Author: Ken Gorro, MSIT

In this tutorial, I will discuss the basics of neural networks by creating a consonant and vowel recognition using the perceptron model.

Neural network refers to the system of interconnected neurons that mimics the human brain.

Figure 1 is a basic example of a neural network:

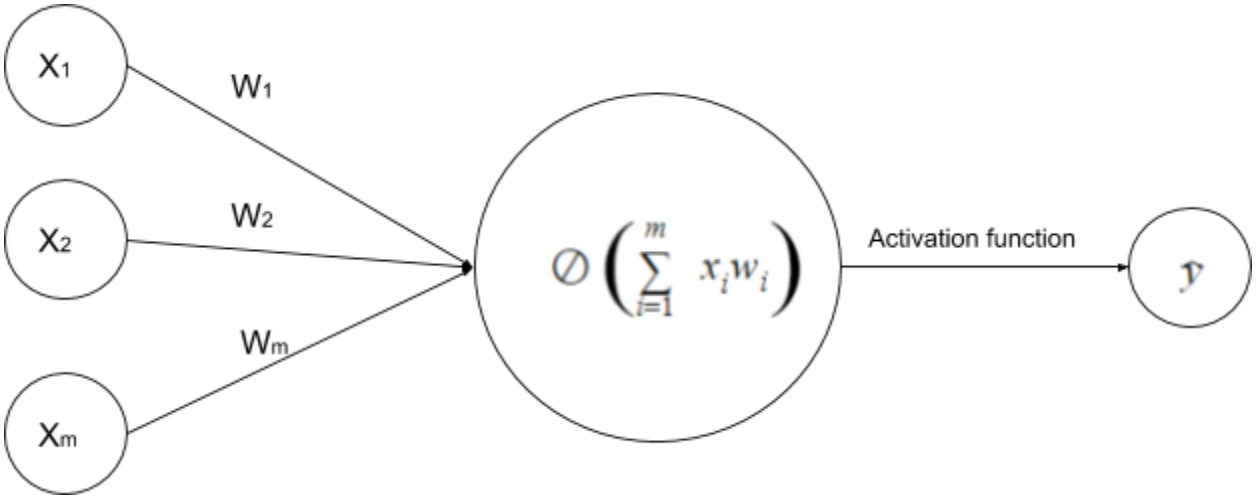

Figure 1. Single Layer Feed Forward Neural Network

Figure 1 consist the following parts:

1.) Input layer

2.) Weights

3.) Weighted Sum:

$$\varnothing\left(\sum_{i=1}^{m} x_i w_i\right)$$

4.) Activation function

5.) Output value ($\widehat{y}$)

Input layer contains your inputs which are the independent variables to predict your dependent variable.

Weights represent the connection between neurons. Weights determine the importance of the input layer to the hidden layer.

Activation function translates the weighted sum into the output layer. There are 4 to 5 activation functions that are commonly used but for the perceptron model we will use the threshold function.

Weighted sum is the sum of the product of inputs and weights.

Figure 1 neural network model is commonly called Perceptron. In this tutorial we are going to focus with the Perceptron model to understand the basics of neural networks. To best understand the Perceptron model we will create a classifier if the given alphabet is **consonant or a vowel.** We will modify the model illustrated in Figure 1 by adding a bias to the weighted sum.

