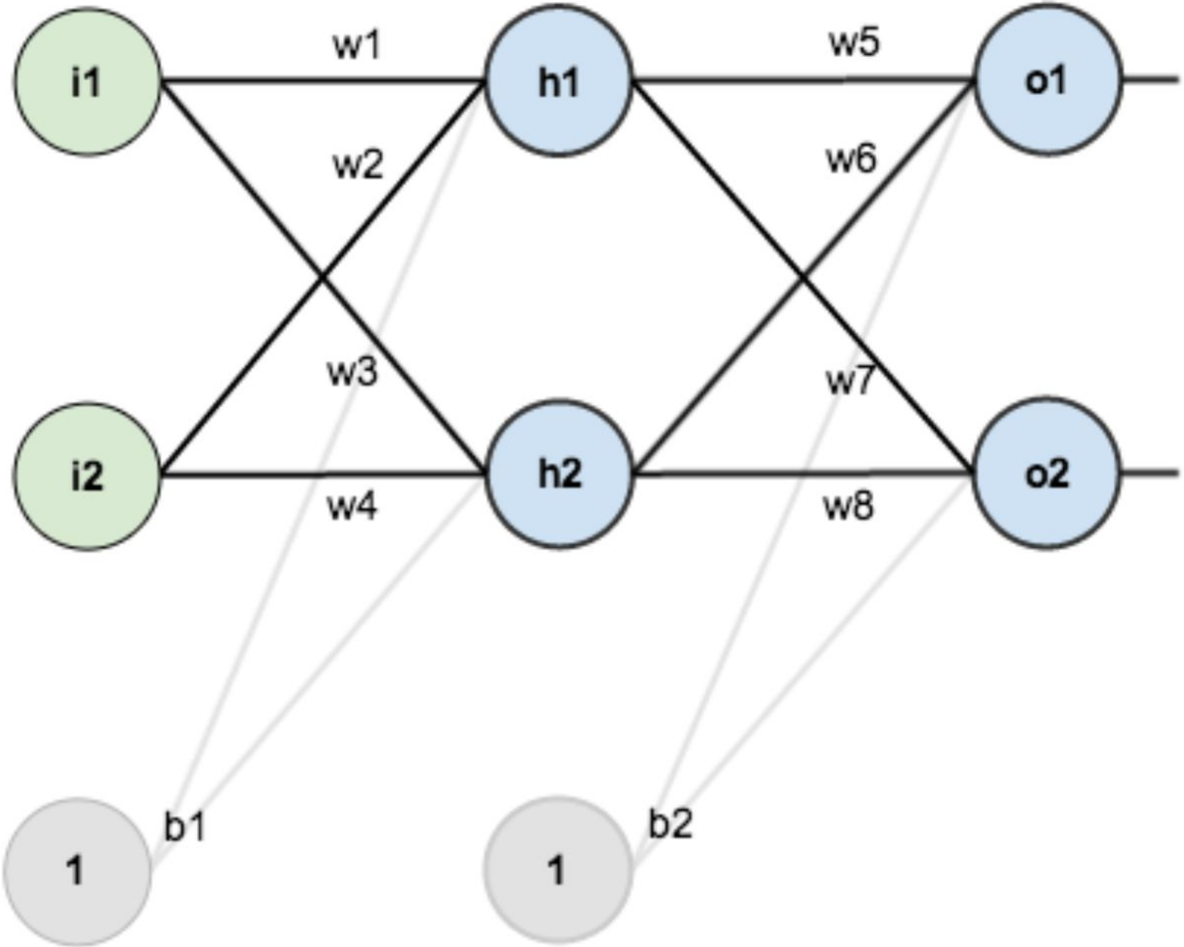
Multi-Layer Perceptron

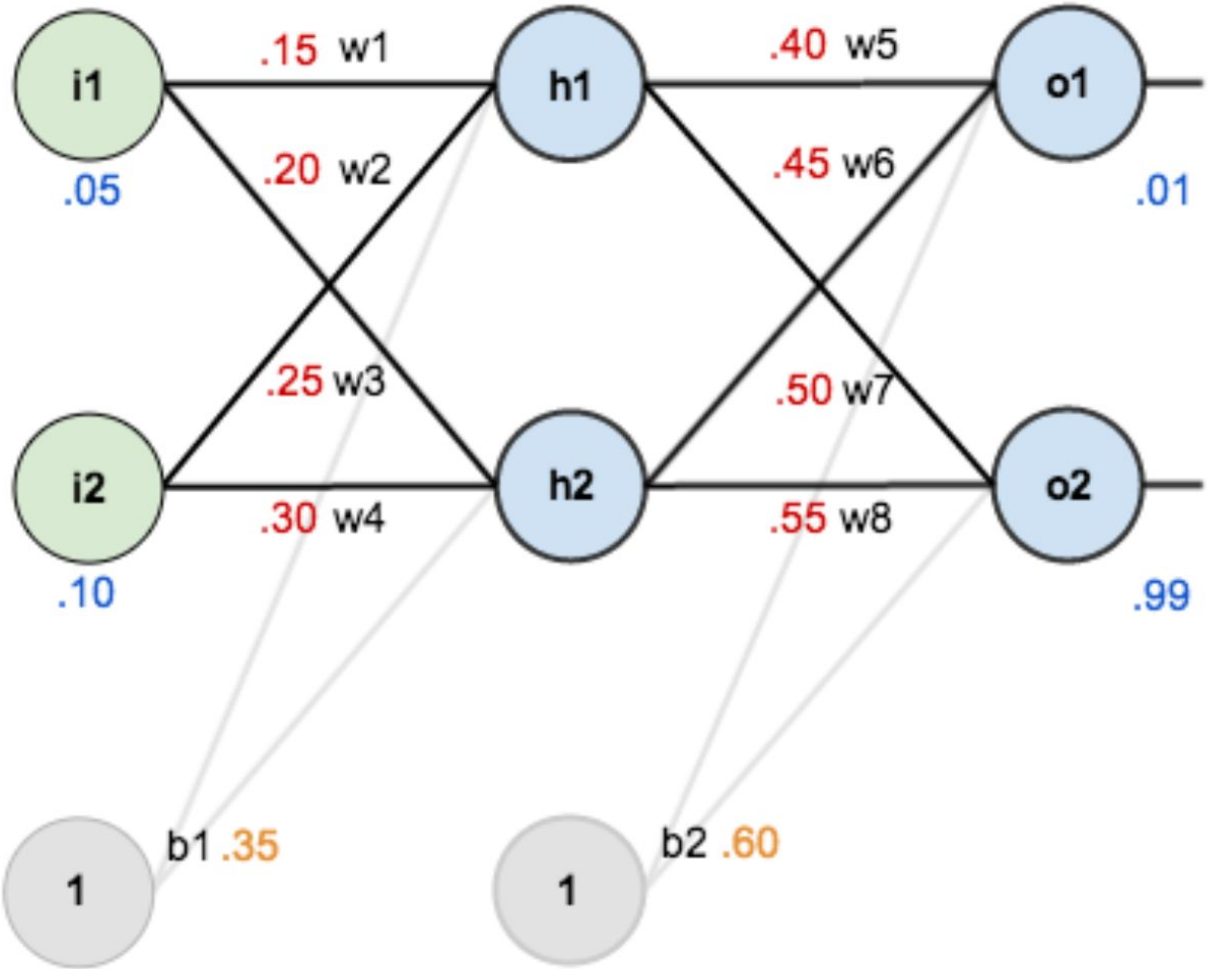
Created and Presented By:
Dr.Mydhili K Nair
Professor
Dept of ISE, MSRIT
For: Sem 6 Elective Class on Machine Learning(Jan-May 2019)



