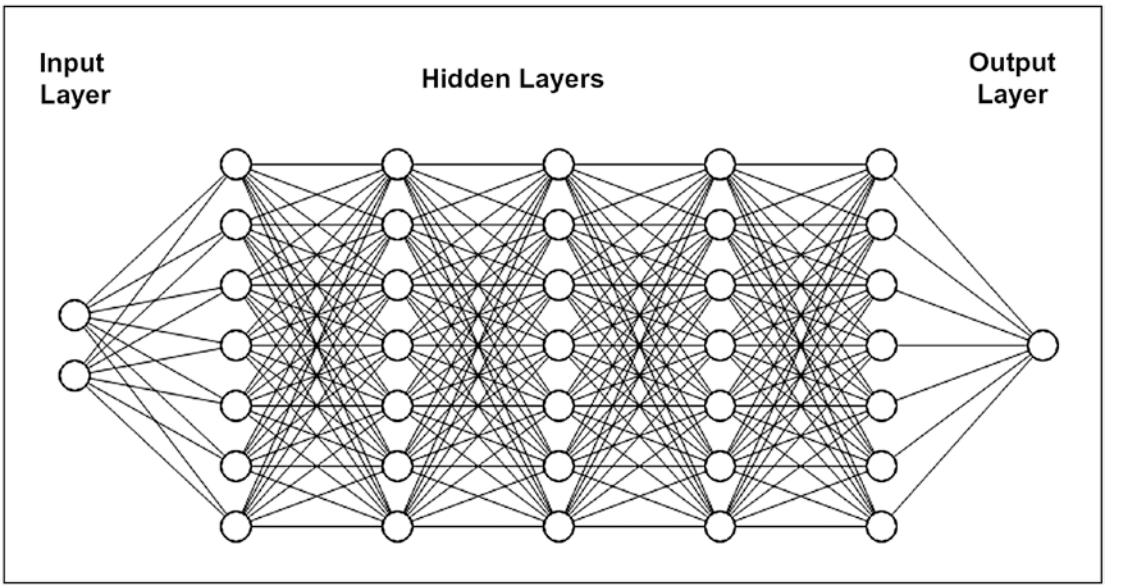


Figure 12-2. *Network architecture*

Both the training and testing data sets are normalized before being processed. You will approximate the function using the conventional network process. Based on the processing results, you will then decide whether you need to use the micro-batch method.

Program Code

Listing 12-1 shows the program code.

