Chapter 1: Overview of Neural Networks

- Understanding Biological Neural Networks
- How an Artificial Neural Network is Constructed
- Appropriate Uses for Neural Networks

Computers can perform many operations considerably faster than a human being. However, faster is not always better for problem solving. There are many tasks for which the computer falls considerably short of its human counterpart. There are numerous examples of this. For instance, given two pictures, a preschool child can easily tell the difference between a cat and a dog. Yet, this same simple task is extremely difficult for today's computers.

The goal of this book is to teach the reader how to construct neural networks using the Java programming language. As with any technology, it is just as important to know when not to use neural networks as it is to understand when they should be used. This chapter provides an introduction to the appropriate uses of neural networks, and explains which programming requirements are conducive to their use.

This chapter begins with a brief introduction to the structure of neural networks. This discussion provides an overview of neural network architecture, and explains how a typical neural network is constructed. Following this introduction is a discussion of how a neural network is trained. Finally, an overview is provided of the ultimate task, validating a trained neural network.

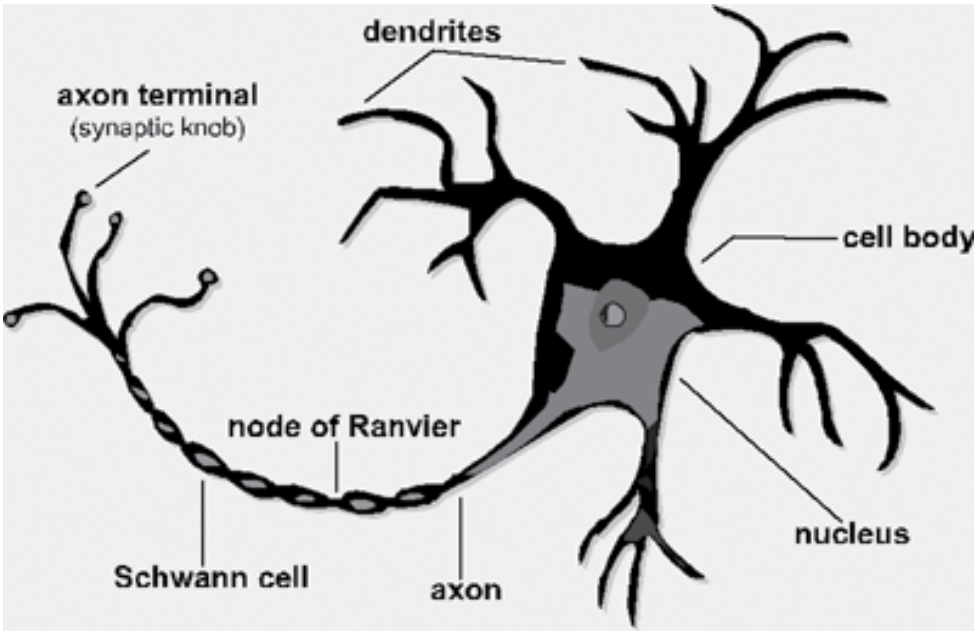
How is a Biological Neural Network Constructed?

The term neural network, as it is normally used, is actually a misnomer. Computers attempt to simulate biological neural networks by implementing artificial neural networks. However, most publications use the term "neural network," rather than "artificial neural network" (ANN). This book follows suit. Unless the term "neural network" is explicitly prefixed with the terms "biological" or "artificial" you can assume that the term "artificial neural network" is intended. To explore this distinction, you will first be shown the structure of a biological neural network.

To construct a computer capable of "human-like thought," researchers have used the only working model they have available—the human brain. However, the human brain as a whole is far too complex to model. Rather, the individual cells that make up the human brain are studied. At the most basic level, the human brain is composed primarily of neuron cells. They are the basic building blocks of the human brain. Artificial neural networks attempt to simulate the behavior of these cells.