- Two Input Neurons
- Two Hidden Neurons
- Two Output Neurons
- One Bias for Hidden
- One Bias for Output

Multi-Layer Perceptron

Input		Hidden		Output	
i1	0.05	**w5**	0.40	**o1(Target)**	**0.01**
i2	0.10	**w6**	0.45	**o2(Target)**	**0.99**
w1	0.15	**w7**	0.50	**b2**	**0.60**
w2	0.20	**w8**	0.55		
w3	0.25	**b1**	0.35		
w4	0.30				



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

FORWARD PASS

FORWARD PASS ALGORITHM

Determine what the neural network currently predicts given

- the weights and biases above
- inputs of 0.05 and 0.10.

<u>Procedure</u>: Feed those inputs forward though the network.

Evaluate

- *total net input* to each hidden layer neuron
- *squash* the total net input using an *activation function* (here we use the *logistic function*)
- repeat the process with the output layer neurons.

Step 1: Calculate the Total Net Input for h_1

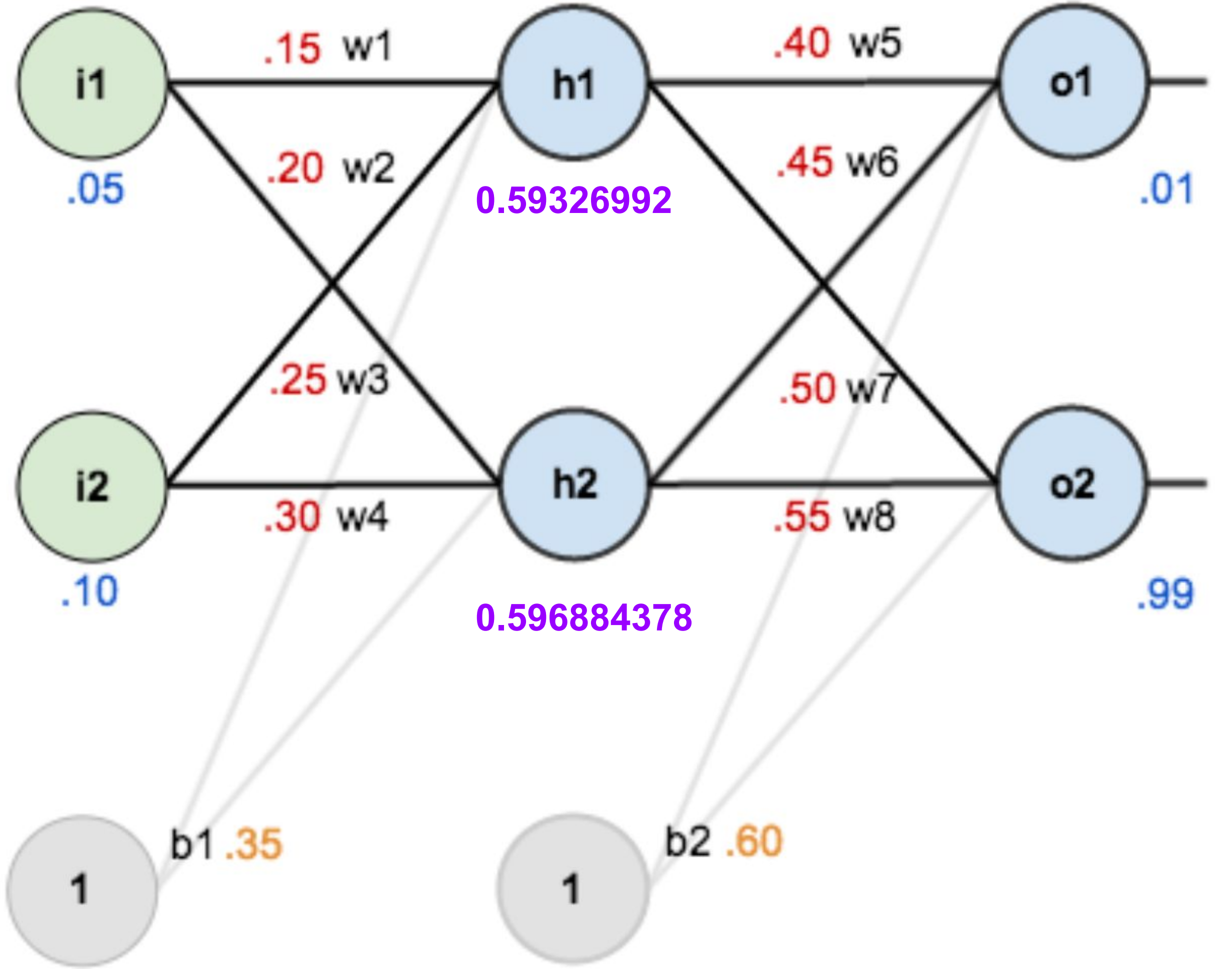
$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

