Chapter 1

Neural Network Activation

- Summation
- Calculating Activation
- Activation Functions
- Bias Neurons

In this chapter you will see how to calculate the output for a feedforward neural network. Most neural networks are in some way based on the feedforward neural network. Seeing how this simple neural network is calculated will form the foundation for understanding training, and other more complex features of neural networks.

Several mathematical terms will be introduced in this chapter. You will be shown summation notation and simple mathematical formula notation. We will begin with a review of the summation operator.

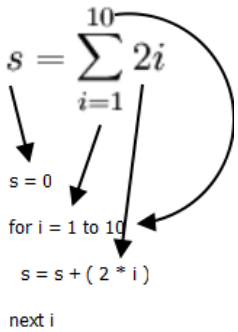
1.1 Understanding the Summation Operator

In this section, we will take a quick look at the summation operator. The summation operator, represented by the capital Greek letter sigma can be seen in Equation 1.1.

$$s = \sum_{i=1}^{10} 2i \tag{1.1}$$

The above equation is a summation. If you are unfamiliar with sigma notation, it is essentially the same thing as a programming **for** loop. Figure 1.1 shows Equation 1.1 reduced to pseudocode.

Figure 1.1: Summation Operator to Code

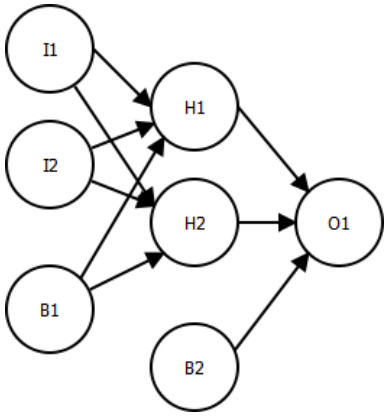


As you can see, the summation operator is very similar to a **for** loop. The information just below the sigma symbol species the stating value and the indexing variable. The information above the sigma species the limit of the loop. The information to the right of sigma specifies the value that is being summed.

1.2 Calculating a Neural Network

We will begin by looking at how a neural network calculates its output. You should already know the structure of a neural network from the book's introduction. Consider a neural network such as the one in Figure 1.2.

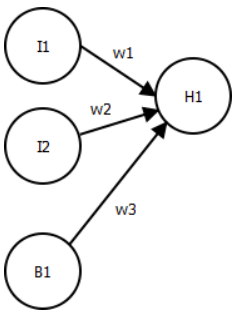
Figure 1.2: A Simple Neural Network



This neural network has one output neuron. As a result, it will have one output value. To calculate the value of this output neuron (**O1**), we must calculate the activation for each of the inputs into **O1**. The inputs that feed into **O1** are **H1**, **H2** and **B2**. The activation for **B2** is simply 1.0, because it is a bias neuron. However, **H1** and **H2** must be calculated independently. To calculate **H1** and **H2**, the activations of **I1**, **I2** and **B1** must be considered. Though **H1** and **H2** share the same inputs, they will not calculate to the same activation. This is because they have different weights. The weights are represented by lines in the above diagram.

First we must look at how one activation calculation is done. This same activation calculation can then be applied to the other activation calculations. We will examine how **H1** is calculated. Figure 1.3 shows only the inputs to **H1**.

Figure 1.3: Calculating H1Šs Activation



We will now see how to calculate H1. This relatively simple equation is shown in Equation 1.2.

$$h_1 = A(\sum_{c=1}^{n}(i_c * w_c)) \tag{1.2}$$

To understand Equation 1.2 we first examine the variables that go into it. For the above equation we have three input values, given by the variable **i**. The three input values are input values of **I1**, **I2** and **B1**. **I1** and **I2** are simply the input values that the neural network was provided to compute the output. **B1** is always 1, because it is the bias neuron.

There are also three weight values considered **w1**, **w2** and **w3**. These are the weighted connections between **H1** and the previous layer. Therefore, the variables to this equation are:

```
i[1] = first input value to the neural network
i[2] = second input value to neural network
i[3] = 1
w[1] = weight from I1 to H1
w[2] = weight from I2 to H1
w[3] = weight from B1 to H1
n = 3, the number of connections
```

Though the bias neuron is not really part of the input array, a one is always placed into the input array for the bias neuron. Treating the bias as a forward-only neuron makes the calculation much easier.