A neuron cell, as seen in Figure 1.1, accepts signals from dendrites. When a neuron accepts a signal, that neuron may fire. When a neuron fires, a signal is transmitted over the neuron's axon. Ultimately, the signal will leave the neuron as it travels to the axon terminals. The signal is then transmitted to other neurons or nerves.

Figure 1.1: A neuron cell.

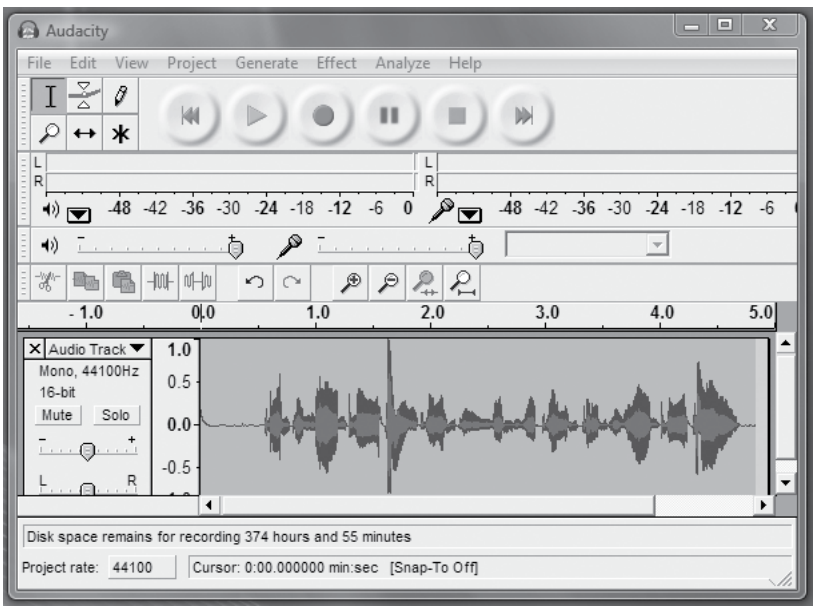


This signal, transmitted by the neuron, is an analog signal. Most modern computers are digital machines, and thus, require a digital signal. A digital computer processes information as either off or on, using the binary digits zero and one, respectively. The presence of an electric signal is indicated with a value of one, whereas the absence of an electrical signal is indicated with a value of zero. Figure 1.2 shows a digital signal.

Figure 1.2: A digital signal.

Some of the early computers were analog, rather than digital. An analog computer uses a much wider range of values than zero and one. This wider range is achieved by increasing or decreasing the voltage of the signal. Figure 1.3 shows an analog signal. Though analog computers are useful for certain simulation activities, they are not suited to processing the large volumes of data that digital computers are typically required to process. Thus, nearly every computer in use today is digital.

Figure 1.3: Sound recorder showing an analog file.



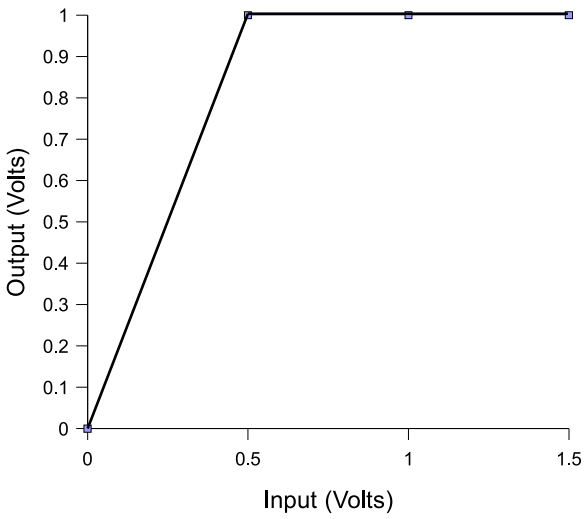
Biological neural networks are analog. As you will see in the next section, simulating analog neural networks on a digital computer can present some challenges. Neurons accept an analog signal through their dendrites, as seen in Figure 1.1. Because this signal is analog, the voltage of each signal will vary. If the voltage is within a

certain range, the neuron will fire. When a neuron fires, a new analog signal is transmitted from the firing neuron to other neurons. This signal is conducted over the firing neuron's axon. The regions of input and output are called synapses. Later, in chapter 5, The Feedforward Backpropagation Neural Network example will demonstrate that the synapses are the interface between a program and a neural network.