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

<u>Step 2:</u> Squash it using Sigmoid Function to get Output for h_1

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

$$out_{h2} = 0.596884378$$


i1
.05
i2
.10
.15 w1
.20 w2
.25 w3
.30 w4
h1
0.59326992
h2
0.596884378
.40 w5
.45 w6
.50 w7
.55 w8
o1
.01
o2
.99
1
b1 .35
1
b2 .60

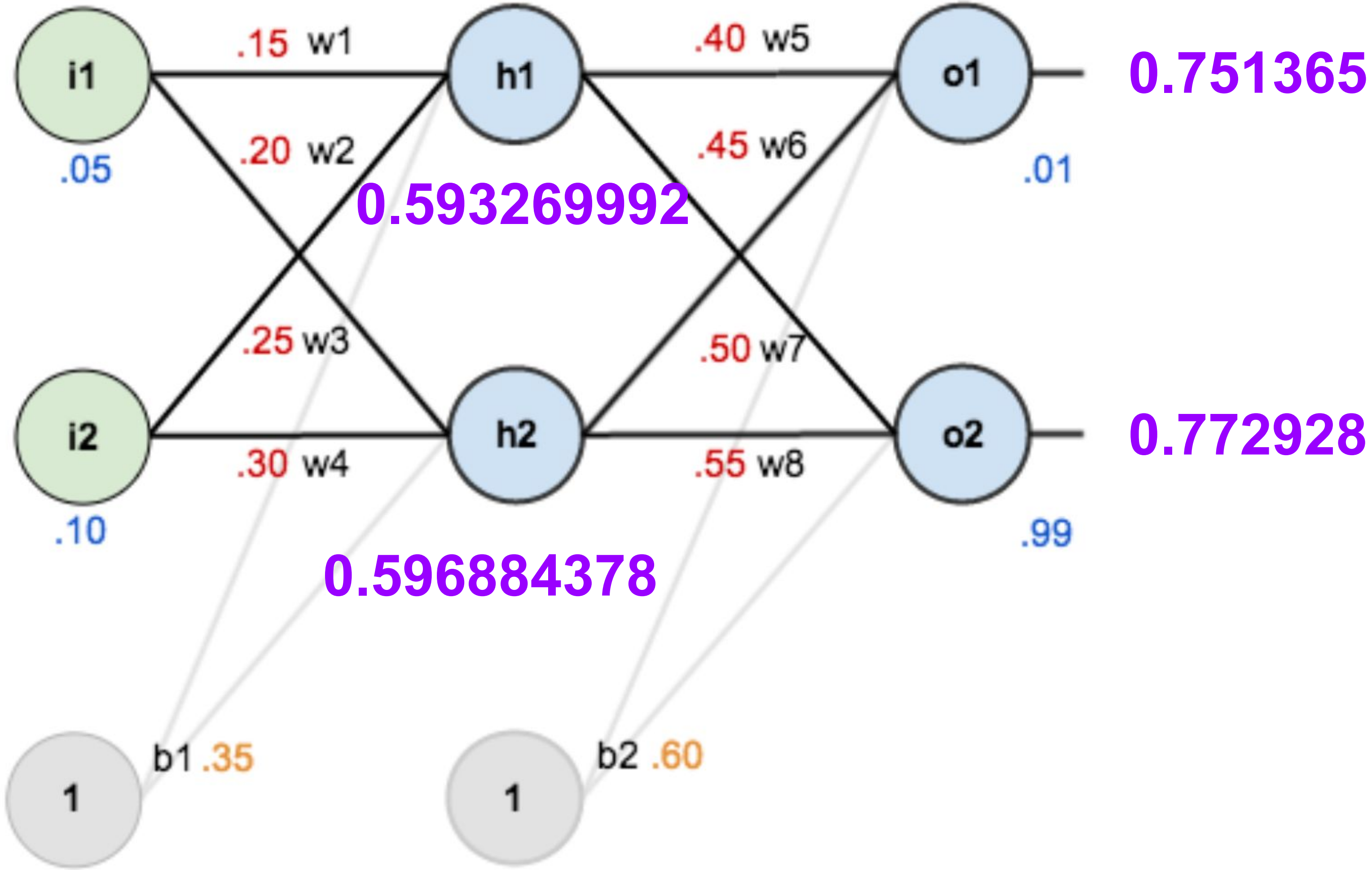
<u>Step 3:</u> Repeat this process for the output layer neurons, using the output from the hidden layer neurons as inputs.

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$

$$out_{o2} = 0.772928465$$


i1
.05
i2
.10
h1
h2
o1
.01
o2
.99
.15 w1
.20 w2
.25 w3
.30 w4
.40 w5
.45 w6
.50 w7
.55 w8
1
b1 .35
1
b2 .60
0.593269992
0.596884378
0.751365
0.772928

<u>Step 4:</u> Calculate the error for each output neuron using the squared error function

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

Divide by 2 because of two neurons.
<u>Tip:</u> Makes it convenient to cancel out the 2, during partial derivation later.