To understand Equation 1.2 we will consider it as pseudocode.

```
double w[3] // the weights
double i[3] // the input values
double sum = 0; // the sum
// perform the summation (sigma)
for c = 0 to 2
  sum = sum + ( w[c] * i[c] )
next
// apply the activation function
sum = A(sum)
```

Here we sum up each of the inputs times its respective weight. Finally this sum is passed to an activation function. Activation functions are a very important concept in neural network programming. In the next section we will examine activation functions.

1.3 Activation Functions

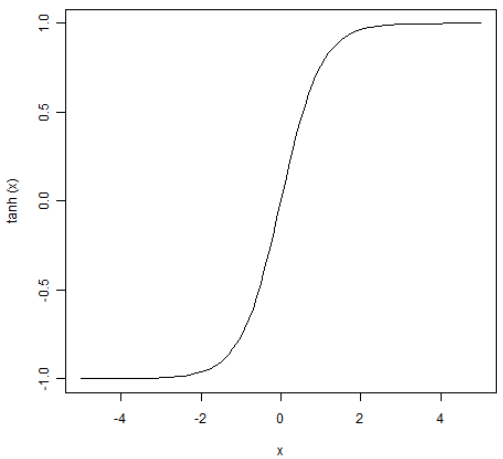
Activation functions are used very commonly in neural networks. Activation functions serve several important functions for a neural network. The primary reason to use an activation function is to introduce non-linearity to the neural network. Without this non-linearity a neural network could do little to learn non-linear functions. The output that we expect neural networks to learn is rarely linear.

The two most common activation functions are the sigmoid and hyperbolic tangent activation function. The hyperbolic tangent activation function is the more common of these two, as has a number range from -1 to 1, compared to the sigmoid function which is only from 0 to 1.

$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{1.3}$$

The hyperbolic tangent function is actually a trigonometric function. However, our use for it has nothing to do with trigonomy. This function was chosen for the shape of its graph. You can see a graph of the hyperbolic tangent function in Figure 1.4.

Figure 1.4: The Hyperbolic Tangent Function



Notice the range is from -1 to 1? This gives it a much wider range of numbers it can

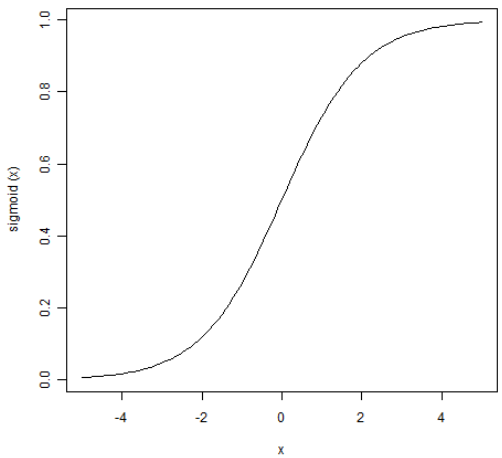
accept. Also notice how values beyond -1 to 1 are quickly scaled? This provides a consistent range of numbers for the network.

Now we will look at the sigmoid function. You can see this in Equation 1.4.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1.4}$$

The sigmoid function is also called the logistic function. Typically it does not perform as well as the hyperbolic tangent function. However, if you have all positive values in the training data, it can perform well. The graph for the sigmoid function is shown in Figure 1.5.

Figure 1.5: The Sigmoid Function

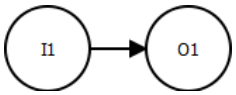


As you can see it scales numbers to 1.0. It also has a range that only includes positive numbers. It is less general purpose than hyperbolic tangent, but it can be useful. For example, the XOR operator, as discussed in the introduction, only has positive numbers. The sigmoid function outperforms the hyperbolic tangent function.

1.4 Bias Neurons

You may be wondering why bias values are even needed? Bias values allow a neural network to output a value of zero even when the input is near one. Adding a bias allows the output of the activation function to be shifted to the left or right on the x-axis. To see this, consider a simple neural network where a single input neuron **I1** is directly connected to an output neuron **O1**. The network shown in Figure 1.6 has no bias.