Listing 12-1. Program Code

```
// ================================================================
// Approximation of the 3-D Function using conventional process.
// The input file is normalized.
// ================================================================

package sample9;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.PrintWriter;
```

```
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.*;
import java.util.Properties;
import java.time.YearMonth;
import java.awt.Color;
import java.awt.Font;
import java.io.BufferedReader;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.LocalDate;
import java.time.Month;
import java.time.ZoneId;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.List;
import java.util.Locale;
import java.util.Properties;

import org.encog.Encog;
import org.encog.engine.network.activation.ActivationTANH;
import org.encog.engine.network.activation.ActivationReLU;
import org.encog.ml.data.MLData;
import org.encog.ml.data.MLDataPair;
import org.encog.ml.data.MLDataSet;
import org.encog.ml.data.buffer.MemoryDataLoader;
import org.encog.ml.data.buffer.codec.CSVDataCODEC;
import org.encog.ml.data.buffer.codec.DataSetCODEC;
import org.encog.neural.networks.BasicNetwork;
import org.encog.neural.networks.layers.BasicLayer;
```

```
import org.encog.neural.networks.training.propagation.resilient.
ResilientPropagation;
import org.encog.persist.EncogDirectoryPersistence;
import org.encog.util.csv.CSVFormat;

import org.knowm.xchart.SwingWrapper;
import org.knowm.xchart.XYChart;
import org.knowm.xchart.XYChartBuilder;
import org.knowm.xchart.XYSeries;
import org.knowm.xchart.demo.charts.ExampleChart;
import org.knowm.xchart.style.Styler.LegendPosition;
import org.knowm.xchart.style.colors.ChartColor;
import org.knowm.xchart.style.colors.XChartSeriesColors;
import org.knowm.xchart.style.lines.SeriesLines;
import org.knowm.xchart.style.markers.SeriesMarkers;
import org.knowm.xchart.BitmapEncoder;
import org.knowm.xchart.BitmapEncoder.BitmapFormat;
import org.knowm.xchart.QuickChart;
import org.knowm.xchart.SwingWrapper;

public class Sample9 implements ExampleChart<XYChart>
{
    // Interval to normalize
   static double Nh =  1;
   static double Nl = -1;

   // First column
   static double minXPointDl = 2.00;
   static double maxXPointDh = 6.00;

   // Second column
   static double minYPointDl = 2.00;
   static double maxYPointDh = 6.00;

   // Third  column - target data
   static double minTargetValueDl = 45.00;
   static double maxTargetValueDh = 55.00;
```

```
 static double doublePointNumber = 0.00;
 static int intPointNumber = 0;
 static InputStream input = null;
 static double[] arrPrices = new double[2700];
 static double normInputXPointValue = 0.00;
 static double normInputYPointValue = 0.00;
 static double normPredictValue = 0.00;
 static double normTargetValue = 0.00;
 static double normDifferencePerc = 0.00;
 static double returnCode = 0.00;
 static double denormInputXPointValue = 0.00;
 static double denormInputYPointValue = 0.00;
 static double denormPredictValue = 0.00;
 static double denormTargetValue = 0.00;
 static double valueDifference = 0.00;
 static int numberOfInputNeurons;
 static int numberOfOutputNeurons;
  static int intNumberOfRecordsInTestFile;
 static String trainFileName;
 static String priceFileName;
 static String testFileName;
 static String chartTrainFileName;
 static String chartTrainFileNameY;
 static String chartTestFileName;
 static String networkFileName;
 static int workingMode;
 static String cvsSplitBy = ",";

 static int numberOfInputRecords = 0;

 static List<Double> xData = new ArrayList<Double>();
 static List<Double> yData1 = new ArrayList<Double>();
 static List<Double> yData2 = new ArrayList<Double>();

 static XYChart Chart;

@Override
```

```
public XYChart getChart()
 {
  // Create Chart

  Chart = new  XYChartBuilder().width(900).height(500).title(getClass().
   getSimpleName()).xAxisTitle("x").yAxisTitle("y= f(x)").build();

  // Customize Chart
  //Chart = new  XYChartBuilder().width(900).height(500).title(getClass().
  //   getSimpleName()).xAxisTitle("y").yAxisTitle("z= f(y)").build();

  //Chart = new  XYChartBuilder().width(900).height(500).title(getClass().
  //          getSimpleName()).xAxisTitle("y").yAxisTitle("z= f(y)").build();

  // Customize Chart
  Chart.getStyler().setPlotBackgroundColor(ChartColor.
  getAWTColor(ChartColor.GREY));
  Chart.getStyler().setPlotGridLinesColor(new Color(255, 255, 255));

  //Chart.getStyler().setPlotBackgroundColor(ChartColor.
  getAWTColor(ChartColor.WHITE));
  //Chart.getStyler().setPlotGridLinesColor(new Color(0, 0, 0));
  Chart.getStyler().setChartBackgroundColor(Color.WHITE);
  //Chart.getStyler().setLegendBackgroundColor(Color.PINK);
  Chart.getStyler().setLegendBackgroundColor(Color.WHITE);
  //Chart.getStyler().setChartFontColor(Color.MAGENTA);
  Chart.getStyler().setChartFontColor(Color.BLACK);
  Chart.getStyler().setChartTitleBoxBackgroundColor(new Color(0, 222, 0));
  Chart.getStyler().setChartTitleBoxVisible(true);
  Chart.getStyler().setChartTitleBoxBorderColor(Color.BLACK);
  Chart.getStyler().setPlotGridLinesVisible(true);
  Chart.getStyler().setAxisTickPadding(20);
  Chart.getStyler().setAxisTickMarkLength(15);
  Chart.getStyler().setPlotMargin(20);
  Chart.getStyler().setChartTitleVisible(false);
  Chart.getStyler().setChartTitleFont(new Font(Font.MONOSPACED, Font.
  BOLD, 24));
```

```
Chart.getStyler().setLegendFont(new Font(Font.SERIF, Font.PLAIN, 18));
Chart.getStyler().setLegendPosition(LegendPosition.OutsideS);
Chart.getStyler().setLegendSeriesLineLength(12);
Chart.getStyler().setAxisTitleFont(new Font(Font.SANS_SERIF, Font.
ITALIC, 18));
Chart.getStyler().setAxisTickLabelsFont(new Font(Font.SERIF, Font.
PLAIN, 11));
Chart.getStyler().setDatePattern("yyyy-MM");
Chart.getStyler().setDecimalPattern("#0.00");

try
   {

     // Common part of config data
     networkFileName =
      "C:/My_Neural_Network_Book/Book_Examples/Sample9_Saved_Network_
      File.csv";
     numberOfInputNeurons = 2;
     numberOfOutputNeurons = 1;

     if(workingMode == 1)
      {
        // Training mode
        numberOfInputRecords = 2602;
