CHAPTER 9: PREDICTIVE NEURAL NETWORKS

- Creating Input and Output Neurons for Prediction
- How to Create Training Sets for a Predictive Neural Network
- Predicting the Sine Wave

Neural networks are particularly good at recognizing patterns. Pattern recognition can be used to predict future patterns in data. A neural network used to predict future patterns is call a predictive, or temporal neural network. A predictive neural network can be used to predict future events, such as stock market trends and sun spot cycles.

This chapter will introduce predictive neural networks through an example network coded to predict the sine wave. The next chapter will present a neural network that attempts to predict the S&P 500.

How to Predict with a Neural Network

Many different kinds of neural networks can be used for prediction. This book will use the feedforward neural network to attempt to learn patterns in data and predict future values. Like all problems applied to neural networks, prediction is a matter of intelligently determining how to configure input and interpret output neurons for a problem.

There are many ways to model prediction problems. This book will focus on one specific technique, which involves taking known data, partitioning it into training sets, and applying it to input neurons. A smaller number of output neurons then represent the future data. Following is a discussion of how to set up input and output neurons for a simple prediction.

Setting Up Input and Output Neurons for Prediction

Consider a simple series of numbers, such as the sequence shown here:

1, 2, 3, 4, 3, 2, 1, 2, 3, 4, 3, 2, 1

A neural network that predicts numbers from this sequence might use three input neurons and a single output neuron. For example, a training set might look like Table 9.1.

Table 9.1: Sample Training Sets for a Predictive Neural Network

Set	Input	Ideal Output
1	1,2,3	4
2	2,3,4	3
3	3,4,3	2
4	4,3,2	1

As you can see, the neural network is prepared to receive several data samples in a sequence. The output neuron then predicts how the sequence will be continued. The idea is that you can now feed any sequence of three numbers, and the neural network will predict the fourth number. Each data point is called a time slice. Therefore, each input neuron represents a known time slice. The output neurons represent future time slices.

Of course this is a very simple example. It is possible to include additional data that might influence the next value. In chapter 10, we will attempt to predict the S&P 500 and will include the current interest rate.

Selecting Data for Prediction

There are several methods for selecting input data for prediction. In the above example, each training set value occurred directly adjacent to the next. This is not always possible. The amount of input data may produce too many training sets. This will be the case for the neural network presented in the next chapter, since we will have easy access to the S&P 500 financial information going all the way back to 1950, as well as prime interest rates. This will be more data than will be practical to work with to train the neural network.

In the example presented in the table above, the actual values are simply input into the neural network. For a simple example, such as the sine wave, this works just fine. However, you may want to feed the neural network percentage increases and cause the output neurons to predict future percentage increases or decreases. This technique will be expanded upon in chapter 10 when a predictive neural network is applied to the S&P 500.

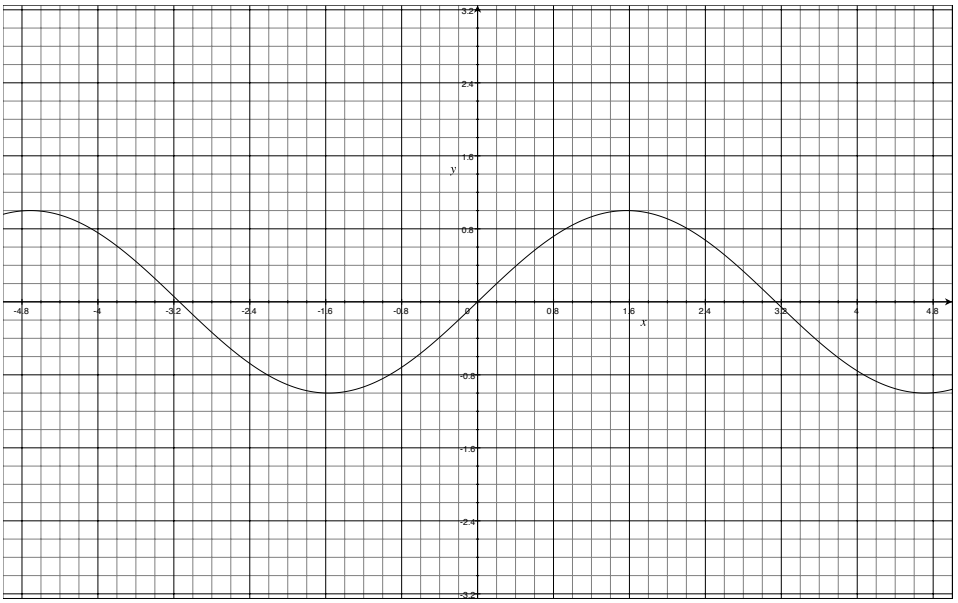
Another consideration for the training set data is to leave enough data to be able to evaluate the performance of the predictive neural network. For example, if the goal is to train a neural network for stock market prediction, perhaps only data prior to 2005 should be used to train the neural network. Everything past 2005 can then be used to evaluate how well the trained neural network is performing.