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

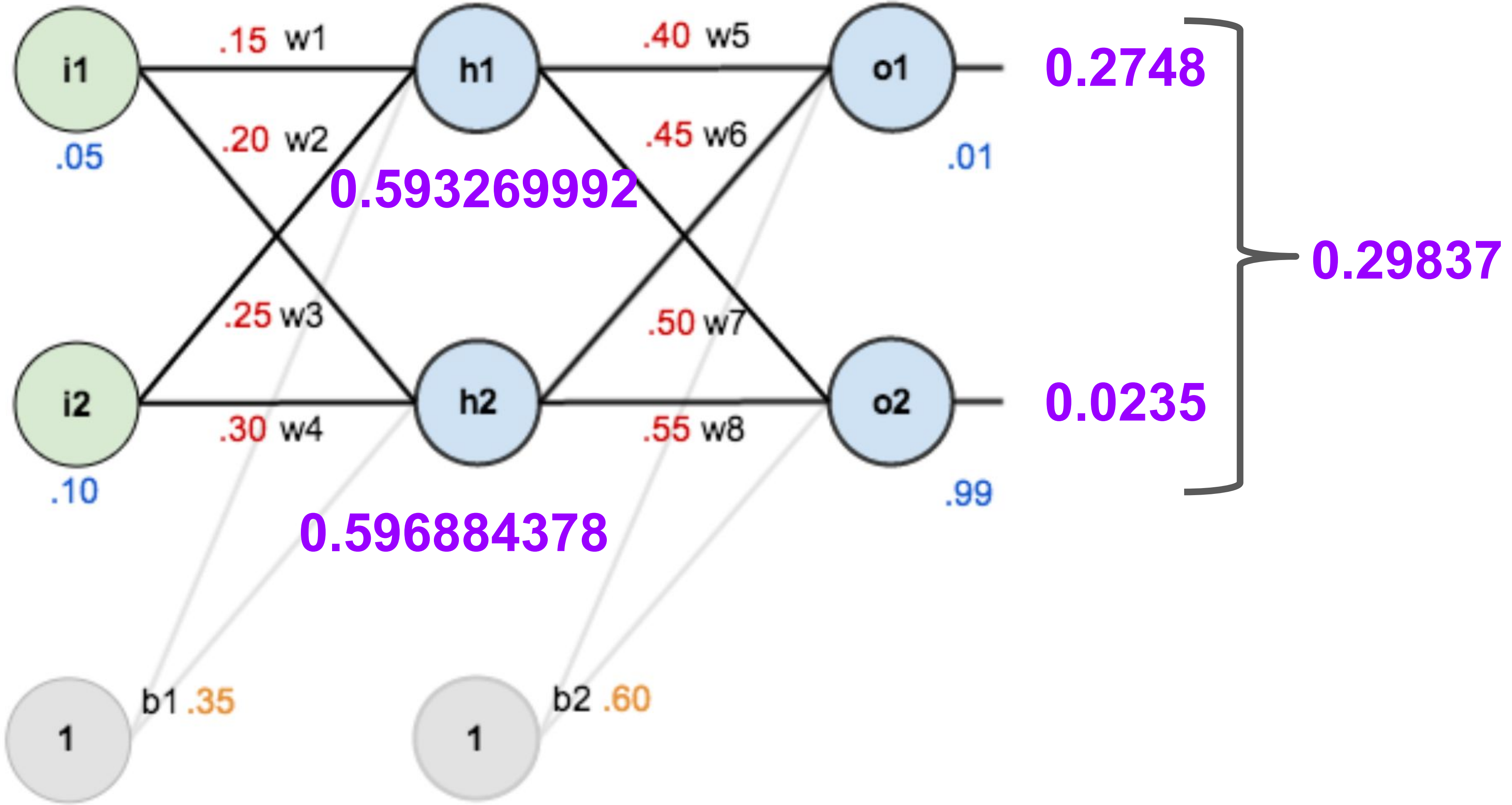
Repeating this process for o_2 (remembering that the target is 0.99) we get:

$$E_{o2} = 0.023560026$$

<u>Step 4:</u> Sum the error at each output neuron to get the total error

The total error for the neural network is the sum of these errors:

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$


i1
i2
h1
h2
o1
o2
1
1
.15 w1
.20 w2
.25 w3
.30 w4
.40 w5
.45 w6
.50 w7
.55 w8
.05
.10
.01
.99
b1 .35
b2 .60
0.593269992
0.596884378
0.2748
0.0235
0.29837

FORWARD PASS ALGORITHM

1. **Input** x**:** Set the corresponding activation a^1 for the input layer.

2. **Feedforward:** For each $l = 2, 3, \ldots, L$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$.

$\boldsymbol{l}$ represents the layer. $\boldsymbol{l}$ = 2,3 because Layer 1 (Input Layer) does not have weights. Weights are given only from the hidden layer.

$\boldsymbol{z^l}$ represents weighted input to the neurons in layer $\boldsymbol{l}$

$\boldsymbol{a^l}$ represents activation function of the neurons at layer $\boldsymbol{l}$

BACK PROPAGATION

BACKWARD PASS ALGORITHM

Our goal with backpropagation is

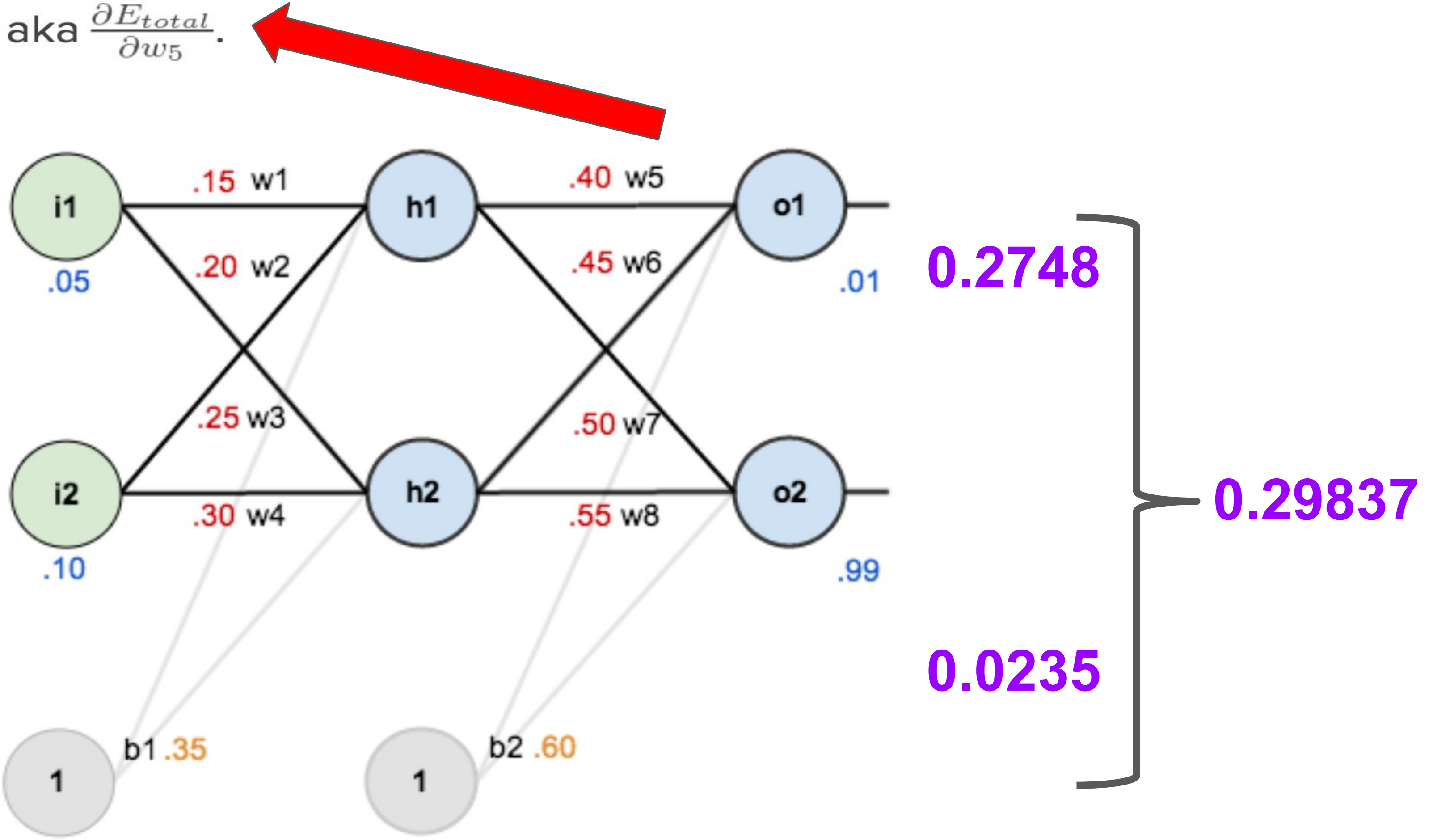
- To make the actual output to be closer the target output
- Minimize the error for each output neuron & the whole network