        trainFileName = "C:/My_Neural_Network_Book/Book_Examples/
                         Sample9_Calculate_Train_Norm.csv";
        chartTrainFileName = "C:/My_Neural_Network_Book/Book_Examples/
                              Sample9_Chart_X_Training_Results.csv";
         chartTrainFileName = "C:/My_Neural_Network_Book/Book_Examples/
                              Sample9_Chart_Y_Training_Results.csv";

        File file1 = new File(chartTrainFileName);
        File file2 = new File(networkFileName);

        if(file1.exists())
          file1.delete();
```

```
            if(file2.exists())
              file2.delete();

            returnCode = 0;    // Clear the error Code

            do
             {
               returnCode = trainValidateSaveNetwork();
             }  while (returnCode > 0);
          }
         else
          {
             // Testing mode
             numberOfInputRecords = 2602;
             testFileName = "C:/My_Neural_Network_Book/Book_Examples/
             Sample9_Calculate_Test_Norm.csv";
             chartTestFileName = "C:/My_Neural_Network_Book/Book_Examples/
             Sample9_Chart_X_Testing_Results.csv";
             chartTestFileName = "C:/My_Neural_Network_Book/Book_Examples/
             Sample9_Chart_Y_Testing_Results.csv";

             loadAndTestNetwork();
          }
       }
      catch (Throwable t)
        {
          t.printStackTrace();
          System.exit(1);
        }
       finally
        {
          Encog.getInstance().shutdown();
        }
```

```
 Encog.getInstance().shutdown();

 return Chart;

}  // End of the method

// ========================================================
// Load CSV to memory.
// @return The loaded dataset.
// ========================================================
public static MLDataSet loadCSV2Memory(String filename, int input,
int ideal, boolean headers,
        CSVFormat format, boolean significance)
  {
     DataSetCODEC codec = new CSVDataCODEC(new File(filename), format,
     headers, input, ideal, significance);
     MemoryDataLoader load = new MemoryDataLoader(codec);
     MLDataSet dataset = load.external2Memory();
     return dataset;
  }

// ========================================================
//  The main method.
//  @param Command line arguments. No arguments are used.
// ========================================================
public static void main(String[] args)
 {
   ExampleChart<XYChart> exampleChart = new Sample9();
   XYChart Chart = exampleChart.getChart();
   new SwingWrapper<XYChart>(Chart).displayChart();
 } // End of the main method
```

```
//=====================================================================
// This method trains, Validates, and saves the trained network file
//=====================================================================
static public double trainValidateSaveNetwork()
 {
   // Load the training CSV file in memory
   MLDataSet trainingSet = loadCSV2Memory(trainFileName,
                           numberOfInputNeurons, numberOfOutputNeurons,
                           true,CSVFormat.ENGLISH,false);

   // create a neural network
   BasicNetwork network = new BasicNetwork();

   // Input layer
   network.addLayer(new BasicLayer(null,true,numberOfInputNeurons));

   // Hidden layer
   network.addLayer(new BasicLayer(new ActivationTANH(),true,7));
   network.addLayer(new BasicLayer(new ActivationTANH(),true,7));
   network.addLayer(new BasicLayer(new ActivationTANH(),true,7));
   network.addLayer(new BasicLayer(new ActivationTANH(),true,7));
   network.addLayer(new BasicLayer(new ActivationTANH(),true,7));

   // Output layer
   network.addLayer(new BasicLayer(new ActivationTANH(),false,1));

   network.getStructure().finalizeStructure();
   network.reset();

   // train the neural network
   final ResilientPropagation train = new ResilientPropagation(network,
   trainingSet);

   int epoch = 1;

   do
    {
     train.iteration();
     System.out.println("Epoch #" + epoch + " Error:" + train.getError());
```

```
    epoch++;

    if (epoch >= 11000 && network.calculateError(trainingSet) >
    0.00000091)    // 0.00000371
        {
         returnCode = 1;

         System.out.println("Try again");
         return returnCode;
        }
 } while(train.getError() > 0.0000009);  // 0.0000037

// Save the network file
EncogDirectoryPersistence.saveObject(new File(networkFileName),network);

System.out.println("Neural Network Results:");

double sumNormDifferencePerc = 0.00;
double averNormDifferencePerc = 0.00;
double maxNormDifferencePerc = 0.00;

int m = 0;                   // Record number in the input file
                                double xPointer = 0.00;

for(MLDataPair pair: trainingSet)
  {
      m++;
      xPointer++;

      //if(m == 0)
      // continue;

       final MLData output = network.compute(pair.getInput());

       MLData inputData = pair.getInput();
       MLData actualData = pair.getIdeal();
       MLData predictData = network.compute(inputData);

       // Calculate and print the results
```

```
normInputXPointValue = inputData.getData(0);
normInputYPointValue = inputData.getData(1);
normTargetValue = actualData.getData(0);
normPredictValue = predictData.getData(0);

denormInputXPointValue = ((minXPointDl - maxXPointDh)*normInpu
tXPointValue -
  Nh*minXPointDl + maxXPointDh *Nl)/(Nl - Nh);

denormInputYPointValue = ((minYPointDl - maxYPointDh)*normInpu
tYPointValue -
  Nh*minYPointDl + maxYPointDh *Nl)/(Nl - Nh);

denormTargetValue =((minTargetValueDl - maxTargetValueDh)*
normTargetValue -
  Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

denormPredictValue =((minTargetValueDl - maxTargetValueDh)*
normPredictValue -
  Nh*minTargetValueDl + maxTargetValueDh*Nl)/(Nl - Nh);

valueDifference =
  Math.abs(((denormTargetValue - denormPredictValue)/
  denormTargetValue)*100.00);

System.out.println ("xPoint = " + denormInputXPointValue +
"  yPoint = " +
  denormInputYPointValue + "  denormTargetValue = " +
     denormTargetValue + "  denormPredictValue = " +
     denormPredictValue + "  valueDifference = " +
     valueDifference);

//System.out.println("intPointNumber = " + intPointNumber);