Predicting the Sine Wave

The example in this chapter is relatively simple. A neural network is presented that predicts the sine wave. The sine wave is mathematically predictable, so in many ways it is not a good example for illustrating a predictive neural network; however, the sine wave is easily understood and varies over time. This makes it a good introduction to predictive neural networks. Chapter 10 will expand upon this and use a neural network to attempt to provide some insight into stock market prediction.

The sine wave can be seen by plotting the trigonometric sine function. Figure 9.1 shows the sine wave.

Figure 9.1: The sine wave.



The sine wave function is used to begin training the neural network. Backpropagation is used to train for the sine wave predictor. When the sine wave example is first executed, you will see the results of the training process. Listing 9.1 shows typical output from the sine wave predictor's training process.

Listing 9.1: Training the Sine Wave Predictor

```
Iteration #1 Error:0.48120350975475823 Iteration #2 Er-
ror:0.36753445768855236 Iteration #3 Error:0.3212066601426759
Iteration #4 Error:0.2952410514715732 Iteration #5 Er-
ror:0.2780102928778258 Iteration #6 Error:0.26556861969786527
Iteration #7 Error:0.25605359706505776 Iteration #8 Er-
```

```
ror:0.24842242500053566 Iteration #9 Error:0.24204767544134156 It-
eration #10 Error:0.23653845782593882
...
Iteration #4990 Error:0.02319397662897425 Iteration #4991 Er-
ror:0.02319310934886356 Iteration #4992 Error:0.023192242246688515
Iteration #4993 Error:0.02319137532183077 Iteration #4994 Er-
ror:0.023190508573672858 Iteration #4995 Error:0.02318964200159761
Iteration #4996 Error:0.02318877560498862 Iteration #4997 Er-
ror:0.02318790938322986 Iteration #4998 Error:0.023187043335705867
Iteration #4999 Error:0.023186177461801745
```

In the beginning, the error rate is fairly high at 48%. This quickly begins to fall off to 36.7% by the second iteration. By the time the 4,999th iteration has occurred, the error rate has fallen to 2.3%. The program is designed to stop before hitting the 5,000th iteration. This succeeds in reducing the error rate to less than 3%.

Additional training would produce a better error rate; however, by limiting the iterations, the program is able to finish in only a few minutes on a regular computer. This program took about two minutes to execute on an Intel Core2 Dual 2mghtz computer.

Once the training is complete, the sine wave is presented to the neural network for prediction. The output from this prediction is shown in Listing 9.2.

Listing 9.2: Predicting the Sine Wave

```
5:Actual=0.76604:Predicted=0.7892166200864351:Difference=2.32% 6:A
ctual=0.86602:Predicted=0.8839210963512845:Difference=1.79% 7:Ac
tual=0.93969:Predicted=0.934526031234053:Difference=0.52% 8:Act
ual=0.9848:Predicted=0.9559577688326862:Difference=2.88% 9:Actu
al=1.0:Predicted=0.9615566601973113:Difference=3.84% 10:Actual=
0.9848:Predicted=0.9558060932656686:Difference=2.90% 11:Actual=
0.93969:Predicted=0.9354447787244102:Difference=0.42% 12:Actual
=0.86602:Predicted=0.8894014978439005:Difference=2.34% 13:Actua
l=0.76604:Predicted=0.801342405700056:Difference=3.53% 14:Actua
l=0.64278:Predicted=0.6633506809125252:Difference=2.06% 15:Actu
al=0.49999:Predicted=0.4910483600917853:Difference=0.89% 16:Act
ual=0.34202:Predicted=0.31286152780645105:Difference=2.92% 17:A
ctual=0.17364:Predicted=0.14608325263568134:Difference=2.76%
18:Actual=0.0:Predicted=-0.008360016796238434:Difference=0.84%
19:Actual=-0.17364:Predicted=-0.15575381460132823:Difference=1.79%
20:Actual=-0.34202:Predicted=-0.3021775158559559:Difference=3.98%
...
490:Actual=-0.64278:Predicted=-0.6515076637590029:Difference=0.87%
491:Actual=-0.76604:Predicted=-0.8133333939237001:Difference=4.73%
492:Actual=-0.86602:Predicted=-0.9076496572125671:Difference=4.16%
493:Actual=-0.93969:Predicted=-0.9492579517460149:Difference=0.96%
```

```
494:Actual=-0.9848:Predicted=-0.9644567437192423:Difference=2.03%
495:Actual=-1.0:Predicted=-0.9664801515670861:Difference=3.35%
496:Actual=-0.9848:Predicted=-0.9579489752650393:Difference=2.69%
497:Actual=-0.93969:Predicted=-0.9340105440194074:Difference=0.57%
498:Actual=-0.86602:Predicted=-0.8829925066754494:Difference=1.70%
499:Actual=-0.76604:Predicted=-0.7913823031308845:Difference=2.53%
```