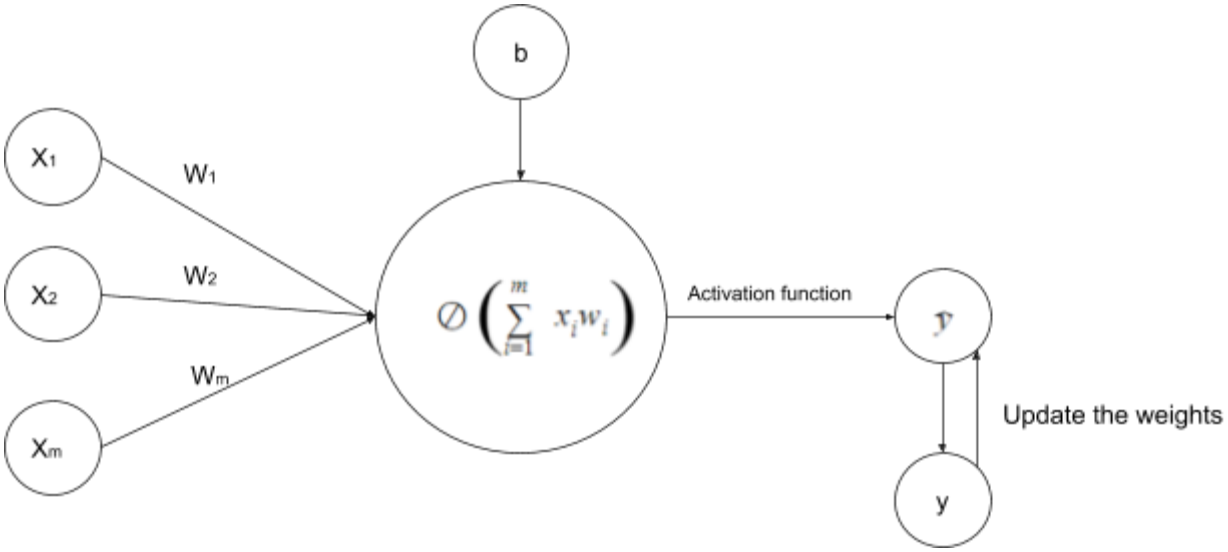

Figure 2. Perceptron model with Bias

Bias in neural network acts as similar to the constant in your linear equation that helps the activation function to fit the output better with the data.

The expected output:

