Chapter 13

Time Series and Recurrent Networks

- Time Series
- Elman Networks
- Jordan Networks
- Deep Recurrent Networks

In this chapter, we will examine time series encoding and recurrent networks, two topics that are logical to put together because they are both methods for dealing with data that spans over time. Time series encoding deals with representing events that occur over time to a neural network. There are many different methods to encode data that occur over time to a neural network. This encoding is necessary because a feedforward neural network will always produce the same output vector for a given input vector. Recurrent neural networks do not require encoding of time series data because they are able to automatically handle data that occur over time.

The variation in temperature during the week is an example of time series data. For instance, if we know that today's temperature is 25 degrees, and tomorrow's temperature is 27 degrees, the recurrent neural networks and time series encoding provide another option to predict the correct temperature for

the week. Conversely, a traditional feedforward neural network will always respond with the same output for a given input. If a feedforward neural network is trained to predict tomorrow's temperature, it should respond 27 for 25. The fact that it will always output 27 when given 25 might be a hindrance to its predictions. Surely the temperature of 27 will not always follow 25. It would be better for the neural network to consider the temperatures for a series of days before the day being predicted. Perhaps the temperature over the last week might allow us to predict tomorrow's temperature. Therefore, recurrent neural networks and time series encoding represent two different approaches to the problem of representing data over time to a neural network.

So far the neural networks that we've examined have always had forward connections. The input layer always connects to the first hidden layer. Each hidden layer always connects to the next hidden layer. The final hidden layer always connects to the output layer. This manner to connect layers is the reason that these networks are called "feedforward." Recurrent neural networks are not so rigid, as backward connections are also allowed. A recurrent connection links a neuron in a layer to either a previous layer or the neuron itself. Most recurrent neural network architectures maintain state in the recurrent connections. Feedforward neural networks don't maintain any state. A recurrent neural network's state acts as a sort of short-term memory for the neural network. Consequently, a recurrent neural network will not always produce the same output for a given input.

13.1 Time Series Encoding

As we saw in previous chapters, neural networks are particularly good at recognizing patterns, which helps them predict future patterns in data. We refer to a neural network that predicts future patterns as a predictive, or temporal, neural network. These predictive neural networks can anticipate future events, such as stock market trends and sun spot cycles.

Many different kinds of neural networks can predict. In this section, the feedforward neural network will attempt to learn patterns in data so it can 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. Because the type of feedforward neural networks in this book always produce the same output for a given input, we need to make sure that we encode the input correctly.

A wide variety of methods can encode time series data for a neural network. The sliding window algorithm is one of the simplest and most popular encoding algorithms. However, more complex algorithms allow the following considerations:

- Weighting older values as less important than newer
- Smoothing/averaging over time
- Other domain-specific (e.g. finance) indicators

We will focus on the sliding window algorithm encoding method for time series. The sliding window algorithm works by dividing the data into two windows that represent the past and the future. You must specify the sizes of both windows. For example, if you want to predict future prices with the daily closing price of a stock, you must decide how far into the past and future that you wish to examine. You might want to predict the next two days using the last five closing prices. In this case, you would have a neural network with five input neurons and two output neurons.

13.1.1 Encoding Data for Input and Output Neurons

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. The following training set has a prediction window of size 1 and a past window size of 3:

```
[1,2,3] -> [4]
[2,3,4] -> [3]
[3,4,3] -> [2]
[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 continue. 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.

It is also possible to predict more than one value into the future. The following training set has a prediction window of size 2 and a past window size of 3:

```
[1,2,3] -> [4,3]
[2,3,4] -> [3,2]
[3,4,3] -> [2,1]
[4,3,2] -> [1,2]
```

The last two examples have only a single stream of data. It is possible to use multiple streams of data to predict. For example, you might predict the price with the price of a stock and its volume. Consider the following two streams:

```
Stream #1: 1, 2, 3, 4, 3, 2, 1, 2, 3, 4, 3, 2, 1
Stream #2: 10, 20, 30, 40, 30, 20, 10, 20, 30, 40, 30, 20, 10
```

You can predict stream #1 with stream #1 and #2. You simply need to add the stream #2 values next to the stream #1 values. A training set can perform this calculation. The following set shows a prediction window of size 1 and a past window size of 3:

```
[1,10,2,20,3,30] -> [4]
[2,20,3,30,4,40] -> [3]
[3,30,4,40,3,30] -> [2]
[4,40,3,30,2,20] -> [1]
```

This same technique works for any number of streams. In this case, stream #1 helps to predict itself. For example, you can use the stock prices of IBM and Apple to predict Microsoft. This technique uses three streams. The stream that we're predicting doesn't need to be among the streams providing the data to form the prediction.