As you can see, both the actual and predicted values are shown for each element. The neural network was only trained for the first 250 elements; yet, the neural network is able to predict well beyond these first 250. You will also notice that the difference between the actual values and the predicted values rarely exceeds 3%.

Obtaining Sine Wave Data

A class named **`ActualData`** is provided that will hold the actual "calculated" data for the sine wave. The **`ActualData`** class also provides convenience methods that can be used to construct the training sets. The **`ActualData`** class is shown in Listing 9.3.

Listing 9.3: Actual Sine Wave Data (ActualData.java)

```
package com.heatonresearch.book.introneuralnet.ch9.predict;

/**
 * Chapter 9: Predictive Neural Networks
 *
 * ActualData: Holds values from the sine wave.
 *
 * @author Jeff Heaton
 * @version 2.1
 */
public class ActualData {
      public static double sinDEG(final double deg) {
            final double rad = deg * (Math.PI / 180);
            final double result = Math.sin(rad);
            return ((int) (result * 100000.0)) / 100000.0;
      }

      private final double actual[];
      private final int inputSize;

      private final int outputSize;

      public ActualData(final int size, final int inputSize,
            final int outputSize) {
            this.actual = new double[size];
            this.inputSize = inputSize;
            this.outputSize = outputSize;
```

```
            int angle = 0;
            for (int i = 0; i < this.actual.length; i++) {
                  this.actual[i] = sinDEG(angle);
                  angle += 10;
            }
      }

      public void getInputData(final int offset,
            final double target[]) {
            for (int i = 0; i < this.inputSize; i++) {
                  target[i] = this.actual[offset + i];
            }
      }

      public void getOutputData(final int offset,
            final double target[]) {
            for (int i = 0; i < this.outputSize; i++) {
                  target[i] = this.actual[offset +
                  this.inputSize + i];
            }
      }


}
```

The constructor for the **ActualData** class initializes the internal variables to the values of the sine wave. The signature for the constructor for the **ActualData** class is shown here:

```
public ActualData(final int size,
final int inputSize,
final int outputSize)
```

Three arguments are passed into the **ActualData** constructor. The **size** argument defines how many values of the sine wave will be considered. The **inputSize** argument specifies the number of input neurons. The **outputSize** argument specifies the number of output neurons.

First, a **double** array is allocated that is large enough to hold all of the requested sine wave values. Then, the arguments are copied to instance variables.

```
this.actual = new double[size];
this.inputSize = inputSize;
this.outputSize = outputSize;
```

The sine wave data starts at angle zero.

```
int angle = 0;
```

We then loop through the number of requested values.

```
for (int i = 0; i < this.actual.length; i++) {
```

For each requested value, the sine is calculated using an angle expressed in degrees.

```
  this.actual[i] = sinDEG(angle);
  angle += 10;
}
```

The looping continues with an angle increment of 10 degrees.