A neuron makes a decision by firing or not firing. The decisions being made are extremely low-level decisions. It requires a large number of decisions to be made by many neurons just to read this sentence. Higher-level decisions are the result of the collective input and output of many neurons.

Decisions can be represented graphically by charting the input and output of neurons. Figure 1.4 illustrates the input and output of a particular neuron. As you will be shown in chapter 5, there are different types of neurons, all of which have differently shaped output graphs. Looking at the graph shown in Figure 1.4, it can be seen that the neuron in this example will fire at any input greater than 0.5 volts.

Figure 1.4: Activation levels of a neuron.



A biological neuron is capable of making basic decisions. Artificial neural networks are based on this model. Following is an explanation of how this model is simulated using a digital computer.

Solving Problems with Neural Networks

A significant goal of this book is to show you how to construct neural networks and to teach you when to use them. As a programmer of neural networks, you must understand which problems are well suited for neural network solutions and which are not. An effective neural network programmer also knows which neural network structure, if any, is most applicable to a given problem. This section begins by first focusing on those problems that are not conducive to a neural network solution.

Problems Not Suited to a Neural Network Solution

Programs that are easily written out as flowcharts are examples of problems for which neural networks are not appropriate. If your program consists of well-defined steps, normal programming techniques will suffice.

Another criterion to consider is whether the logic of your program is likely to change. One of the primary features of neural networks is their ability to learn. If the algorithm used to solve your problem is an unchanging business rule, there is no reason to use a neural network. In fact, it might be detrimental to your application if the neural network attempts to find a better solution, and begins to diverge from the desired process and produces unexpected results.

Finally, neural networks are often not suitable for problems in which you must know exactly how the solution was derived. A neural network can be very useful for solving the problem for which it was trained, but the neural network cannot explain its reasoning. The neural network knows something because it was trained to know it. The neural network cannot explain how it followed a series of steps to derive the answer.

Problems Suited to a Neural Network

Although there are many problems for which neural networks are not well suited, there are also many problems for which a neural network solution is quite useful. In addition, neural networks can often solve problems with fewer lines of code than a traditional programming algorithm. It is important to understand which problems call for a neural network approach.

Neural networks are particularly useful for solving problems that cannot be expressed as a series of steps, such as recognizing patterns, classification, series prediction, and data mining.

Pattern recognition is perhaps the most common use for neural networks. For this type of problem, the neural network is presented a pattern. This could be an image, a sound, or any other data. The neural network then attempts to determine if the input data matches a pattern that it has been trained to recognize. Chapter 3, Using a Hopfield Neural Network, provides an example of a simple neural network that recognizes input patterns.

Classification is a process that is closely related to pattern recognition. A neural network trained for classification is designed to take input samples and classify them into groups. These groups may be fuzzy, lacking clearly defined boundaries. Alternatively, these groups may have quite rigid boundaries. Chapter 12, OCR and the Self-Organizing Map, introduces an example program capable of optical character recognition (OCR). This program takes handwriting samples and classifies them by letter (e.g., the letter "A" or "B").

Training Neural Networks

The individual neurons that make up a neural network are interconnected through their synapses. These connections allow the neurons to signal each other as information is processed. Not all connections are equal. Each connection is assigned a connection weight. If there is no connection between two neurons, then their connection weight is zero. These weights are what determine the output of the neural network; therefore, it can be said that the connection weights form the memory of the neural network.