Figure 1.6: A Bias-less Connection



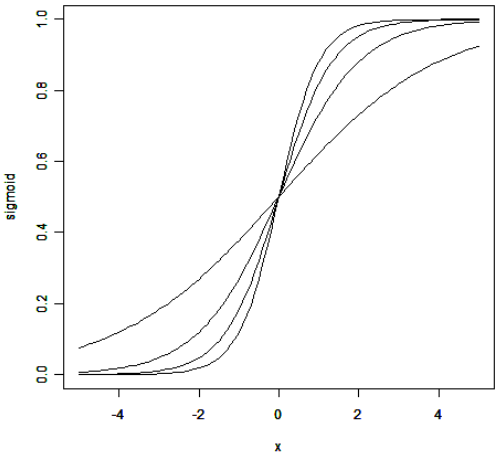
This network's output is computed by multiplying the input (**x**) by the weight (**w**). The result is then passing an Activation Function. In this case, we are using the Sigmoid Activation Function.

Consider the output of the sigmoid function for the following four weights.

```
sigmoid(0.5*x)
sigmoid(1.0*x)
sigmoid(1.5*x)
sigmoid(2.0*x)
```

Given the above weights, the output of the sigmoid will be as seen in Figure 1.7.

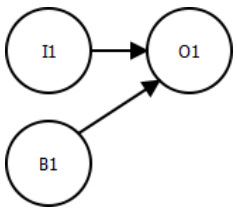
Figure 1.7: Adjusting Weights



Changing the weight w alters the "steepness" of the sigmoid function. This allows the neural network to learn patterns. However what if wanted the network to output 0 when x is a value other than 0, such as 3? Only changing the steepness of the sigmoid will not accomplish this. You must be able to shift the entire curve to the right.

That is the purpose of bias. Adding a bias neuron causes the neural network to appear as Figure 1.8.

Figure 1.8: A Biased Connection



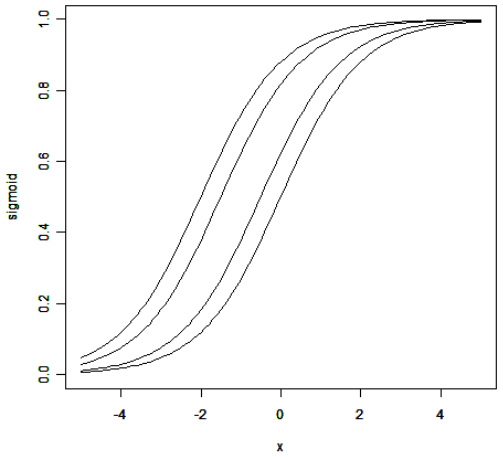
Now we calculate with the bias neuron present. We will calculate for several bias weights.

```
sigmoid(1*x + 1*1)
sigmoid(1*x + 0.5*1)
sigmoid(1*x + 1.5*1)
```

```
sigmoid(1*x + 2*1)
```

This produces the following plot, seen in Figure 1.9.

Figure 1.9: Adjusting Bias



As you can see, the entire curve now shifts.

1.5 Chapter Summary

This chapter demonstrated how a feedforward neural network calculates output. The output of a neural network is determined by calculating each successive layer, after the input layer. The final output of the neural network eventually reaches the output layer.

Neural networks make use of activation functions. An activation function provides non-linearity to the neural network. Because most of the data that a neural network seeks to learn is non-linear the activation functions must be non-linear. An activation function is applied after the weights and activations have been multiplied.

Most neural networks have bias neurons. Bias is an important concept for neural networks. Bias neurons are added to every non-output layer of the neural network. Bias neurons are different than ordinary neurons in two very important ways. First, the output from a

bias neuron is always one. Second, a bias neuron has no inbound connections. The constant value of one allows the a layer to respond with non-zero values even when the input to the layer is zero. This can be very important for certain data sets.

The neural networks will output values determined by the weights of the connections. These weights are usually set to random initial values. Training is the process where these random weights are adjusted to produce meaningful results. We need a way for the neural network to measure the effectiveness of the neural network. This measure is called error calculation. Error calculation is discussed in the next chapter.