The concept of neural networks date to 1943 when Warren McCulloch and Walter Pitts provided the mathematical foundation for them. However they didn’t become popular until the 1980s when computers became powerful enough to iterate through the MANY learning cycles needed to teach the networks. In the 1990’s they were overtaken by simpler methods such as support vector machines until the 2000’s when there was a renewed interest following the advent of deep learning and biorobotics. The work shown here is going to focus on application rather than theory. Another detailed tutorial is shown [here](http://takinginitiative.wordpress.com/2008/04/03/basic-neural-network-tutorial-theory/) that uses C++.

The concept is simple to understand, there are however many inputs to the system (called features), these are weighted, summed, and run through a switching function. This process is repeated however many times it is deemed necessary (usually twice for a three layer network). The output is taken from the last layer of the network (the output layer). A diagram is shown in Figure One.

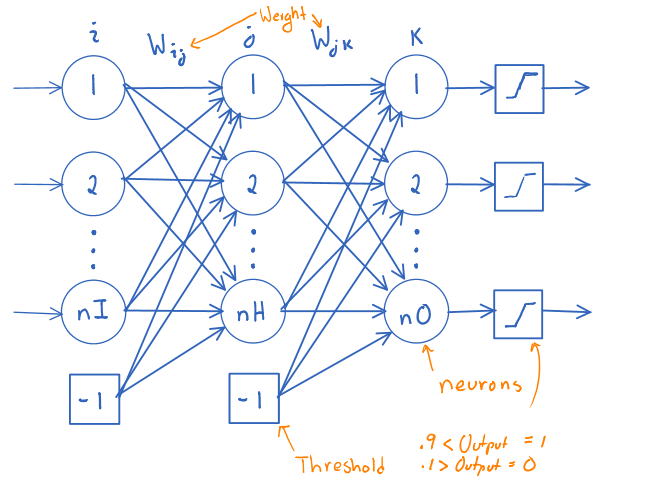


Figure 1: A Three Layer Neural Network.

A neural network (NN) is made up of ‘neuron’, there are *nI* input neurons, *nH* hidden neurons, and *nO* output neurons. The number of input neurons are equal to the number of inputs to the system. The number of output neurons is equal to the number of outputs, and the number of hidden neurons is based on whatever provides the best accuracy.

The value of a neuron is calculated in the following equation. Here is the resulting value of the neuron *j* in the current layer, *N* is the number of neurons in the previous layer, is the weight that is given to the neuron *i* of the previous layer when used to calculate the value of neuron *j* in the current layer, is the value of the neuron *i* of the previous layer and is a threshold value.

The threshold value changes during the learning process. It is created by defining a neuron for the first two layers that has a value of negative 1, and has weights for each neuron in the next layer (except the threshold neuron).

The function *f()* is called the switching function, typically the sigmoid function is used, it is defined in the next equation and is graphed in Figure Two.

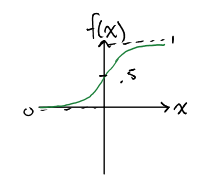


Figure 2: The Sigmoid Function

There are three phases to the neural network.

1. Training – Used to make the network adapt to the system
2. Testing – Used to test the accuracy of the network
3. Validating – Used to show how good your network is

The first thing to do is get some test data, I recommend the [UCI Machine Learning Repository](http://archive.ics.uci.edu/ml/index.html), I used it during my biorobotics class. I used the letter recognition data set, it has a lot of data that is useful in training and testing. The data needs to be split (randomly) into three data sets, usually 50% training, 25% testing, and 25% validation. In MATLAB there is a command called [*divideint()*](http://www.mathworks.com/help/nnet/ref/divideint.html) that will do this for you.

Once the data has been divided up the training data is fed through the NN and compared against the known outputs. The letter ‘*i*’ will be used to refer to the neurons of the input layer, the letter ‘*j*’ will be used to refer to the neurons in the hidden layer, the letter ‘*k*’ will be used to refer to the neurons in the output layer. The feedforward process goes as follows.

1. Assign the inputs to the input neurons, the assignment must be consistent. For example input 1 must always be assigned to input neuron 1.
2. The hidden neurons are calculated using the following equation. Remember *T* is negative one with weights for each neuron, and *f()* is the sigmoid activation function.
3. The output neurons are calculated using the following equation.

Now that the outputs have been calculated the weights for the NN must be modified to reduce the error. The adaptation law for the hidden layer to output layer weights is shown in the next equation.

Where

Here α is the learning rate and is between 0 and 1, is the value at hidden neuron *j* and is defined as follows.

Here is the value at output neuron *k*, and is the desired output of output neuron *k*.

The adaptation law for the input to hidden layer weights are more complicated. The first two equations are similar.

Where

The parameters mean the same thing as before, however is different.

So now that the adaptation law is defined the backpropagation technique is defined in the following steps.

1. Feed first set of inputs through and calculate outputs.
2. Calculate change in weights.
3. Repeat for all training data.
4. Repeat until desired amount of training cycles or until desired accuracy is found.

It is important you don’t use the validation data to check accuracy, wait until you have your desired performance during training and testing then check the system on the validation accuracy.

There is another technique called momentum. The idea is that you ass a scaled version of the previous weight to the new weight. The change in the system of equations is shown below.

Where β is the momentum constant and is between 0 and 1.

I wrote a MATLAB script that implements this in order to identify the letter ‘a’ and when a vowel is shown. I was able to identify the letter ‘a’ with an accuracy of 96% and identify a vowel with an 80% accuracy during validation. You can download my files here.