Procedure: Update each of the weights in the network by back-propagating the error.

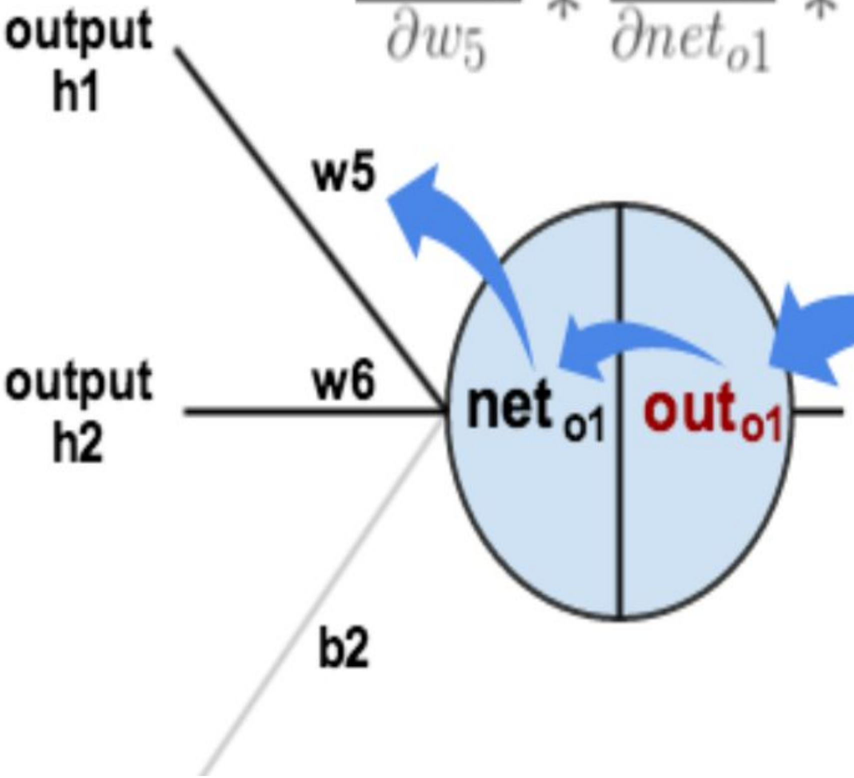
BACK PROPAGATION

---- Output Layer

Consider w_5. We want to know how much a change in w_5 affects the total error, aka $\frac{\partial E_{total}}{\partial w_5}$.



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



$$E_{o1} = \tfrac{1}{2}(target_{o1} - out_{o1})^2$$

$$E_{total} = E_{o1} + E_{o2}$$

BACK PROPAGATION ALGORITHM(output layer)

Step 1: Find the partial derivative of E_{total} w.r.t weights at output neuron, example w5 (or)
FInd The 'Gradient' w.r.t w5

Formula : "Chain Rule" In calculus, the chain rule is a formula for computing the derivative of the composition of two or more functions. Here the functions are

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

Procedure: Compute each partial derivative

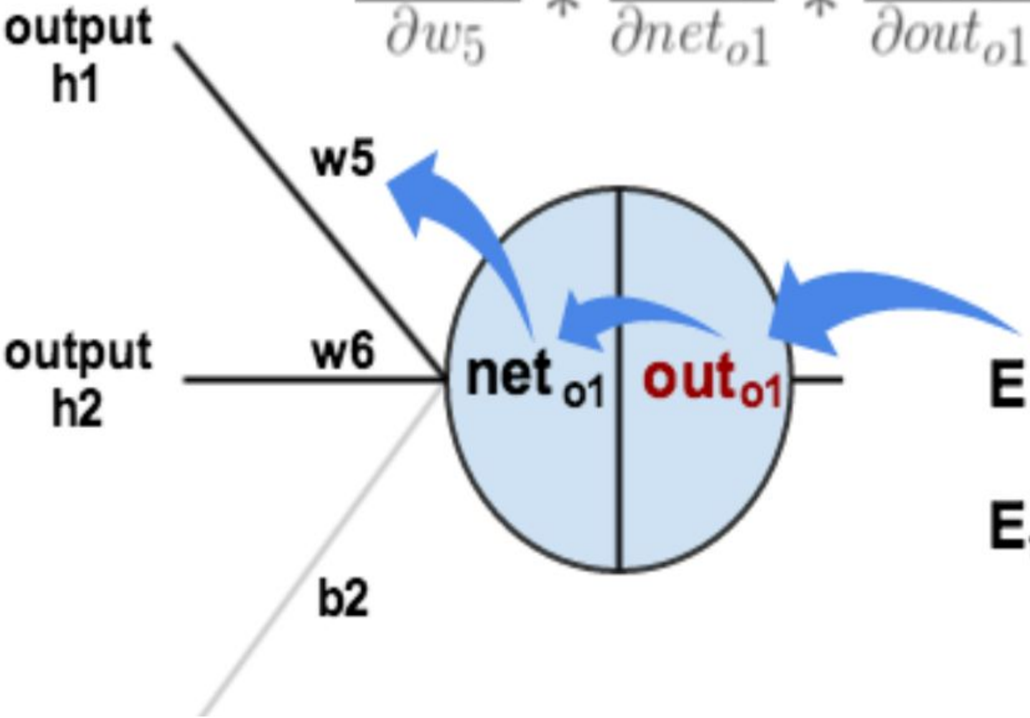
What is the effect of changing the weights to **w5** for the net summation error at **o1**?