sumNormDifferencePerc = sumNormDifferencePerc + valueDifference;

if (valueDifference > maxNormDifferencePerc)
  maxNormDifferencePerc = valueDifference;

xData.add(denormInputYPointValue);
```

```
      //xData.add(denormInputYPointValue);
      yData1.add(denormTargetValue);
      yData2.add(denormPredictValue);

 }   // End for pair loop

 XYSeries series1 = Chart.addSeries("Actual data", xData, yData1);
 XYSeries series2 = Chart.addSeries("Predict data", xData, yData2);

 series1.setLineColor(XChartSeriesColors.BLACK);
 series2.setLineColor(XChartSeriesColors.LIGHT_GREY);

 series1.setMarkerColor(Color.BLACK);
 series2.setMarkerColor(Color.WHITE);
 series1.setLineStyle(SeriesLines.SOLID);
 series2.setLineStyle(SeriesLines.SOLID);

 try
  {
    //Save the chart image
    //BitmapEncoder.saveBitmapWithDPI(Chart, chartTrainFileName,
    //  BitmapFormat.JPG, 100);

    BitmapEncoder.saveBitmapWithDPI(Chart,chartTrainFileName,BitmapF
    ormat.JPG, 100);

    System.out.println ("Train Chart file has been saved") ;
  }
catch (IOException ex)
 {
  ex.printStackTrace();
  System.exit(3);
 }

 // Finally, save this trained network
 EncogDirectoryPersistence.saveObject(new File(networkFileName),net
 work);
 System.out.println ("Train Network has been saved") ;

 averNormDifferencePerc  = sumNormDifferencePerc/numberOfInputRecords;
```

```
        System.out.println(" ");
        System.out.println("maxErrorPerc = " + maxNormDifferencePerc +
        "  averErrorPerc = " + averNormDifferencePerc);

        returnCode = 0.00;
        return returnCode;

    }   // End of the method

   //================================================
   // This method load and test the trainrd network
   //================================================
   static public void loadAndTestNetwork()
    {
     System.out.println("Testing the networks results");

     List<Double> xData = new ArrayList<Double>();
     List<Double> yData1 = new ArrayList<Double>();
     List<Double> yData2 = new ArrayList<Double>();

     double targetToPredictPercent = 0;
     double maxGlobalResultDiff = 0.00;
     double averGlobalResultDiff = 0.00;
     double sumGlobalResultDiff = 0.00;
     double maxGlobalIndex = 0;
     double normInputXPointValueFromRecord = 0.00;
     double normInputYPointValueFromRecord = 0.00;
     double normTargetValueFromRecord = 0.00;
     double normPredictValueFromRecord = 0.00;

     BasicNetwork network;


     maxGlobalResultDiff = 0.00;
     averGlobalResultDiff = 0.00;
     sumGlobalResultDiff = 0.00;

     // Load the test dataset into memory
     MLDataSet testingSet =
```

```
loadCSV2Memory(testFileName,numberOfInputNeurons,numberOfOutputNeurons
,true,
  CSVFormat.ENGLISH,false);

// Load the saved trained network
network =
  (BasicNetwork)EncogDirectoryPersistence.loadObject(new
  File(networkFileName));

int i = - 1; // Index of the current record
double xPoint = -0.00;

for (MLDataPair pair:  testingSet)
 {
      i++;
      xPoint = xPoint + 2.00;

      MLData inputData = pair.getInput();
      MLData actualData = pair.getIdeal();
      MLData predictData = network.compute(inputData);

      // These values are Normalized as the whole input is
      normInputXPointValueFromRecord = inputData.getData(0);
      normInputYPointValueFromRecord = inputData.getData(1);
      normTargetValueFromRecord = actualData.getData(0);
      normPredictValueFromRecord = predictData.getData(0);

      denormInputXPointValue = ((minXPointDl - maxXPointDh)*
        normInputXPointValueFromRecord - Nh*minXPointDl +
        maxXPointDh*Nl)/(Nl - Nh);

      denormInputYPointValue = ((minYPointDl - maxYPointDh)*
        normInputYPointValueFromRecord - Nh*minYPointDl +
        maxYPointDh*Nl)/(Nl - Nh);

      denormTargetValue = ((minTargetValueDl - maxTargetValueDh)*
        normTargetValueFromRecord - Nh*minTargetValueDl +
        maxTargetValueDh*Nl)/(Nl - Nh);

      denormPredictValue =((minTargetValueDl - maxTargetValueDh)*
```

```
            normPredictValueFromRecord - Nh*minTargetValueDl +
            maxTargetValueDh*Nl)/(Nl - Nh);

          targetToPredictPercent = Math.abs((denormTargetValue -
          denormPredictValue)/
            denormTargetValue*100);

          System.out.println("xPoint = " + denormInputXPointValue + "
          yPoint = " +
              denormInputYPointValue + "  TargetValue = " +
               denormTargetValue + "  PredictValue = " +
               denormPredictValue + "  DiffPerc = " +
                  targetToPredictPercent);

          if (targetToPredictPercent > maxGlobalResultDiff)
             maxGlobalResultDiff = targetToPredictPercent;

          sumGlobalResultDiff = sumGlobalResultDiff + targetToPredictPercent;

          // Populate chart elements
          xData.add(denormInputXPointValue);
          yData1.add(denormTargetValue);
          yData2.add(denormPredictValue);

       }  // End for pair loop

      // Print the max and average results
      System.out.println(" ");
      averGlobalResultDiff = sumGlobalResultDiff/numberOfInputRecords;

      System.out.println("maxErrorPerc = " + maxGlobalResultDiff);
      System.out.println("averErrorPerc = " + averGlobalResultDiff);

      // All testing batch files have been processed
      XYSeries series1 = Chart.addSeries("Actual data", xData, yData1);
      XYSeries series2 = Chart.addSeries("Predict data", xData, yData2);

      series1.setLineColor(XChartSeriesColors.BLACK);
      series2.setLineColor(XChartSeriesColors.LIGHT_GREY);
```

```
        series1.setMarkerColor(Color.BLACK);
        series2.setMarkerColor(Color.WHITE);
        series1.setLineStyle(SeriesLines.SOLID);
        series2.setLineStyle(SeriesLines.SOLID);

        // Save the chart image
        try
         {
           BitmapEncoder.saveBitmapWithDPI(Chart, chartTestFileName ,
           BitmapFormat.JPG, 100);
         }
        catch (Exception bt)
         {
            bt.printStackTrace();
         }

        System.out.println ("The Chart has been saved");
        System.out.println("End of testing for test records");

    } // End of the method

} // End of the class
```

