|  |
| --- |
| 2802ICT |
| Intelligent Systems |
| Assignment 2 |

|  |
| --- |
| Sam Grumley  S5048240 |

# Part 1

The forward pass also known as forward propagate, provides the weighted sum of the input into the next node. With this method we can move forward through the network and get an output based of the current weights. Moving through each node requires getting the net value which is the weighted sum going into the node and then we can normalise the result using the sigmoid function to keep the value between -1 and 1. This must be done for both inputs X1 and X2.

Once the outputs have been calculated via forward pass you can then use the squared error function to check if the output was correct or the difference between.

The term training the network refers to finding the best cost function. The best cost function is the one with the least errors. To do this we need to use gradient decent to make sure that our cost function is following the data to the lowest point on the error axis. To maximise the efficiency of training of the data, a variant of the gradient descent algorithm, “gradient decent mini batch”. In this case we will determine a batch size which refers to how many inputs to calculate before updating the weights. To add this to the backpropagation part of the training, we calculate the derivative of the total error with respect to the weight we are adjusting for each input in the batch. We can then multiply the sum of the batch with the learning rate and divide by the number of inputs in the minibatch. This then gives us the change in weight we need to update the weight value. Simply subtract the change in the weight from the current weight and that will be the new value.

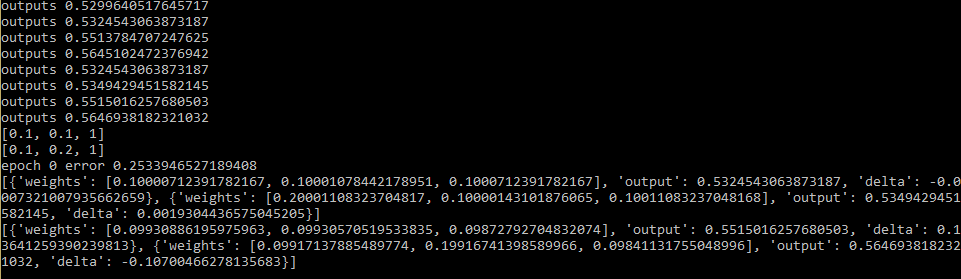


This is the general equation for updating the weights and below is the fleshed out version



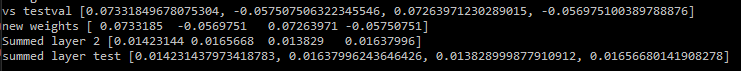
By using the minibatch algorithm we can increase the frequency of updating the weights which helps us avoid hitting local maximums and using less computational memory.

Below is the procedure to find the error total with respect to the hidden layer one. The same will apply elsewhere but will swap variables according to the diagram. Each colour represents the equations that make up the other equations



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | X1 | X2 | Program | |
| W1 | 0.0999961705361 | | 0.09999287608226642 | |
| W2 | 0.1999988916763 | | 0.19998921557833008 | |
| W3 | 0.09999896133595 | | 0.0999889167630617 | |
| W4 | 0.09999985690495 | | 0.0999985689813837 | |
| W5 | 0.099308861975 | | 0.09930886197388886 | |
| W6 | 0.0991713789 | | 0.09917137885954593 | |
| W7 | 0.0993057521 | | 0.09930570520953383 | |
| W8 | 0.19916741399 | | 0.19916741399056967 | |
| W9 | 0.099961705358 | | 0.09992876082266425 | |
| W10 | 0.099889103235 | | 0.09988916763061688 | |
| W11 | 0.09872792708 | | 0.09872792707492782 | |
| W12 | 0.0984113175 | | 0.09841131755925499 | |
|  | 0.5513784697 | 0.5515016246 | 0.05513784707 | 0.5515016257 |
|  | 0.5645102464 | 0.5646938182 | 0.5646938182 | 0.564510247237 |
| Error Total | 0.253394 | | 0.255339465 | |

Change in accuracy



## Forward Pass

### X1



= 0.1 \* 0.1 + 0.1 \* 0.1 + 0.1 \* 1

= 0.12



= 1/ 1 + e^(-0.12)

=1 / 1.8869

=0.5299



= 0.2 \* 0.1 + 0.1 \* 0.1 + 0.1 \* 1

= 0.13



= 1/ 1 + e^(-0.13)

=1 / 1.8780

=0.5324



= 0.1 \* 0.5299 + 0.1 \* 0.5324 + 0.1 \* 1

= 0.2062



= 1/ 1 + e^(-0.2062)

=1 / 1.8136

=0.5513



= 0.1 \* 0.5299 + 0.2 \* 0.5324 + 0.1 \* 1

= 0.2594



= 1/ 1 + e^(-0.2594)

=1 / 1.7715

=0.5645

### X2



= 0.1 \* 0.1 + 0.1 \* 0.2 + 0.1 \* 1

= 0.13



= 1/ 1 + e^(-0.13)

=1 / 1.8780

=0.5325



= 0.2 \* 0.1 + 0.1 \* 0.1 + 0.1 \* 1

= 0.14



= 1/ 1 + e^(-0.14)

=1 / 1.8693

=0.5350



= 0.1 \* 0. 5325 + 0.1 \* 0. 5350 + 0.1 \* 1

= 0.2067



= 1/ 1 + e^(-0.2067)

=1 / 1.8132

=0.5515



= 0.1 \* 0.5325 + 0.2 \* 0.5350 + 0.1 \* 1

= 0.2603



= 1/ 1 + e^(-0.2603)

=1 / 1.7708

=0.5647

## Calculating the error

### X1

Squared error function



= 0.5 \* (1 – 0.5513)^2

=0.1006



= 0.5 \* (0 – 0.5645)^2

= 0.1593



= 0.1006 + 0.1593

=0.2599

### X2

Squared error function



= 0.5 \* (0 – 0.5515)^2

=0.1521



= 0.5 \* (1 – 0.5647)^2

= 0.0947



= 0.1521 + 0.0947

=0.2468

## Backward Propagation

### W5

### W6

0.07354751

### W7

### W8:

0.19916741399

### W11

### W12

0.0984113175

### W1

### X1

= 0.1

0.00006926827

### X2

0.01364125936447827

= 0.1

0.1

### W2

### X1

= 0.1

### X2

= 0.1

### W3

### X1

= 0.1

0.00000692827

### X2

= 0.2

### W4

### X1

= 0.1

### X2

= 0.2

### W10

### X1

### X2

### W9

### X1

0.000069268276

### X2

0.01364125936447827

0.000038294642

0.099961705358

## Part 2

Train the Neural network with setting

Epoch = 30

Minibatch = 20

N=3.0

Plot accuracy vs epoch

Train n = 0..001, 0.1, 1.0, 10, 100

Train m = 1, 5, 10, 20, 100

Try different parameters and determine the best

Run part 1 implementation for 3 iterations

## Part 3

Substitute in the cross entropy function and run the same results as tested above