What is the effect of changing the net summation error at **o1** to the output error (after applying activation function) at **o1**?

What is the effect of changing the output error at **o1** to the total Error?

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

Effect of weights at output neuron is a composite function of three functions ***net$_{01}$***, ***out$_{01}$*** as well as ***Etotal*** at ***o1***



$$E_{o1} = \tfrac{1}{2}(target_{o1} - out_{o1})^2$$

$$E_{total} = E_{o1} + E_{o2}$$

<u>Step 1a):</u> How much does total error change w.r.t output?

?

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

Step 1a): How much does total error change w.r.t output?

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

Chain Rule needs to be applied to differentiate as this is a composite function

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

E.g. $f(x) = (2x + 3)^5$ is partially differentiated using Chain Rule as:

$f^{l}(x) = 5 *(2x+3)^{5-1} *2$, that is, $10*(2x+3)^4$

Also the second term,

$\frac{1}{2}(target_{o2} - out_{o2})^2$ partial derivative is a constant ***out***$_{01}$ does not effect it - no ***out***$_{01}$ term in the expression, therefore evaluated to zero.

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

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

<u>Step 1b):</u> How much does output change w.r.t net_{01} (total net output of o1)?

$$0.74136507$$

?

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

<u>Step 1b):</u> How much does output change w.r.t net_{01} (total net output of o1)?

net_{01} is calculating using an activation function. Here, we have considered 'Sigmoid'. Therefore, the partial derivative of 'Sigmoid' must be taken.

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1})$$

$$= 0.75136507(1 - 0.75136507) = 0.186815602$$

<u>Step 1c):</u> How much does net_{o1} (total net output of o1) change w.r.t w5?

?

0.74136507 0.186815602

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

Step 1c): How much does net_{01} (total net output of o1) change w.r.t w5?

Formula is:

$$net_{o1} = \underbrace{w_5 * out_{h1}} + w_6 * out_{h2} + b_2 * 1$$

Chain Rule needs to be applied to differentiate as this is a composite function

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$

Putting it all together:

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

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = \underline{0.082167041}$$

$$\underbrace{\frac{\partial E_{total}}{\partial w_5}}_{} = \overset{0.74136507}{\frac{\partial E_{total}}{\partial out_{o1}}} * \overset{0.186815602}{\frac{\partial out_{o1}}{\partial net_{o1}}} * \overset{0.593269992}{\frac{\partial net_{o1}}{\partial w_5}}$$

BACK PROPAGATION ALGORITHM(output layer)

Step 2: **To decrease the error**

- **subtract this value of** $\frac{\partial E_{total}}{\partial w_5}$ **from the current weight**
- **optionally multiply** $\frac{\partial E_{total}}{\partial w_5}$ **by some learning rate, eta, a value between 0 to 1. Here we'll set to 0.5.**

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

BACK PROPAGATION ALGORITHM(output layer)

<u>Step 3:</u> Repeat the process to get the new weights of w6,w7 and w8.

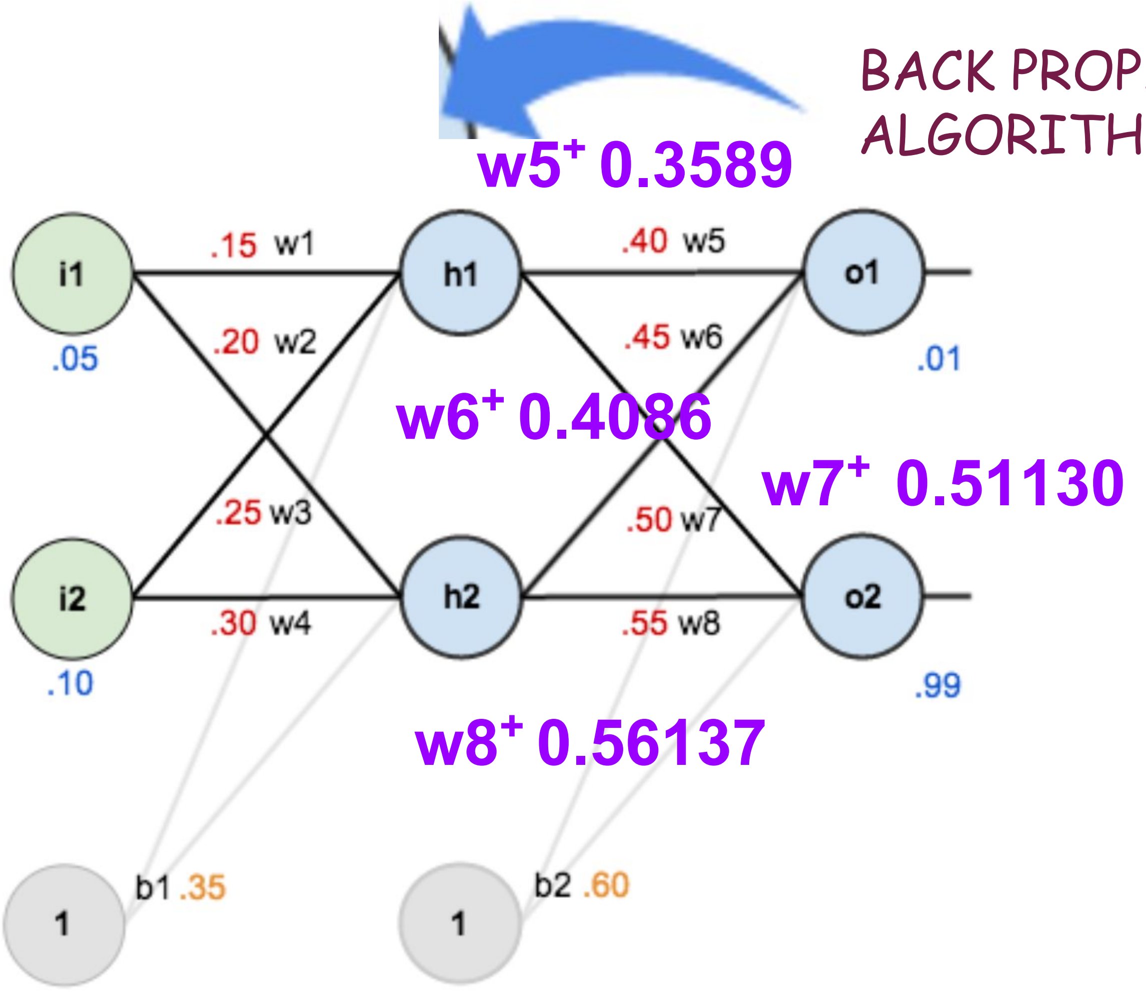
$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