Processing Results

Listing 12-2 shows the end fragment of the training processing results.

Listing 12-2. End Fragment of the Training Processing Results

```
xPoint = 4.0  yPoint = 3.3    TargetValue = 50.59207
PredictedValue = 50.58836  DiffPerc = 0.00733
xPoint = 4.0  yPoint = 3.32  TargetValue = 50.65458
PredictedValue = 50.65049  DiffPerc = 0.00806
xPoint = 4.0  yPoint = 3.34  TargetValue = 50.71290
PredictedValue = 50.70897  DiffPerc = 0.00775
xPoint = 4.0  yPoint = 3.36  TargetValue = 50.76666
PredictedValue = 50.76331  DiffPerc = 0.00659
```

```
xPoint = 4.0  yPoint = 3.38  TargetValue = 50.81552
PredictedValue = 50.81303  DiffPerc = 0.00488
xPoint = 4.0  yPoint = 3.4    TargetValue = 50.85916
PredictedValue = 50.85764  DiffPerc = 0.00298
xPoint = 4.0  yPoint = 3.42  TargetValue = 50.89730
PredictedValue = 50.89665  DiffPerc = 0.00128
xPoint = 4.0  yPoint = 3.44  TargetValue = 50.92971
PredictedValue = 50.92964  DiffPerc = 1.31461
xPoint = 4.0  yPoint = 3.46  TargetValue = 50.95616
PredictedValue = 50.95626  DiffPerc = 1.79849
xPoint = 4.0  yPoint = 3.48  TargetValue = 50.97651
PredictedValue = 50.97624  DiffPerc = 5.15406
xPoint = 4.0  yPoint = 3.5    TargetValue = 50.99060
PredictedValue = 50.98946  DiffPerc = 0.00224
xPoint = 4.0  yPoint = 3.52  TargetValue = 50.99836
PredictedValue = 50.99587  DiffPerc = 0.00488
xPoint = 4.0  yPoint = 3.54  TargetValue = 50.99973
PredictedValue = 50.99556  DiffPerc = 0.00818
xPoint = 4.0  yPoint = 3.56  TargetValue = 50.99471
PredictedValue = 50.98869  DiffPerc = 0.01181
xPoint = 4.0  yPoint = 3.58  TargetValue = 50.98333
PredictedValue = 50.97548  DiffPerc = 0.01538
xPoint = 4.0  yPoint = 3.6    TargetValue = 50.96565
PredictedValue = 50.95619  DiffPerc = 0.01856
xPoint = 4.0  yPoint = 3.62  TargetValue = 50.94180
PredictedValue = 50.93108  DiffPerc = 0.02104
xPoint = 4.0  yPoint = 3.64  TargetValue = 50.91193
PredictedValue = 50.90038  DiffPerc = 0.02268
xPoint = 4.0  yPoint = 3.66  TargetValue = 50.87622
PredictedValue = 50.86429  DiffPerc = 0.02344
xPoint = 4.0  yPoint = 3.68  TargetValue = 50.83490
PredictedValue = 50.82299  DiffPerc = 0.02342
xPoint = 4.0  yPoint = 3.7    TargetValue = 50.78825
PredictedValue = 50.77664  DiffPerc = 0.02286
```

```
xPoint = 4.0  yPoint = 3.72  TargetValue = 50.73655
PredictedValue = 50.72537  DiffPerc = 0.02203
xPoint = 4.0  yPoint = 3.74  TargetValue = 50.68014
PredictedValue = 50.66938  DiffPerc = 0.02124
xPoint = 4.0  yPoint = 3.76  TargetValue = 50.61938
PredictedValue = 50.60888  DiffPerc = 0.02074
xPoint = 4.0  yPoint = 3.78  TargetValue = 50.55466
PredictedValue = 50.54420  DiffPerc = 0.02069
xPoint = 4.0  yPoint = 3.8    TargetValue = 50.48639
PredictedValue = 50.47576  DiffPerc = 0.02106
xPoint = 4.0  yPoint = 3.82  TargetValue = 50.41501
PredictedValue = 50.40407  DiffPerc = 0.02170
xPoint = 4.0  yPoint = 3.84  TargetValue = 50.34098
PredictedValue = 50.32979  DiffPerc = 0.02222
xPoint = 4.0  yPoint = 3.86  TargetValue = 50.26476
PredictedValue = 50.25363  DiffPerc = 0.02215
xPoint = 4.0  yPoint = 3.88  TargetValue = 50.18685
PredictedValue = 50.17637  DiffPerc = 0.02088
xPoint = 4.0  yPoint = 3.9   TargetValue = 50.10775
PredictedValue = 50.09883  DiffPerc = 0.01780
xPoint = 4.0  yPoint = 3.92  TargetValue = 50.02795
PredictedValue = 50.02177  DiffPerc = 0.01236
xPoint = 4.0  yPoint = 3.94  TargetValue = 49.94798
PredictedValue = 49.94594  DiffPerc = 0.00409
xPoint = 4.0  yPoint = 3.96  TargetValue = 49.86834
PredictedValue = 49.87197  DiffPerc = 0.00727
xPoint = 4.0  yPoint = 3.98  TargetValue = 49.78954
PredictedValue = 49.80041  DiffPerc = 0.02182
xPoint = 4.0  yPoint = 4.0   TargetValue = 49.71209
PredictedValue = 49.73170  DiffPerc = 0.03944

maxErrorPerc = 0.03944085774812906
averErrorPerc = 0.00738084715672128
```

