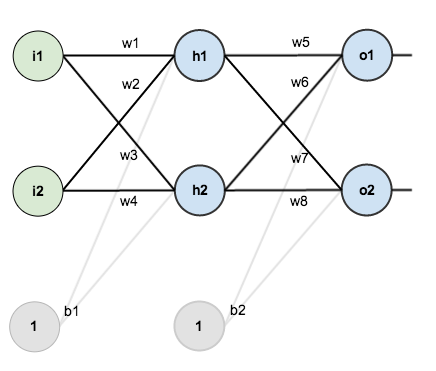
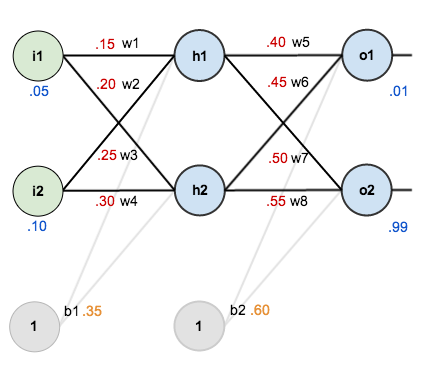
**Neural Network**



We Initialize the weights and biases using the random function given by Python and take our Input and Output from the training file to train the model.

The goal of backpropagation is to optimize the weights so that the neural network can learn how to correctly map arbitrary inputs to outputs.

**Example :-**



**The Forward Pass:-**

Here’s how we calculate the total net input for h\_1:

net\_{h1} = w\_1 \* i\_1 + w\_2 \* i\_2 + b\_1 \* 1

net\_{h1} = 0.15 \* 0.05 + 0.2 \* 0.1 + 0.35 \* 1 = 0.3775

We then squash it using the logistic function to get the output of h\_1:

out\_{h1} = \frac{1}{1+e^{-net\_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992

Carrying out the same process for h\_2 we get:

out\_{h2} = 0.596884378

We repeat this process for the output layer neurons, using the output from the hidden layer neurons as inputs.

Here’s the output for o\_1:

net\_{o1} = w\_5 \* out\_{h1} + w\_6 \* out\_{h2} + b\_2 \* 1

net\_{o1} = 0.4 \* 0.593269992 + 0.45 \* 0.596884378 + 0.6 \* 1 = 1.105905967

out\_{o1} = \frac{1}{1+e^{-net\_{h1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507

And carrying out the same process for o\_2 we get:

out\_{o2} = 0.772928465

**The Backwards Pass**

By applying chain rule and updation the weights based on the error calculation we get the optimized weights. By repeating the step multiple times we would have updated all our weights.

**Code Overview**

Shorthand:

"pd\_" as a variable prefix means "partial derivative"

"d\_" as a variable prefix means "derivative"

"\_wrt\_" is shorthand for "with respect to"

"w\_ho" and "w\_ih" are the index of weights from hidden to output layer neurons and input to hidden layer neurons respectively

1. We first Initialize the neural Network with random weights and biases.
2. To train the network we feed it with the training data provided.
3. During the training we constantly update the weights based on the error generated.
4. We repeat the training process for 100 times.
5. We apply the trained network on our Test set to get the output.
6. The output is compared with the actual result to produce the accuracy and confusion matrix

**Result**

For KNN, the values we get are

For k = 1, Accuracy – 65% Time – 29 minutes

For k =3, Accuracy – 67% Time – 29 minutes

For k =5, Accuracy – 69% Time – 29 minutes

The accuracy increases as the nodes in the hidden layer increases. Below is a brief summary of the same:-

|  |  |  |  |
| --- | --- | --- | --- |
| Hidden Nodes | Number of Iterations | Accuracy | Time Taken |
| 2 | 100 | 36% | 84 |
| 5 | 100 | 38% | 132 |
| 10 | 100 | 40% | 184 |
| 15 | 100 | 40% | 234 |
| 10 | 1000 | 52% | 12578 |

The accuracy of the system increase significantly when the number of iteration done to train the network increase but it comes at the cost of time.

For the sake of time management have changed the iterations to 100 but can be easily modified to 1000 or 10000 to gain grater accuracy.