Constructing Training Sets for the Sine Wave

As you will recall from previous chapters, you can provide training sets to train the feedforward neural network. These training sets consist of two arrays. The first array specifies input values for the neural network. The second array specifies the ideal outputs for each of the input values. The **ActualData** class provides two methods to retrieve each of these arrays. The first method, named **getInputData**, is shown here:

```
public void getInputData(final int offset,
final double target[])
```

The **offset** variable specifies the offset in the actual data at which copying begins. The **target** array will receive the actual values. The **getInputData** method simply continues by copying the appropriate values into the **target** array.

```
for (int i = 0; i < this.inputSize; i++) {
  target[i] = this.actual[offset + i];
}
```

The **getOutputData** method retrieves the ideal values with which the neural network will be trained. The signature for the **getOutputData** method is shown here:

```
public void getOutputData(final int offset,
final double target[])
```

As above, the **offset** variable specifies the offset in the actual data at which to begin copying ideal values. The **target** array will receive the ideal values. The **getInputData** method simply continues by copying the appropriate values into the **target** array.

```
for (int i = 0; i < this.outputSize; i++) {
  target[i] = this.actual[offset + this.inputSize + i];
}
```

The above loop will populate the target array with the actual values. Table 9.2 shows these values.

Table 9.2: Sine Wave Training Data

Input 1	Input 2	Input 3	Input 4	Input 5	Output 1
0.0	0.17364	0.34202	0.49999	0.64278	0.76604
0.17364	0.34202	0.49999	0.64278	0.76604	0.86602
0.34202	0.49999	0.64278	0.76604	0.86602	0.93969
0.49999	0.64278	0.76604	0.86602	0.93969	0.9848
0.64278	0.76604	0.86602	0.93969	0.9848	1.0
0.76604	0.86602	0.93969	0.9848	1.0	0.9848
0.86602	0.93969	0.9848	1.0	0.9848	0.93969
0.93969	0.9848	1.0	0.9848	0.93969	0.86602
0.9848	1.0	0.9848	0.93969	0.86602	0.76604
1.0	0.9848	0.93969	0.86602	0.76604	0.64278
0.9848	0.93969	0.86602	0.76604	0.64278	0.49999
0.93969	0.86602	0.76604	0.64278	0.49999	0.34202
0.86602	0.76604	0.64278	0.49999	0.34202	0.17364
0.76604	0.64278	0.49999	0.34202	0.17364	0.0
0.64278	0.49999	0.34202	0.17364	0.0	-0.17364
0.49999	0.34202	0.17364	0.0	-0.17364	-0.34202
0.34202	0.17364	0.0	-0.17364	-0.34202	-0.5
0.17364	0.0	-0.17364	-0.34202	-0.5	-0.64278
0.0	-0.17364	-0.34202	-0.5	-0.64278	-0.76604
-0.17364	-0.34202	-0.5	-0.64278	-0.76604	-0.86602
-0.34202	-0.5	-0.64278	-0.76604	-0.86602	-0.93969
-0.5	-0.64278	-0.76604	-0.86602	-0.93969	-0.9848
-0.64278	-0.76604	-0.86602	-0.93969	-0.9848	-1.0
-0.76604	-0.86602	-0.93969	-0.9848	-1.0	-0.9848
-0.86602	-0.93969	-0.9848	-1.0	-0.9848	-0.93969
-0.93969	-0.9848	-1.0	-0.9848	-0.93969	-0.86602
-0.9848	-1.0	-0.9848	-0.93969	-0.86602	-0.76604

These training values will be used in the next section to train the neural network.

Training the Sine Wave Predictor

The sine wave predictor uses a hyperbolic tangent activation function, rather than the sigmoid activation function that many of the neural networks in this book have used. This is because the sine function returns numbers between -1 and 1. This can be seen in Table 9.2. The sigmoid function can only handle numbers between 0 and 1, and thus would fail when presented with the values of the sine function.

As mentioned earlier, the neural network is trained with a backpropagation algorithm. Backpropagation was covered in chapter 5, "Feedforward Backpropagation Neural Networks." When using the hyperbolic tangent as an activation function, it is important to use a low learning rate and momentum. Otherwise, the adjustments will be too large and the network may fail to converge on an acceptable error rate.

The training occurs in the **trainNetworkBackprop** method. The signature for the **trainNetworkBackprop** is shown here:

```
private void trainNetworkBackprop()
```

This method begins by creating a **Backpropagation** object to train the network. A learning rate of 0.001 and a momentum of 0.1 are used. These are sufficiently small to properly train this neural network.

```
final Train train = new Backpropagation(this.network, this.input,
      this.ideal, 0.001, 0.1);
```

A local variable named **epoch** is created to count the number of training epochs.

```
int epoch = 1;
```

The loop is entered, and then for each training epoch the **iteration** method is called on the **train** object.

```
do {
  train.iteration();
  System.out.println("Iteration #" + epoch + " Error:"
        + train.getError());
  epoch++;
} while ((epoch < 5000) && (train.getError() > 0.01));
}
```

This continues until either 5,000 epochs have passed, or the error rate is less than 1%.

