Feed Forward / Forward Pass

Jithin

Algorithm to train a MLP

1. We take input and output

2. We initialize weights and biases with random values
   This is one time initiation. In the next iteration, we will use updated weights, and biases.

3. We take matrix dot product of input and weights assigned to edges between the input and hidden layer then add biases of the hidden layer neurons to respective inputs, this is known as **linear transformation.**

hidden_layer_input= matrix_dot_product(X,wh) + bh

Algorithm to train a MLP

4. Perform non-linear transformation using an activation function (Sigmoid). Sigmoid will return the output as 1/(1 + exp(-x))

```
hiddenlayer_activations = sigmoid(hidden_layer_input)
```

5. Perform a linear transformation on hidden layer activation (take matrix dot product with weights and add a bias of the output layer neuron) then apply an activation function (again used sigmoid, but you can use any other activation function depending upon your task) to predict the output

```
output_layer_input = matrix_dot_product (hiddenlayer_activations * wout ) + bout
output = sigmoid(output_layer_input)
```

Forward Propagation Section

Algorithm to train an MLP

6. Compare prediction with actual output and calculate the gradient of error (Actual – Predicted). Error is the mean square loss

```
E = y – output
```

7. Compute the slope/ gradient of hidden and output layer neurons

*(To compute the slope, we calculate the derivatives of non-linear activations x at each layer for each neuron). Gradient of sigmoid can be returned as x * (1 – x).*

```
slope_output_layer = derivatives_sigmoid(output)
slope_hidden_layer = derivatives_sigmoid(hiddenlayer_activations)
```

8. Compute change factor(delta) at output layer, dependent on the gradient of error multiplied by the slope of output layer activation

```
d_output = E * slope_output_layer
```

Algorithm to train an MLP

9. At this step, the error will propagate back into the network which means error at hidden layer. *For this, we will take the dot product of output layer delta with weight parameters of edges between the hidden and output layer (wout.T).*

```
Error_at_hidden_layer = matrix_dot_product(d_output, wout.Transpose)
```

10. Compute change factor(delta) at hidden layer, multiply the error at hidden layer with slope of hidden layer activation

```
d_hiddenlayer = Error_at_hidden_layer * slope_hidden_layer
```

11. Update weights at the output and hidden layer. The weights in the network can be updated from the errors calculated for training example.

```
wout = wout + matrix_dot_product(hiddenlayer_activations.Transpose, d_output)*learning_rate
wh =  wh + matrix_dot_product(X.Transpose,d_hiddenlayer)*learning_rate
```

Algorithm to train an MLP

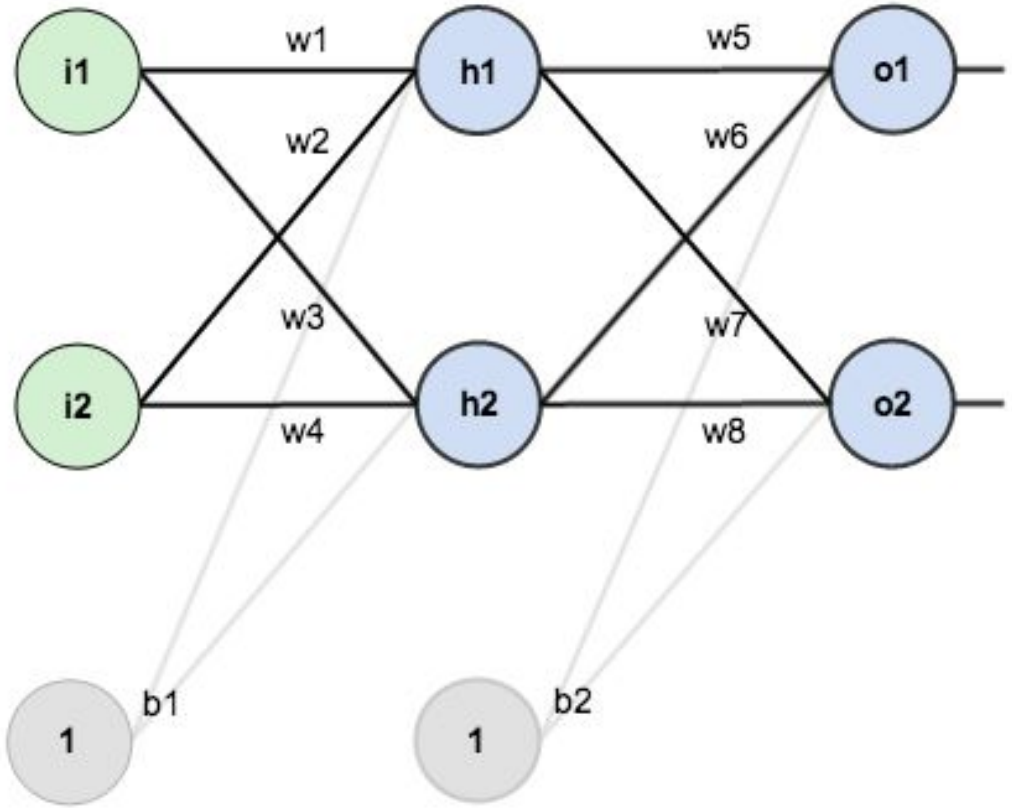
12. Update biases at the output and hidden layer: The biases in the network can be updated from the aggregated errors at that neuron.