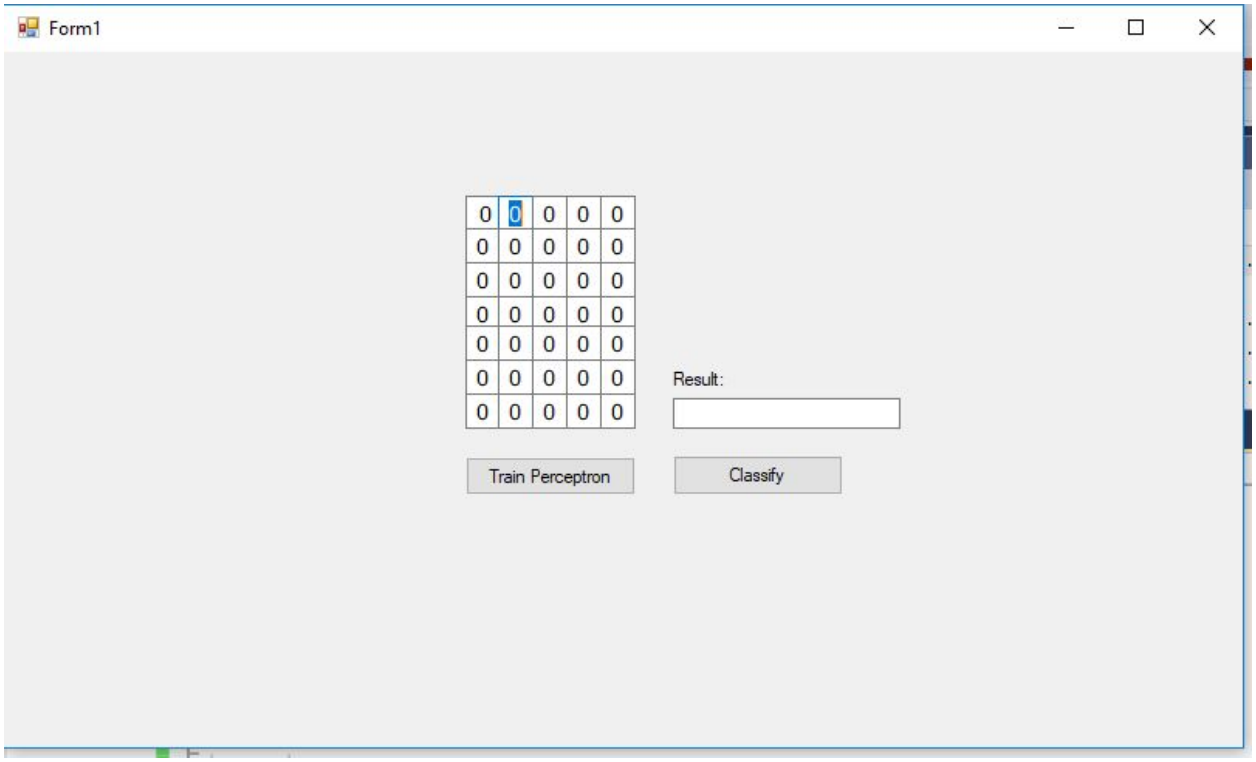

Figure 3. Sample output

Perceptron is a single layer neural network model which belongs to the category of linear classifier.

Input layer: Each letter in the english alphabet is represented in a 7x5 matrix of 1 and 0. 1 denotes black pixel while 0 denotes white pixel.

Weights: Transform each input data to the hidden layers.For this tutorial we will have 35 weights plus 1 bias.

Weighted sum: It is the sum of Weight x Input.

Activation function: The best activation to be used is the threshold function. Below would be our activation function.

$$1 \text{ if } x >= 0$$
$$0 \text{ if } x <= 0$$

Figure 4. Threshold function

Cost function (Output value): $C = \sum \frac{1}{2} (\widehat{y} - y)$

Pseudo code:

a.) Initialize inputs

b.) Initialize weights plus the bias to random numbers.

c.) Calculate for the weighted sum with the bias

d.) Apply the weighted sum to the threshold function.

e.) Calculate cost function

f.) Update the weights

Figure 5. Pseudo code

a.) Initialize inputs: An array of 0 and 1 representing a letter in an alphabet.

b.) Initialize weights: Create a list of integers with 35 + 1 integers and set them to any random numbers.

```
public void initializeWeights()
{
   /*Initialize the weight: +1 with bias*/
   for(int i = 0; i < MAXINPUT; i++)
   {
      weights.Add(Math.Round(randNum.NextDouble(),2));
   }
}
```

c.) Calculate the weighted sum with bias.

```
for (int j = 0; j < MAXINPUT; j++)
{
   if (j < (MAXINPUT - 1))
      weightedSum+= Math.Round(trainingLetterPattern[i][j] * weights[j], 5);
   else
      weightedSum+= weights[j];
}
```

weightedSum variable in this case represents the weighted sum. MAXINPUT is representing the number of weights plus the bias (36 weights in total).

d.) Apply the weighted sum to the cost function:

```
/*evaluate the vValue*/
 yValue = (weightedSum <= 0) ? 0 : 1;
```

e.) Calculate the cost function using delta:

```
/*Calculate the delta value*/
deltaValue = outputLayer[i] - yValue;

/*calculate the error*/
totalError += Math.Abs(deltaValue);
```

f.) Update the weights using the cost function:

```
for (int j = 0; j < MAXINPUT; j++)
{
    if (j < (MAXINPUT- 1))
        weights[j] = Math.Round(weights[j] + (learningRate * deltaValue * trainingLetterPattern[i][j]), 2);
    else
        weights[j] = Math.Round(weights[j] + (learningRate * deltaValue), 2);
}
```

In the code above, I intentionally add the learning rate. Learning rate is a hyper parameter that controls how much to change the model in response to the estimated error each the model weights are updated.

g.) If the cost function is equal to zero stop the learning process else repeat step C to F.

Note: The example project is written in C#.NET. The training data for the letters are drawn in capital letter format.

Link to Github Repo: https://github.com/kgorro/NeuralNetworkModels