Predicting the Future of the Sine Wave

To see how effective the neural network is at predicting the future of the sine wave, the **display** method should be called. The signature for the **display** method is shown here:

```
private void display() {
```

The numbers displayed from the sine wave will be rounded to two decimal places. We use the Java **NumberFormat** class to accomplish this.

```
final NumberFormat percentFormat = NumberFormat.
      getPercentInstance();
percentFormat.setMinimumFractionDigits(2);
```

Input and output arrays need to be created to hold the input to the neural network, as well as its output.

```
final double input[] = new double[SinWave.INPUT_SIZE];
final double output[] = new double[SinWave.OUTPUT_SIZE];
```

Next, we loop through all of the actual data. The neural network was not trained on all of the actual data, so some of this will be prediction.

```
for (int i = SinWave.INPUT_SIZE; i < SinWave.ACTUAL_SIZE; i++) {
```

We obtain the input and output data from the actual data array. The output data are the ideal values. We compare the actual output of the neural network to this data.

```
  this.actual.getInputData(i - SinWave.INPUT_SIZE, input);
  this.actual.getOutputData(i - SinWave.INPUT_SIZE, output);
```

A **StringBuilder** is created to hold the formatted data.

```
  final StringBuilder str = new StringBuilder();
  str.append(i);
  str.append(":Actual=");
  for (int j = 0; j < output.length; j++) {
      if (j > 0) {
        str.append(',');
      }
      str.append(output[j]);
  }
```

The neural network is called to compute its prediction.

```
  final double predict[] = this.network.computeOutputs(input);
```

The predicted value is displayed with the actual value.

```
  str.append(":Predicted=");
  for (int j = 0; j < output.length; j++) {
      if (j > 0) {
        str.append(',');
      }
      str.append(predict[j]);
  }
```

The difference is also displayed as a percentage.

```
  str.append(":Difference=");

  final ErrorCalculation error = new ErrorCalculation();
  error.updateError(predict, output);
  str.append(percentFormat.format(error.calculateRMS()));

  System.out.println(str.toString());
}
```

Finally, the line for this actual value is output. This neural network does a reasonably good job of predicting future values. There are many different ways that the input data can be presented for prediction. In the next chapter we will explore how to create neural networks that attempt to predict financial markets.

Chapter Summary

The feedforward neural network is very adept at recognizing patterns. Used properly, a feedforward neural network can be used to predict future patterns. Such neural networks are called predictive, or temporal neural networks. A predictive neural network is not one specific type of neural network; rather, it is any type of neural network used for prediction. This book uses feedforward neural networks for all prediction examples.

Implementing a feedforward neural network that predicts is simply a matter of properly constructing the input and output neurons. Time is divided into several blocks, called time slices. For example, a neural network may have five known time slices followed by an unknown time slice. This would produce a neural network with five input neurons and one output neuron. Such a neural network would be trained using known actual data in groups of six time slices. The first five time slices in each group would be the input neurons. The sixth would be the ideal output. To have the neural network predict, you would simply provide five known time slices to the input neurons. The output from the output neuron would be the neural network's prediction for the sixth time slice.

An entire book could easily be written on predictive neural networks. This chapter introduced predictive neural networks at a basic level by predicting the sine wave. Perhaps one of the most common applications of predictive neural networks is predicting movement in financial markets. The next chapter will provide an introduction to programming neural networks for financial market predictions.

Vocabulary

Actual Data

Predictive Neural Network

Sine Wave

Temporal Neural Network

Time Slice

Questions for Review

1. Can a self-organizing map be used for prediction? If so, how would you set up the input and output neurons?

2. When a feedforward neural network is used for prediction, what do the input neurons represent? What do the output neurons represent?

3. You have 500 time slices of information. How could you use this data to both train and test a predictive neural network.

4. How would you choose between using a hyperbolic tangent and a sigmoid function as the activation function for a predictive neural network?

5. Why might you use more than one output neuron in a predictive neural network?