Training is the process by which these connection weights are assigned. Most training algorithms begin by assigning random numbers to a weights matrix. Then, the validity of the neural network is examined. Next, the weights are adjusted based on how well the neural network performed and the validity of the results. This process is repeated until the validation error is within an acceptable limit. There are many ways to train neural networks. Neural network training methods generally fall into the categories of supervised, unsupervised, and various hybrid approaches.

Supervised training is accomplished by giving the neural network a set of sample data along with the anticipated outputs from each of these samples. Supervised training is the most common form of neural network training. As supervised training proceeds, the neural network is taken through a number of iterations, or epochs, until the output of the neural network matches the anticipated output, with a reasonably small rate of error. Each epoch is one pass through the training samples.

Unsupervised training is similar to supervised training, except that no anticipated outputs are provided. Unsupervised training usually occurs when the neural network is being used to classify inputs into several groups. The training involves many epochs, just as in supervised training. As the training progresses, the classification groups are "discovered" by the neural network. Unsupervised training is covered in chapter 11, Using a Self-Organizing Map.

There are several hybrid methods that combine aspects of both supervised and unsupervised training. One such method is called reinforcement training. In this method, a neural network is provided with sample data that does not contain anticipated outputs, as is done with unsupervised training. However, for each output, the neural network is told whether the output was right or wrong given the input.

It is very important to understand how to properly train a neural network. This book explores several methods of neural network training, including backpropagation, simulated annealing, and genetic algorithms. Chapters 4 through 7 are dedicated to the training of neural networks. Once the neural network is trained, it must be validated to see if it is ready for use.

Validating Neural Networks

The final step, validating a neural network, is very important because it allows you to determine if additional training is required. To correctly validate a neural network, validation data must be set aside that is completely separate from the training data.

As an example, consider a classification network that must group elements into three different classification groups. You are provided with 10,000 sample elements. For this sample data, the group that each element should be classified into is known. For such a system, you would randomly divide the sample data into two groups of 5,000 elements each. The first group would form the training set. Once the network was properly trained, the second group of 5,000 elements would be used to validate the neural network.

It is very important that a separate group of data always be maintained for validation. First, training a neural network with a given sample set and also using this same set to predict the anticipated error of the neural network for a new arbitrary set will surely lead to bad results. The error achieved using the training set will almost always be substantially lower than the error on a new set of sample data. The integrity of the validation data must always be maintained.

This brings up an important question. What happens if the neural network that you have just finished training performs poorly on the validation data set? If this is the case, then you must examine possible causes. It could mean that the initial random weights were not appropriate. Rerunning the training with new initial weights could correct this. While an improper set of initial random weights could be the cause, a more likely possibility is that the training data was not properly chosen.

If the validation is performing poorly, it is likely that there was data present in the validation set that was not available in the training data. The way this situation should be rectified is to try a different random approach to separating the data into training and validation sets. If this fails, you must combine the training and validation sets into one large training set. New data must then be acquired to serve as the validation data.

In some situations it may be impossible to gather additional data to use as either training or validation data. If this is the case, then you are left with no other choice but to combine all or part of the validation set with the training set. While this approach will forgo the security of a good validation, if additional data cannot be acquired this may be your only alternative.

Problems Commonly Solved With Neural Networks

There are many different problems that can be solved with a neural network. However, neural networks are commonly used to address particular types of problems. The following four types of problem are frequently solved with neural networks:

- Classification
- Prediction
- Pattern recognition
- Optimization

These problems will be discussed briefly in the following sections. Many of the example programs throughout this book will address one of these four problems.

Classification

Classification is the process of classifying input into groups. For example, an insurance company may want to classify insurance applications into different risk categories, or an online organization may want its email system to classify incoming mail into groups of spam and non-spam messages.

Often, the neural network is trained by presenting it with a sample group of data and instructions as to which group each data element belongs. This allows the neural network to learn the characteristics that may indicate group membership.