I won't be displaying the chart of the training results here because drawing two crossing 3D charts gets messy. Instead, I will project all the target and predicted values

from the chart on a single panel so they can be easily compared. Figure 12-3 shows the chart with the projection of function values on a single panel.

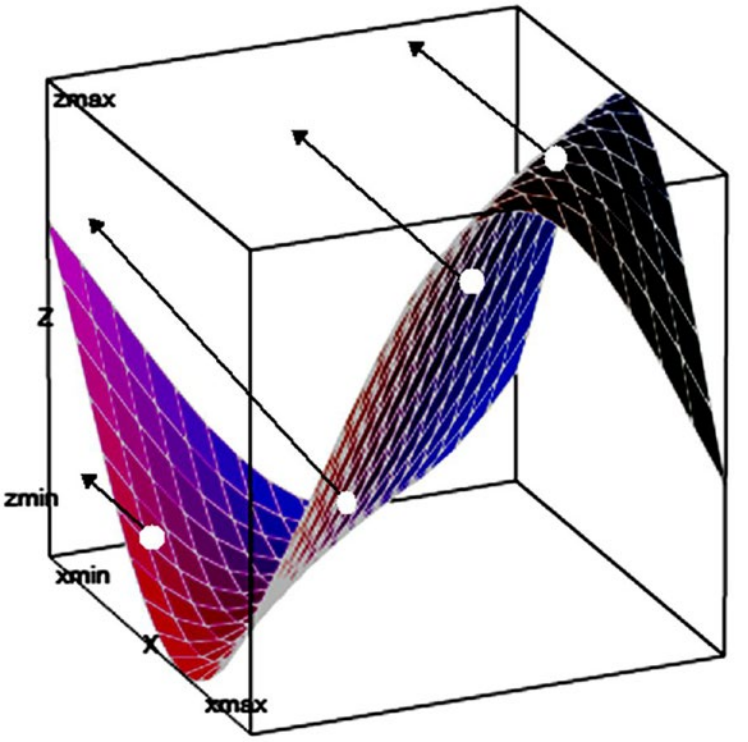


Figure 12-3. *Projection of the function values on a single panel*

Figure 12-4 shows the projection chart of the training results.

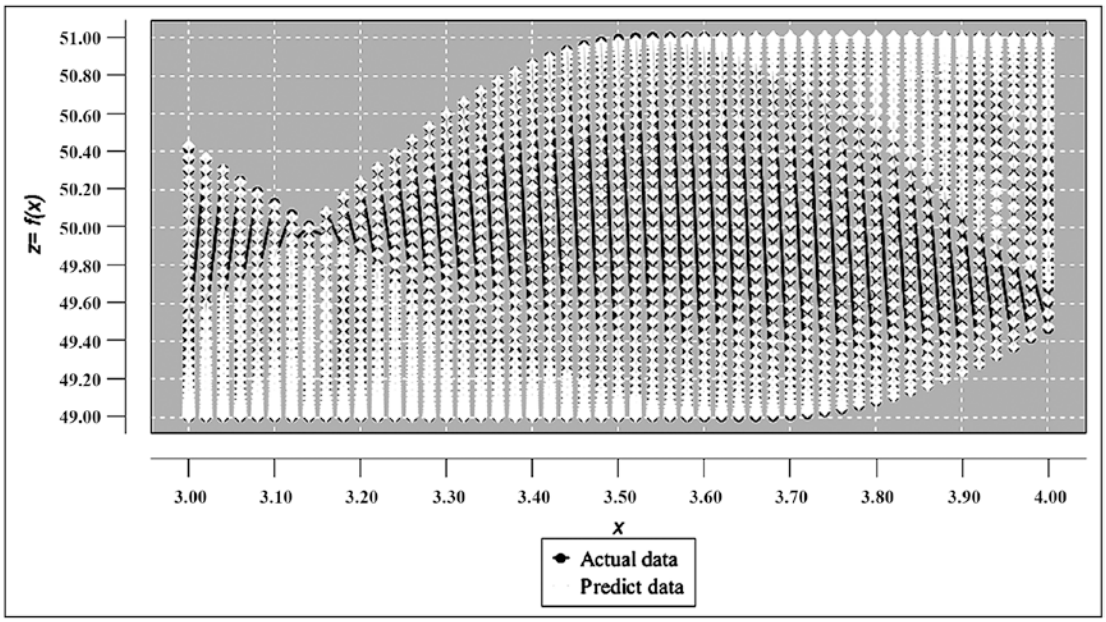


Figure 12-4. *Projection chart of the training results*

Listing 12-3 shows the testing results.

