# Multilayer Neural Networks and Back Propagation : Project - 2

Atsushi oba (s1230128), Hitoshi Ikuta (m5221106)
Chowdhury Md Intisar (m5212106), Yunosuke Teshima (m5221151)

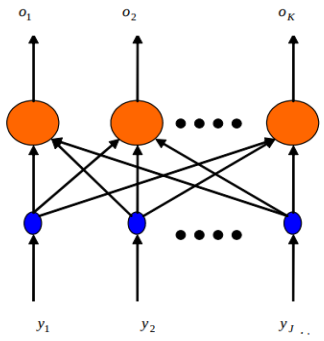
April 24, 2018

## 1 Back Propagation Algorithm

In this project, we have implemented the Back propagation learning algorithm to solved the 4-bit parity problem. The basic property of back propagation is, it propagates the error from its output layer to the subsequent hidden layer. The weight of the network is updated by gradient descent method.

Below is a figure of 3 layer neural network

Figure 1: A 3-layer Neural Network




## 2 Terms and Terminology

- $z_i$ is the i-th input
- $y_j$ is the output of the j-th neuron
- $o_k$ is the weight from the i-th input to the j-th hidden neuron
- $v_{ji}$ is the weight from the i-th input to the j-th hidden neuron

- $w_{kj}$ is the weight from the j-th hidden neuron to the k-th output neuron

# 3 Rule for Updating weights

The rule for updating the weight is same as the previous project-1 for output layer. However, the rule for hidden layer weight update has few changes as follows. For the hidden layer

$$v_{ji} = v_{ji}^{old} + \eta \delta_{yj} z_i \tag{1}$$

Thus we require to calcualte the deltas of the neuron. Following two equation was used to calculate the delta for output and hidden layer respectively.

$$\delta_{ok} = (d_k - o_k)(1 - o_k^2)/2 \tag{2}$$

$$\delta_{yj} = \Big(\sum_{k=1}^{K} \delta_{ok} W_{kj}\Big)(1 - y_j^2)/2 \tag{3}$$

The above two equations are derived from the error function with the method of chain rule.

# 4 Input and Output Generations

First, we have written an algorithm to generated all possible 4 bit configurations for parity. Next, we have generated output based on the even and odd occurrence of 1 in the 4 bit parity as follows.

Figure 2: Input generate by recursive function

```
37  void generateInput( int val, int coun) {
38
39      if ( coun <= 1 ) {
40          int i;
41          for(i = 3; i >= 0; i--) {
42              x[ind][i] = (val >> i) & 1;
43          }
44          ind++;
45      } else {
46          generateInput(val*2, coun-1);
47          generateInput(val*2+1, coun-1);
48      }
49  }
```

Figure 3: Generate Output

```
51  void generateOutput(double (&x)[16][5]) {
52
53      double tot = 0.0;
54      for( int i = 0; i < 16; i++) {
55          tot = 0.0;
56          for( int j = 0; j < 4; j++)
57              tot += (int)x[i][j];
58          d[0][i] = !((int)tot&1 ) ? 1 : 0;
59      }
60  }
```

Figure 4: Output of the program

```
Input = {0,0,0,0} Output = 1
Input = {1,0,0,0} Output = 0
Input = {0,1,0,0} Output = 0
Input = {1,1,0,0} Output = 1
Input = {0,0,1,0} Output = 0
Input = {1,0,1,0} Output = 1
Input = {0,1,1,0} Output = 1
Input = {1,1,1,0} Output = 0
Input = {0,0,0,1} Output = 0
Input = {1,0,0,1} Output = 1
Input = {0,1,0,1} Output = 1
Input = {1,1,0,1} Output = 0
Input = {0,0,1,1} Output = 1
Input = {1,0,1,1} Output = 0
Input = {0,1,1,1} Output = 0
Input = {1,1,1,1} Output = 1
```

# 5 Results

The network has been tested with different number of neuron ranging from 4 to 10. Each run with different number of neuron gave different result. The result has been depicted below.

As we can see from Table 1, with increase of hidden neuron the number of epoch also decreases. But, sometimes use of too many hidden neuron can result in over-fitting. One key point of observation is, the epoch for 6,7,8 is nearly same.

Thus, we can conclude that

- With the increase of neuron in the hidden layer complex function can be constructed.
- Hidden layer is necessary if we want to solve a non-linear problem.
- As the number of neuron in hidden layer was increased it took less epochs to converge to solution.

Table 1: No. of Neuron and Epochs

| **Neurons** | **Epochs** |
|---|---|
| 4 | 52496 |
| 5 | 35741 |
| 6 | 9676 |
| 7 | 9757 |
| 8 | 9603 |
| 9 | 10769 |
| 10 | 8999 |

- Care should be taken in case, increase number of neuron can cause over fitting problem.
- Too many neuron in hidden layer / or too many hidden layer is computationally expensive. Thus, a trade off is to be made in choosing number of neuron and compromising performance.