Prediction

Prediction is another common application for neural networks. Given a time-based series of input data, a neural network will predict future values. The accuracy of the guess will be dependent upon many factors, such as the quantity and relevancy of the input data. For example, neural networks are commonly applied to problems involving predicting movements in financial markets.

This book will demonstrate several examples of prediction. Chapter 9, Predictive Neural Networks, provides an introductory explanation of how to use a neural network to make predictions. Chapter 10 shows a basic neural approach to analyzing the S&P 500.

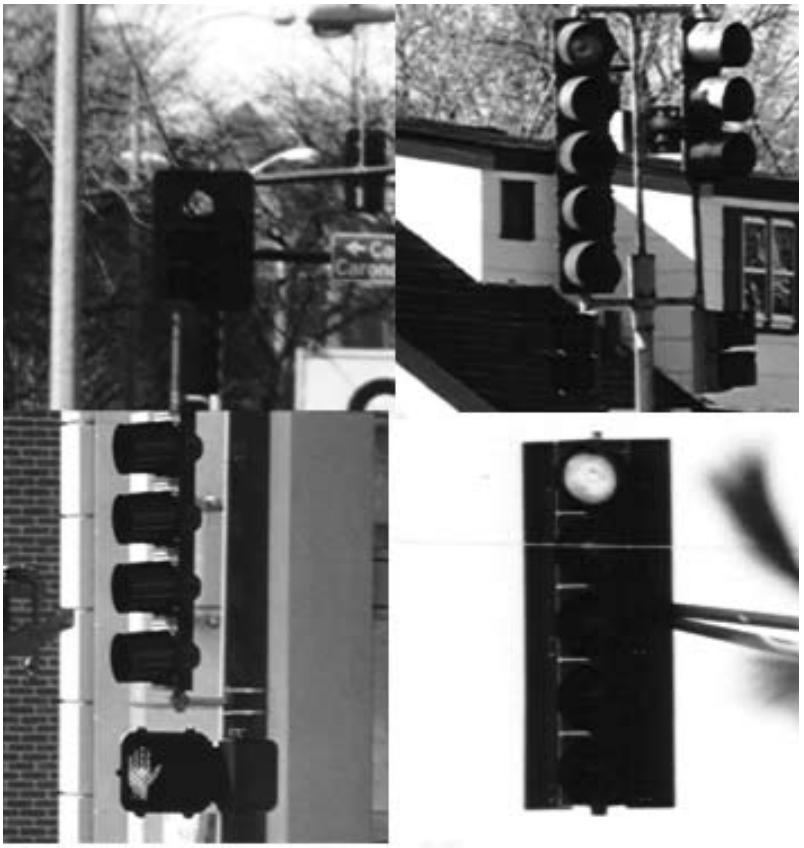
Pattern Recognition

Pattern recognition is one of the most common uses for neural networks. Pattern recognition is a form of classification. Pattern recognition is simply the ability to recognize a pattern. The pattern must be recognized even when it is distorted. Consider the following everyday use of pattern recognition.

Every person who holds a driver's license should be able to accurately identify a traffic light. This is an extremely critical pattern recognition procedure carried out by countless drivers every day. However, not every traffic light looks the same, and the appearance of a particular traffic light can be altered depending on the time of day or the season. In addition, many variations of the traffic light exist. Still, recognizing a traffic light is not a hard task for a human driver.

How hard is it to write a computer program that accepts an image and tells you if it is a traffic light? Without the use of neural networks, this could be a very complex task. Figure 1.5 illustrates several different traffic lights. Most common programming algorithms are quickly exhausted when presented with a complex pattern recognition problem.

Figure 1.5: Different Traffic Lights



Later in this book, an example will be provided of a neural network that reads handwriting. This neural network accomplishes the task by recognizing patterns in the individual letters drawn.

Optimization

Another common use for neural networks is optimization. Optimization can be applied to many different problems for which an optimal solution is sought. The neural network may not always find the optimal solution; rather, it seeks to find an acceptable solution. Optimization problems include circuit board assembly, resource allocation, and many others.