$$w_8^+ = 0.561370121$$

<u>Point to Note:</u> To calculate the Back Propagation Error, we use the original weights of the Hidden Layer w6, w7 and w8 and **NOT** the updated weights $w6^+$, $w7^+$ and $w8^+$.

BACK PROPAGATION ALGORITHM(output layer)



BACK PROPAGATION ALGORITHM

1. **Input** x**:** Set the corresponding activation a^1 for the input layer.

2. **Feedforward:** For each $l = 2, 3, \ldots, L$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$.

4. **Backpropagate the error:** For each $l = L-1, L-2, \ldots, 2$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.

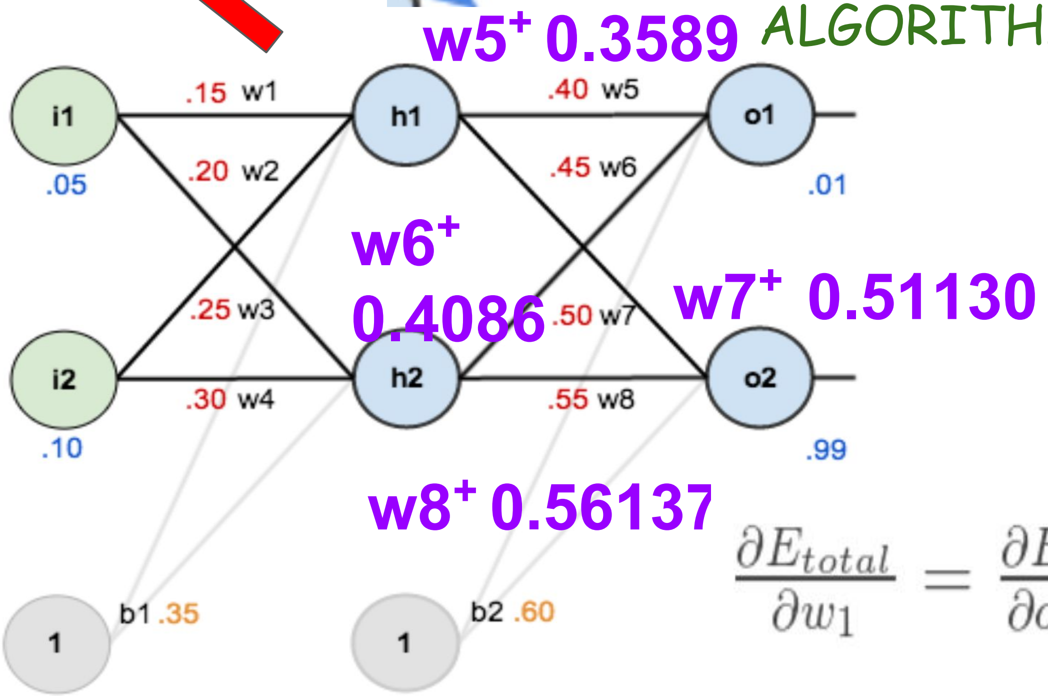
5. **Output:** The gradient of the cost function is given by $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$.

BACK PROPAGATION

---Hidden Layer

Consider w1. We want to know, how much change in w1 affects the total error, that is : $\frac{\partial E_{total}}{\partial w_1}$

EFFECT OF BACK PROPAGATION ALGORITHM(hidden layer)



