Building Logic Gates with Feed Forward Neural Networks

Introduction

The aim of this experiment was to implement simple feed forward networks that could act as logic gates, capable of correctly classifying two inputs.

Method

Input and corresponding outputs of 5 logic gates (AND, OR, XOR, NOT and NAND) were stored in matrices such that the inputs for a given output were in the same column. For example

```
AND_In  := 0 1 0 1
           0 0 1 1

AND_Out := 0 0 0 1
```

Then for each network a set of randomly generated weights (one for each input), between 0 and 1 were created. Then by vector multiplication the weights were multiplied by the inputs to give a single scalar output, to which a linear threshold of 0.5 was applied.

```
if output >= 0.5
    output = 1
else
    output = 0
```

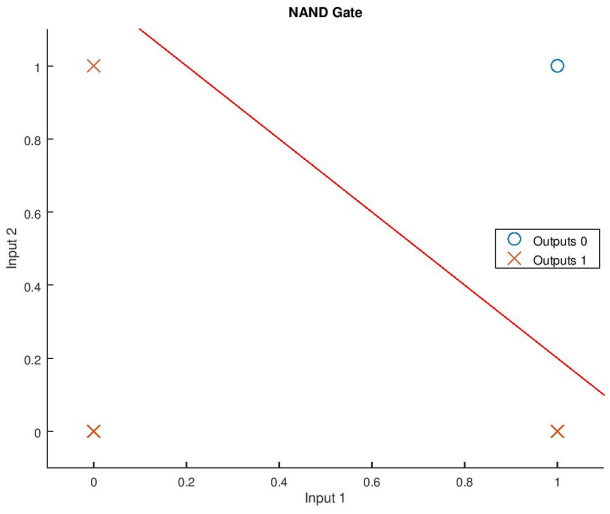
Finally the sum of the squared errors was found by squaring the subtraction of the output from the expected correct result for the given logic gate. If the sum of the errors was found not to be 0 then the algorithm would loop again until the sum of the squared errors was 0.

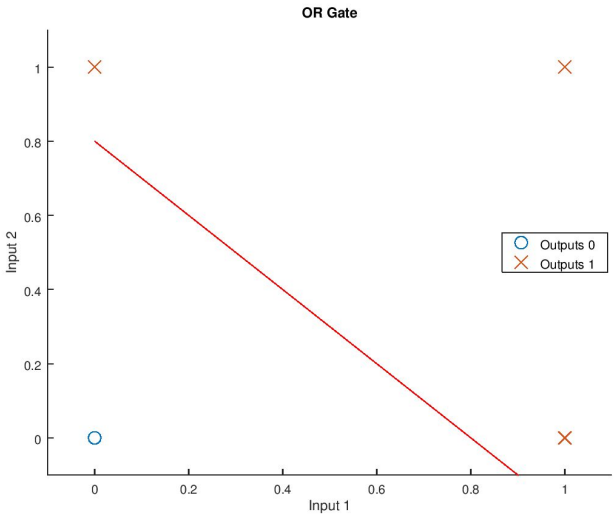
A function was created that would take the input and output of a given and logic gate and output the correct given w for this operation called **NetworkWeights** that would loop a maximum of 1000 times. at which point it returned the correct weights.

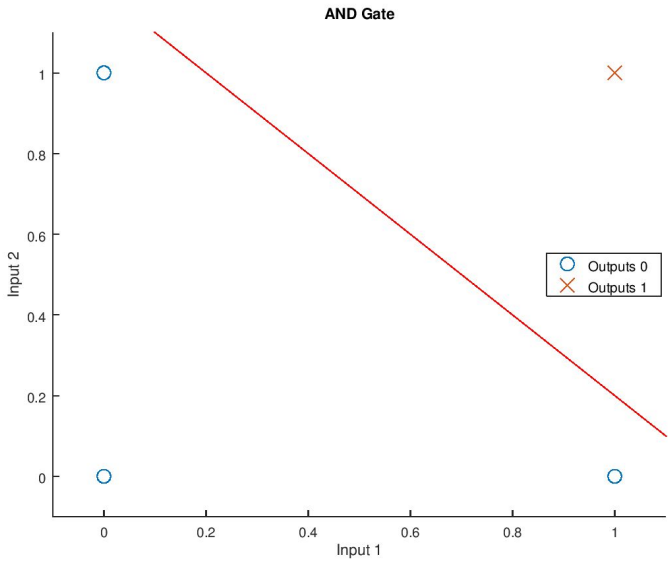
When programming the XOR Gate, an extra hidden layer of nodes and weights needed to be added. As as single layer perceptron was unable to solve the problem alone. Firstly the XOR gate was programmed by hand by creating an OR Gate and an NAND gate that fed into an AND gate.

When adding the XOR gate to the learning function it needed to account the extra layers. So after failing to build a single layer perceptron after 1000 attempts, a 3*3 matrix of random weights was generated and up to 10000 attempts are made to find a suitable neural network.

Results







Discussion

The simplest of networks were found to be the AND, NAND and OR gates. This is because the classification of the outputs was linearly separable As can be seen in the graphed results.

When implementing the random generator it was found that the threshold - or what was instead used, a bias node - existed within a range. The weights connecting each node were proportional to each other, so when implementing the XOR gate by hand it can be seen that the weights multiplying each input range between the number -30 and 30.

Further what was found was that when implementing a learning algorithm that a bias node needed to be used. This was important because in order to move the decision boundary to correctly classify the outputs of the perceptron the threshold needed to be changed. So by using an input bias and setting the threshold to 0 the learning algorithm could generate a weight to move the threshold.