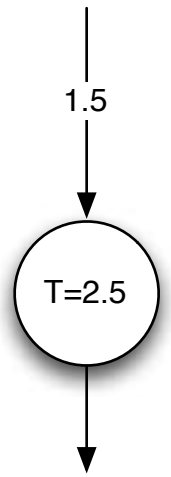
Perhaps one of the most well-known optimization problems is the traveling salesman problem (TSP). A salesman must visit a set number of cities. He would like to visit all cities and travel the fewest number of miles possible. With only a few cities, this is not a complex problem. However, with a large number of cities, brute force methods of calculation do not work nearly as well as a neural network approach.

Using a Simple Neural Network

Following is an example of a very simple neural network. Though the network is simple, it includes nearly all of the elements of the more complex neural networks that will be covered later in this book.

First, consider an artificial neuron, as shown in Figure 1.6.

Figure 1.6: Artificial neuron.



There are two attributes associated with this neuron: the threshold and the weight. The weight is 1.5 and the threshold is 2.5. An incoming signal will be amplified, or de-amplified, by the weight as it crosses the incoming synapse. If the weighted input exceeds the threshold, then the neuron will fire.

Consider a value of one (**`true`**) presented as the input to the neuron. The value of one will be multiplied by the weight value of 1.5. This results in a value of 1.5. The value of 1.5 is below the threshold of 2.5, so the neuron will not fire. This neuron will never fire with Boolean input values. Not all neurons accept only boolean values. However, the neurons in this section only accept the boolean values of one (**`true`**) and zero (**`false`**).

A Neural Network for the And Operator

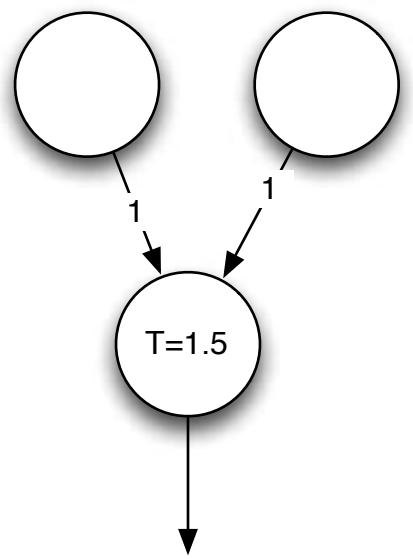
The neuron shown in Figure 1.6 is not terribly useful. However, most neurons are not terribly useful—at least not independently. Neurons are used with other neurons to form networks. We will now look at a neural network that acts as an **AND** gate. Table 1.1 shows the truth table for the **AND** logical operation.

Table 1.1: The AND Logical Operation

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

A simple neural network can be created that recognizes the **AND** logical operation. There will be three neurons in total. This network will contain two inputs and one output. A neural network that recognizes the **AND** logical operation is shown in Figure 1.7.

Figure 1.7: A neural network that recognizes the AND logical operation.



There are two inputs to the network shown in Figure 1.7. Each neuron has a weight of one. The threshold is 1.5. Therefore, a neuron will only fire if both inputs are **`true`**. If either input is **`false`**, the sum of the two inputs will not exceed the threshold of 1.5.

Consider inputs of **`true`** and **`false`**. The **`true`** input will send a value of one to the output neuron. This is below the threshold of 1.5. Likewise, consider inputs of **`true`** and **`true`**. Each input neuron will send a value of one. These two inputs are summed by the output neuron, resulting in two. The value of two is greater than 1.5, therefore, the neuron will fire.