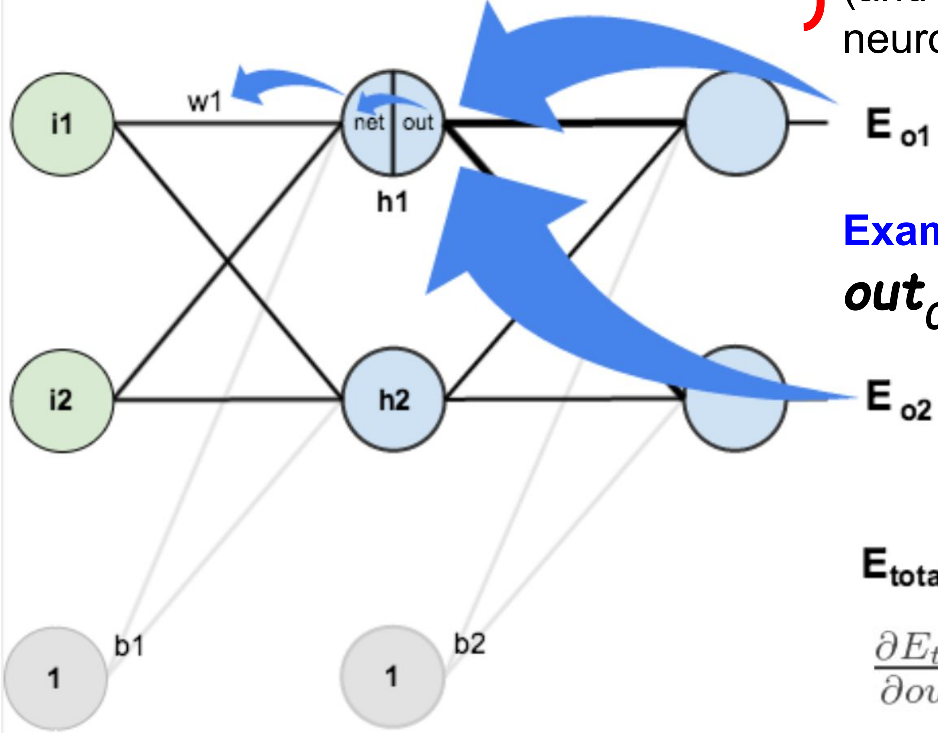
Mathematically,

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

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

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

NOTE: The output of each hidden layer neuron contributes to the output (and therefore error) of multiple output neurons.



Example: out_{01} affects both out_{01} and out_{02}.

$E_{total} = E_{o1} + E_{o2}$

$\frac{\partial E_{total}}{\partial out_{h1}}$ has to consider effect on both the output neurons.

BACK PROPAGATION ALGORITHM(hidden layer)

<u>Step 1:</u> Find the partial derivative of E_{total} w.r.t weights at hidden layer neuron, example w1(or)
FInd The ‘Gradient’ w.r.t w1

<u>Formula</u> : “Chain Rule” In calculus, the chain rule is a formula for computing the derivative of the composition of two or more functions. Here the functions are

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

<u>Procedure:</u> Compute each partial derivative

Step 1a): How much does total error change w.r.t output?

$$\frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}} \qquad \frac{\partial E_{o2}}{\partial net_{o2}} * \frac{\partial net_{o2}}{\partial out_{h1}}$$

?

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

Already Calculated in Step 1b)

<u>NOTE</u> ADDITIONAL CALCULATION USING CHAIN RULE

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

<u>Step 1a(i):</u> How much does total error change w.r.t error at output neuron 1 that is o1, symbolically $\frac{\partial E_{o1}}{\partial out_{h1}}$?

Starting with $\frac{\partial E_{o1}}{\partial out_{h1}}$:

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}$$

We can calculate $\frac{\partial E_{o1}}{\partial net_{o1}}$ using values we calculated earlier:

$$\frac{\partial E_{o1}}{\partial net_{o1}} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = 0.74136507 * 0.186815602 = 0.138498562$$

And $\frac{\partial net_{o1}}{\partial out_{h1}}$ is equal to w_5:

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.40$$

Plugging them in:

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}} = 0.138498562 * 0.40 = 0.055399425$$

<u>Step 1a(ii):</u> How much does total error change w.r.t error at output neuron 2 that is o2, symbolically $\frac{\partial E_{o2}}{\partial out_{h1}}$?

Following the same process for $\frac{\partial E_{o2}}{\partial out_{h1}}$, we get:

$$\frac{\partial E_{o2}}{\partial out_{h1}} = -0.019049119$$

<u>Step 1a:</u> Add these two partial derivative values:

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

$$= 0.055399425 + -0.019049119 = 0.036350306$$

<u>Step 1b):</u> How much does output change w.r.t net_{01} (total net hidden layer of h1)?

0.036350306

?

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

<u>Step 1b):</u> How much does output change w.r.t net_{01} (total net hidden layer of h1)?

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

$$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1 - out_{h1}) = 0.59326999(1 - 0.59326999) = 0.241300709$$

Step 1c): How much does net_{01} (total net output of o1) change w.r.t w5?

?

0.036350306 0.241300709

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

<u>Step 1c):</u> How much does net_{01} (total net output of o1) change w.r.t w5?

We calculate the partial derivative of the total net input to h_1 with respect to w_1 the same as we did for the output neuron:

$$net_{h1} = w_1 * i_1 + w_3 * i_2 + b_1 * 1$$

$$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$$

Putting it all together:

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

$$\frac{\partial E_{total}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05 = \underline{0.000438568}$$

$$0.036350306 \quad 0.241300709 \quad 0.05$$

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

BACK PROPAGATION ALGORITHM(hidden layer)

Step 2: **To decrease the error**

- **subtract this value of** $\frac{\partial E_{total}}{\partial w_1}$ **from the current weight**
- **optionally multiply** $\frac{\partial E_{total}}{\partial w_1}$ **by some learning rate, eta, a value between 0 to 1. Here we'll set to 0.5.**

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

BACK PROPAGATION ALGORITHM(hidden layer)

Step 3: Repeat the process to get the new weights of w6,w7 and w8.

Repeating this for w_2, w_3, and w_4

$$w_2^+ = 0.19956143$$

$$w_3^+ = 0.24975114$$

$$w_4^+ = 0.29950229$$

Point to Note: In the following Feed Forward Phase, the updated weights $w1^+$, $w2^+$ and $w3^+$ are used by the hidden layer and $w6^+$, $w7^+$ and $w7^+$ are used by the output layer.

Originally: The Total Error on the network was **0.298371109.**

AFTER NEXT FORWARD PASS

That is, After this first round of backpropagation,

--- Using the Updated weights

---- $w1^+$, $w2^+$ and $w3^+$ in the hidden layer

---- $w6^+$, $w7^+$ and $w7^+$ in the output layer

Total error is now down to **0.291027924.**

Inference: It might not seem like much, but after repeating this process **10,000 times**, for example, the error plummets to **0.0000351085**. At this point, when we feed forward 0.05 and 0.1, the two outputs neurons generate **0.015912196** (vs 0.01 target) and **0.984065734** (vs 0.99 target).

BACK PROPAGATION ALGORITHM

1. **Input** x**:** Set the corresponding activation a^1 for the input layer.

2. **Feedforward:** For each $l = 2, 3, \ldots, L$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$.

4. **Backpropagate the error:** For each $l = L-1, L-2, \ldots, 2$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.

5. **Output:** The gradient of the cost function is given by $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$.