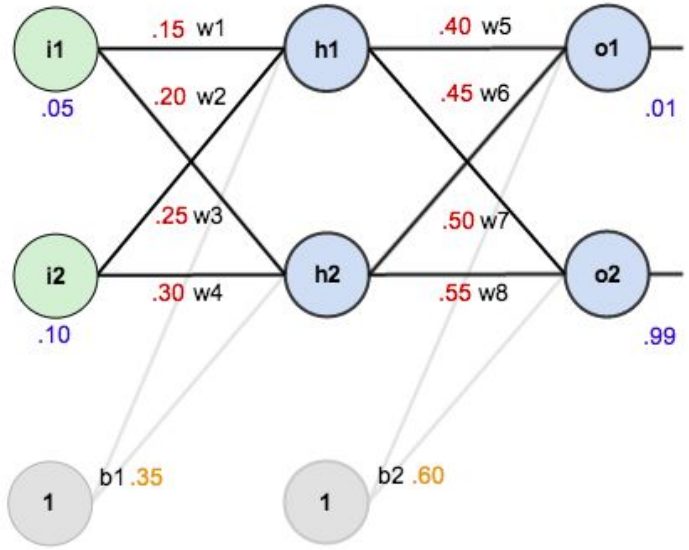
```
bh = bh + sum(d_hiddenlayer, axis=0) * learning_rate
bout = bout + sum(d_output, axis=0)*learning_rate
```

Backward Propagation Section

13. One forward and backward propagation iteration is considered as one training cycle **[Epoch].** This is an example of **full batch gradient descent algorithm.**

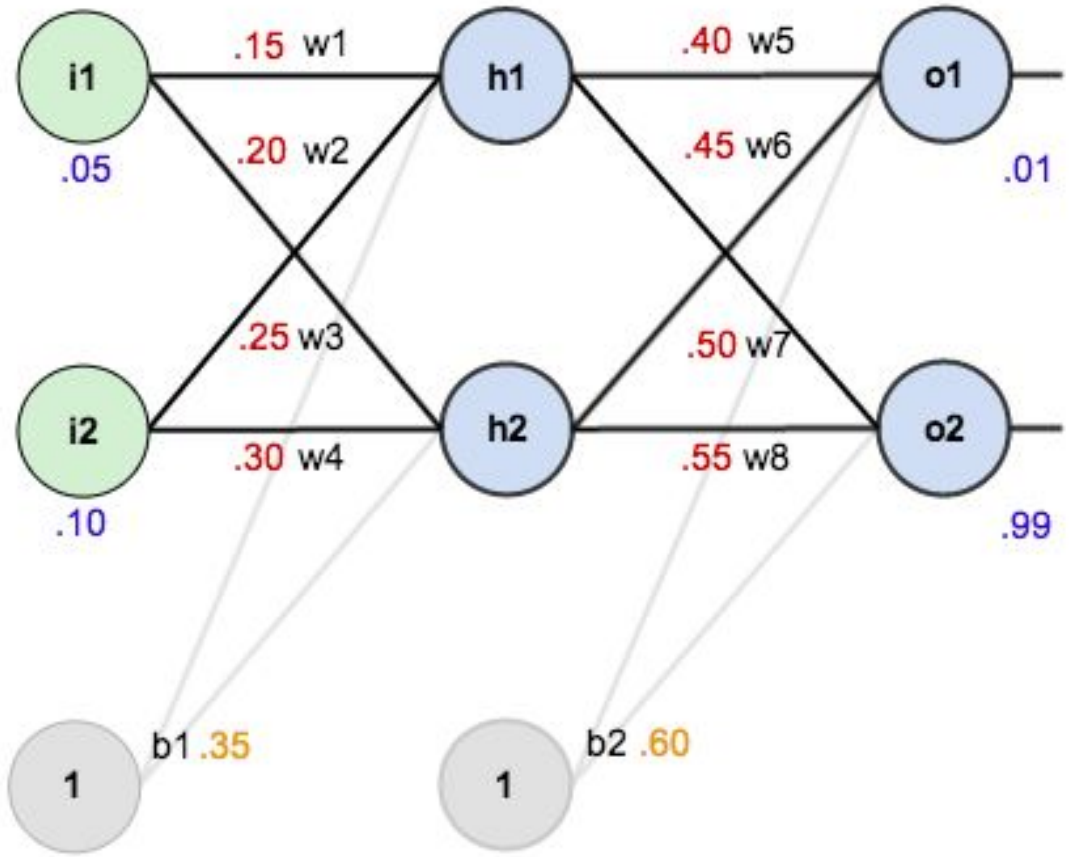
Example


i1
i2
h1
h2
o1
o2
w1
w2
w3
w4
w5
w6
w7
w8
b1
b2
1
1


i1
i2
h1
h2
o1
o2
.05
.10
.15 w1
.20 w2
.25 w3
.30 w4
.40 w5
.45 w6
.50 w7
.55 w8
.01
.99
1
b1 .35
1
b2 .60

Example

- The goal of backpropagation is to optimize the weights so that the neural network can learn how to correctly map arbitrary inputs to outputs.
- Assign Random Weights & Bias
- Given inputs 0.05 and 0.10, we want the neural network to output 0.01 and 0.99.