A Neural Network for the Or Operation

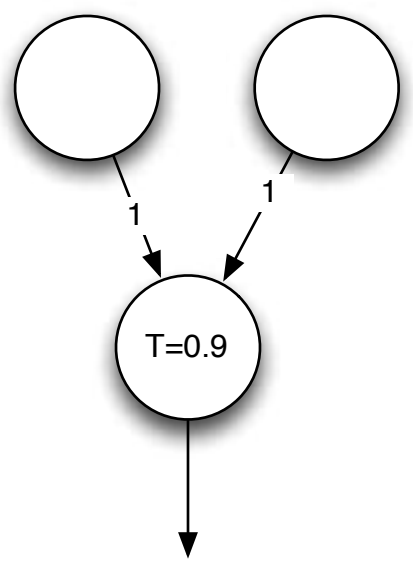
Neural networks can be created to recognize other logical operations as well. Consider the **OR** logical operation. The truth table for the **OR** logical operation is shown in Table 1.2. The **OR** logical operation is **`true`** if either input is **`true`**.

Table 1.2: The OR Logical Operation

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

The neural network that will recognize the **OR** operation is shown in Figure 1.8.

Figure 1.8: A neural network that recognizes the OR logical operation.



The **OR** neural network looks very similar to the **AND** neural network. The biggest difference is the threshold value. Because the threshold is lower, only one of the inputs needs to have a value of **`true`** for the output neuron to fire.

A Neural Network for the XOR Operation

Next we will consider a neural network for the exclusive or (**XOR**) logical operation. The **XOR** truth table is shown in Table 1.3.

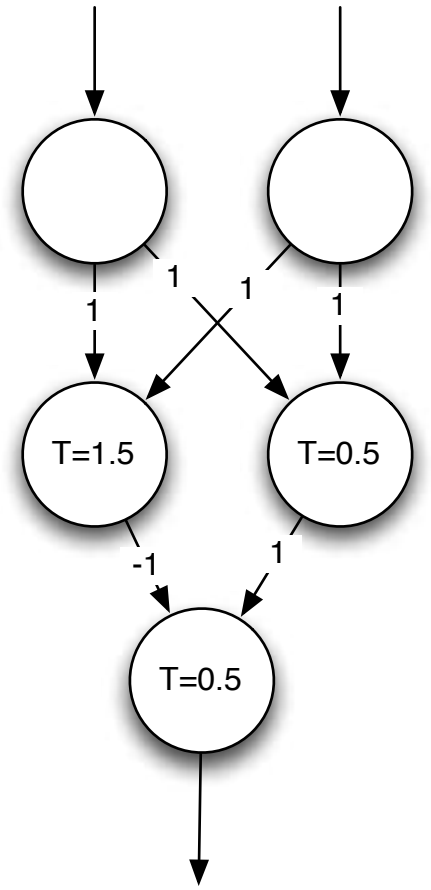
Table 1.3: The XOR Logical Operation

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

The **XOR** logical operation requires a slightly more complex neural network than the **AND** and **OR** operators. The neural networks presented so far have had only two layers— an input layer and an output layer. More complex neural networks also include one or more hidden layers. The **XOR** operator requires a hidden layer. As a result, the **XOR** neural network often becomes a sort of "Hello World" application for neural networks. You will see the **XOR** operator again in this book as different types of neural network are introduced and trained.

Figure 1.9 shows a three-layer neural network that can be used to recognize the **XOR** operator.

Figure 1.9: A neural network that recognizes the XOR logical operation.



Consider the case in which the values of **`true`** and **`true`** are presented to this neural network. Both neurons in the hidden layer receive the value of two. This is above the thresholds of both of the hidden layer neurons, so they will both fire. However, the first hidden neuron has a weight of -1, so its contribution to the output neuron is -1. The second neuron has a weight of 1, so its contribution to the output neuron is 1. The sum of 1 and -1 is zero. Zero is below the threshold of the output neuron, so the output neuron does not fire. This is consistent with the **XOR** operation, because it will produce **`false`** if both inputs are **`true`**.