Listing 12-3. Testing Results

```
xPoint = 3.99900  yPoint = 3.13900  TargetValue = 49.98649
PredictValue = 49.98797  DiffPerc = 0.00296
xPoint = 3.99900  yPoint = 3.15900  TargetValue = 50.06642
PredictValue = 50.06756  DiffPerc = 0.00227
xPoint = 3.99900  yPoint = 3.17900  TargetValue = 50.14592
PredictValue = 50.14716  DiffPerc = 0.00246
xPoint = 3.99900  yPoint = 3.19900  TargetValue = 50.22450
PredictValue = 50.22617  DiffPerc = 0.00333
xPoint = 3.99900  yPoint = 3.21900  TargetValue = 50.30163
PredictValue = 50.30396  DiffPerc = 0.00462
xPoint = 3.99900  yPoint = 3.23900  TargetValue = 50.37684
PredictValue = 50.37989  DiffPerc = 0.00605
```

```
xPoint = 3.99900  yPoint = 3.25900  TargetValue = 50.44964
PredictValue = 50.45333  DiffPerc = 0.00730
xPoint = 3.99900  yPoint = 3.27900  TargetValue = 50.51957
PredictValue = 50.52367  DiffPerc = 0.00812
xPoint = 3.99900  yPoint = 3.29900  TargetValue = 50.58617
PredictValue 50.59037  DiffPerc = 0.00829
xPoint = 3.99900  yPoint = 3.31900  TargetValue = 50.64903
PredictValue = 50.65291  DiffPerc = 0.00767
xPoint = 3.99900  yPoint = 3.33900  TargetValue = 50.70773
PredictValue = 50.71089  DiffPerc = 0.00621
xPoint = 3.99900  yPoint = 3.35900  TargetValue = 50.76191
PredictValue = 50.76392  DiffPerc = 0.00396
xPoint = 3.99900  yPoint = 3.37900  TargetValue = 50.81122
PredictValue = 50.81175  DiffPerc = 0.00103
xPoint = 3.99900  yPoint = 3.39900  TargetValue = 50.85535
PredictValue = 50.85415  DiffPerc = 0.00235
xPoint = 3.99900  yPoint = 3.41900  TargetValue = 50.89400
PredictValue = 50.89098  DiffPerc = 0.00594
xPoint = 3.99900  yPoint = 3.43900  TargetValue = 50.92694
PredictValue = 50.92213  DiffPerc = 0.00945
xPoint = 3.99900  yPoint = 3.45900  TargetValue = 50.95395
PredictValue = 50.94754  DiffPerc = 0.01258
xPoint = 3.99900  yPoint = 3.47900  TargetValue = 50.97487
PredictValue = 50.96719  DiffPerc = 0.01507
xPoint = 3.99900  yPoint = 3.49900  TargetValue = 50.98955
PredictValue = 50.98104  DiffPerc = 0.01669
xPoint = 3.99900  yPoint = 3.51900  TargetValue = 50.99790
PredictValue = 50.98907  DiffPerc = 0.01731
xPoint = 3.99900  yPoint = 3.53900  TargetValue = 50.99988
PredictValue = 50.99128  DiffPerc = 0.01686
xPoint = 3.99900  yPoint = 3.55900  TargetValue = 50.99546
PredictValue = 50.98762  DiffPerc = 0.01537
xPoint = 3.99900  yPoint = 3.57900  TargetValue = 50.98468
PredictValue = 50.97806  DiffPerc = 0.01297
```

```
xPoint = 3.99900  yPoint = 3.59900  TargetValue = 50.96760
PredictValue = 50.96257  DiffPerc = 0.00986
xPoint = 3.99900  yPoint = 3.61900  TargetValue = 50.94433
PredictValue = 50.94111  DiffPerc = 0.00632
xPoint = 3.99900  yPoint = 3.63900  TargetValue = 50.91503
PredictValue = 50.91368  DiffPerc = 0.00265
xPoint = 3.99900  yPoint = 3.65900  TargetValue = 50.87988
PredictValue = 50.88029  DiffPerc = 8.08563
xPoint = 3.99900  yPoint = 3.67900  TargetValue = 50.83910
PredictValue = 50.84103  DiffPerc = 0.00378
xPoint = 3.99900  yPoint = 3.69900  TargetValue = 50.79296
PredictValue = 50.79602  DiffPerc = 0.00601
xPoint = 3.99900  yPoint = 3.71900  TargetValue = 50.74175
PredictValue = 50.74548  DiffPerc = 0.00735
xPoint = 3.99900  yPoint = 3.73900  TargetValue = 50.68579
PredictValue = 50.68971  DiffPerc = 0.00773
xPoint = 3.99900  yPoint = 3.75900  TargetValue = 50.62546
PredictValue = 50.62910  DiffPerc = 0.00719
xPoint = 3.99900  yPoint = 3.77900  TargetValue = 50.56112
PredictValue = 50.56409  DiffPerc = 0.00588
xPoint = 3.99900  yPoint = 3.79900  TargetValue = 50.49319
PredictValue = 50.49522  DiffPerc = 0.00402
xPoint = 3.99900  yPoint = 3.81900  TargetValue = 50.42211
PredictValue = 50.42306  DiffPerc = 0.00188
xPoint = 3.99900  yPoint = 3.83900  TargetValue = 50.34834
PredictValue = 50.34821  DiffPerc = 2.51335
xPoint = 3.99900  yPoint = 3.85900  TargetValue = 50.27233
PredictValue = 50.27126  DiffPerc = 0.00213
xPoint = 3.99900  yPoint = 3.87900  TargetValue = 50.19459
PredictValue = 50.19279  DiffPerc = 0.00358
xPoint = 3.99900  yPoint = 3.89900  TargetValue = 50.11560
PredictValue = 50.11333  DiffPerc = 0.00452
xPoint = 3.99900  yPoint = 3.91900  TargetValue = 50.03587
PredictValue = 50.03337  DiffPerc = 0.00499
```

```
xPoint = 3.99900  yPoint = 3.93900  TargetValue = 49.95591
PredictValue = 49.95333  DiffPerc = 0.00517
xPoint = 3.99900  yPoint = 3.95900  TargetValue = 49.87624
PredictValue = 49.87355  DiffPerc = 0.00538
xPoint = 3.99900  yPoint = 3.97900  TargetValue = 49.79735
PredictValue = 49.79433  DiffPerc = 0.00607
xPoint = 3.99900  yPoint = 3.99900  TargetValue = 49.71976
PredictValue = 49.71588  DiffPerc = 0.00781

maxErrorPerc = 0.06317757842407223
averErrorPerc = 0.007356218626151153
```