Forward Propagation / Forward Pass

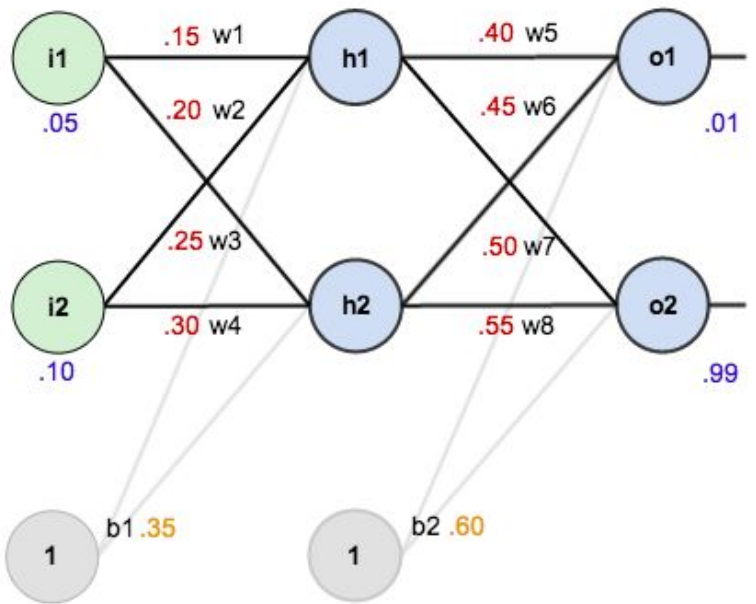
$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

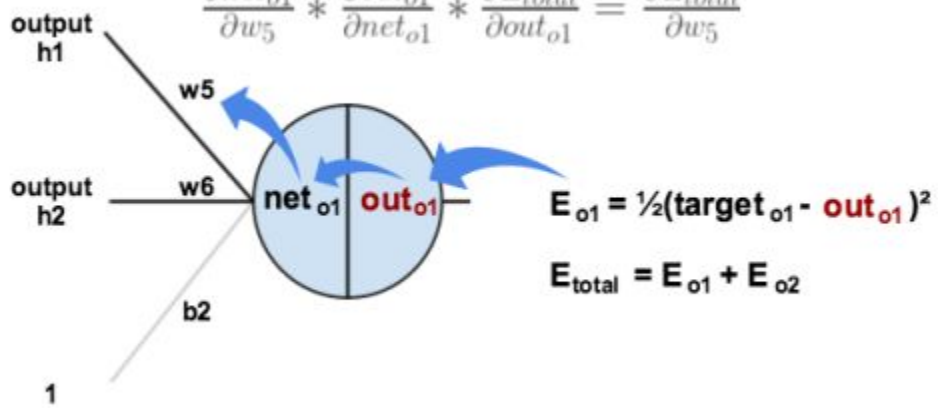
Carrying out the same process for h_2 we get:

$$out_{h2} = 0.596884378$$



Backward Pass

$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$



$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

We can repeat this process to get the new weights w_6, w_7, and w_8:

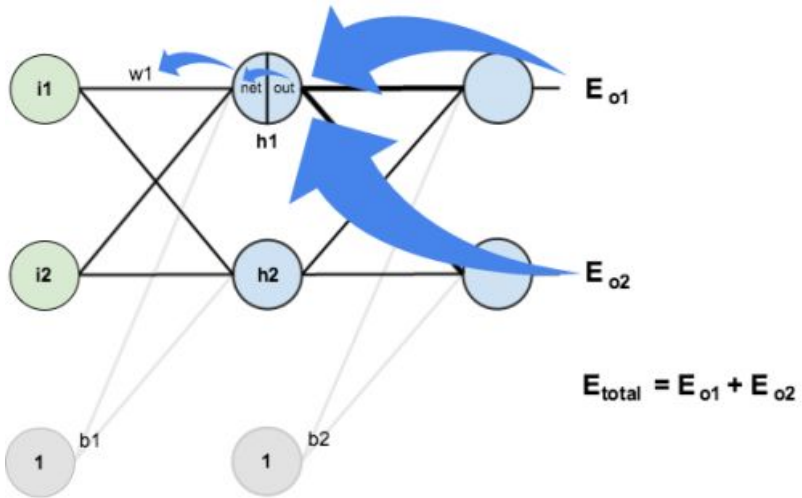
$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

$$w_8^+ = 0.561370121$$

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



$$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$$

Repeating this for w_2, w_3, and w_4

$$w_2^+ = 0.19956143$$

$$w_3^+ = 0.24975114$$

$$w_4^+ = 0.29950229$$

Thank You

Open for Questions