Now consider if the values of **`false`** and **`true`** are presented to the neural network. The input to the first hidden layer neuron will be 1, from the second input neuron. This is lower than the threshold of 1.5, so it will not fire. The input to the second hidden layer neuron will also be 1, from the second input neuron. This is over the 0.5

threshold, so it will fire. The input to the output neuron will be zero from the left hidden neuron and 1 from the right hidden neuron. This is greater than 0.5, so the output neuron will fire. This is consistent with the **XOR** operation, because it will produce **`true`** if one of the input neurons is **`true`** and the other **`false`**.

Of course, the neural networks shown in the preceding sections are very simple. However, they illustrate all of the key points for more complex neural networks. Future chapters will introduce additional types of neural networks; however, neural networks will almost always feature weights and thresholds.

Chapter Summary

Computers can process information considerably faster than human beings. Yet, a computer is incapable of performing many of the same tasks that a human can easily perform. For processes that cannot easily be broken into a finite number of steps, a neural network can be an ideal solution.

The term neural network typically refers to an artificial neural network. An artificial neural network attempts to simulate the biological neural networks contained in the brains of all animals. Artificial neural networks were first introduced in the 1950's and through the years of their development have experienced numerous setbacks; they have yet to deliver on the promise of simulating human thought.

Neural networks are constructed of neurons that form layers. Input is presented to the layers of neurons. If the input to a neuron is within the range that the neuron has been trained for, then the neuron will fire. When a neuron fires, a signal is sent to the layers of neurons to which the firing neuron is connected. The connections between neurons are called synapses. Java can be used to construct such a network.

Neural networks must be trained and validated. A training set is usually split in half to provide both a training and validation set. Training the neural network consists of running the neural network over the training data until the neural network learns to recognize the training set with a sufficiently low error rate. Validation occurs when the neural network's results are checked.

Just because a neural network can process the training data with a low rate of error, does not mean the neural network is trained and ready for use. Before the neural network is placed into production use, it must be validated. Validation involves presenting the validation set to the neural network and comparing the actual results produced by the neural network with the anticipated results.

The neural network is ready to be placed into production if, at the end of the validation process, the results from the validation run meet a satisfactory error level. If the results are not satisfactory, then the neural network will have to be retrained before it can be placed into production.

Neural networks are comprised of many neurons. Their threshold and weight values are combined into weight matrixes. A weight matrix is stored in a regular mathematical matrix. Chapter 2 will introduce several Java classes designed to store matrix values and perform matrix mathematics. The neural networks in this book will be built upon these matrix classes.

Vocabulary

Activation Level

Analog Computer

Artificial Intelligence

Artificial Neural Network

Axon

Binary

Biological Neural Network

Classification

Dendrite

Digital Computer

Fire

Hidden Layer

Input Layer

Layer

Matrix

Neural Network

Neuron

Output Layer

Pattern Recognition

Prediction

Supervised Training

Signal

Synapse

Thresholds

Training

Truth Table

Unsupervised Training

Weight Matrix

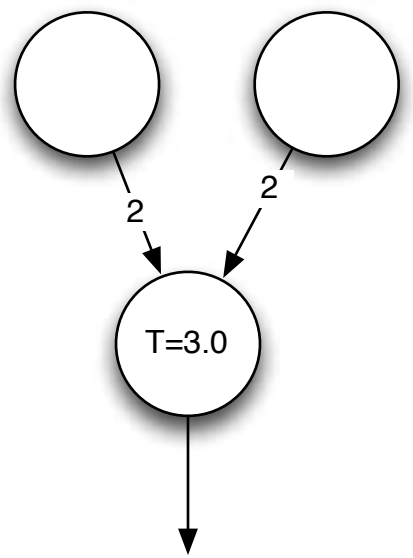
Validation

XOR

Questions for Review

1. What types of problems are neural networks better able to address than traditional programming practices?

2. Shown below is a simple neural network for an operator. Write the truth table for this neural network. What operator is this?



3. Explain the purpose of a classification neural network. What sort of "real world" problem might a classification neural network be used for?

4. What is the purpose of a threshold value?

5. Explain the difference between supervised and unsupervised training.