Figure 12-5 shows the projected chart of the testing results.

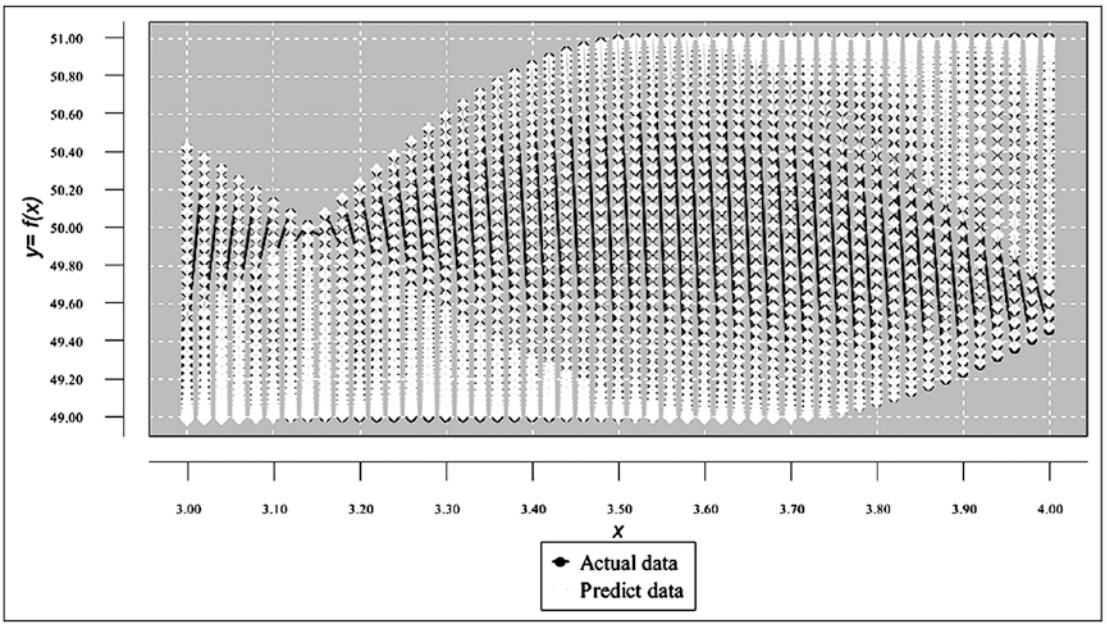


Figure 12-5. *Projected chart of the testing results*

The approximation results are acceptable; therefore, there is no need to use the micro-batch method.

Summary

This chapter discussed how to use a neural network to approximate functions in 3D space. You learned that the approximation of functions in 3D space (functions with two variables) is done similarly to the approximation of functions in 2D space. The only difference is that the network architecture should include two inputs, and the training and testing data sets should include records for all the possible combinations of x and y values. These results can be extended for the approximation of functions with more than two variables.

Index

I. Livshin, *Artificial Neural Networks with Java*, https://doi.org/10.1007/978-1-4842-4421-0

E

F

G, H

I

J, K

L

M

N

O

P, Q

R

S

T, U, V

W

X, Y, Z