MLAI 504
NEURAL NETWORKS & DEEP LEARNING

MLAI – 2023/2024

Dr. Zein Al Abidin IBRAHIM

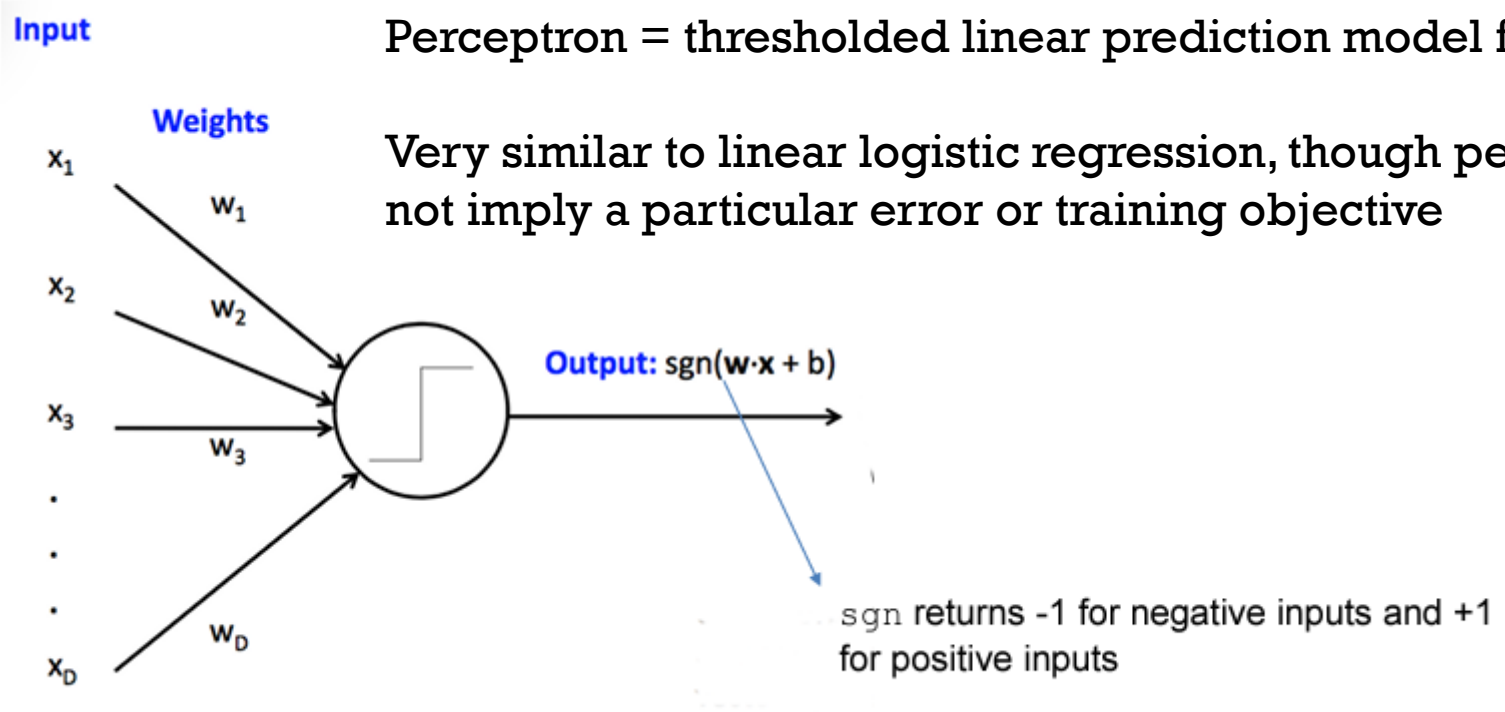
zein.ibrahim@ul.edu.lb

MLP & BACKPROPAGATION

NEURAL NETWORK

Adapted from Peter Lewis slides

RECAP: PERCEPTRONS



Perceptron = thresholded linear prediction model for classification

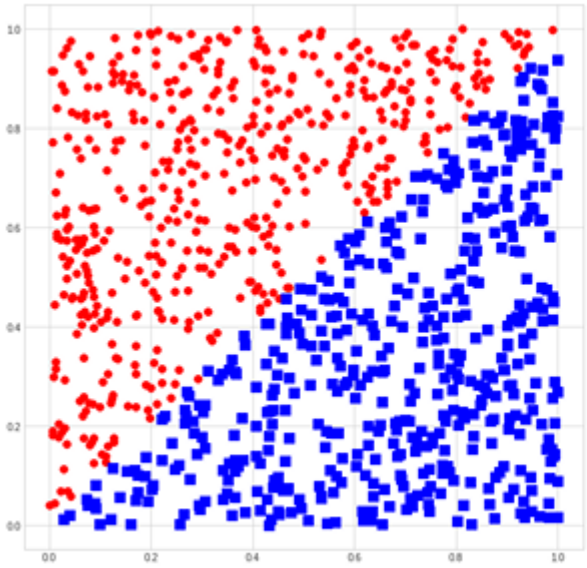
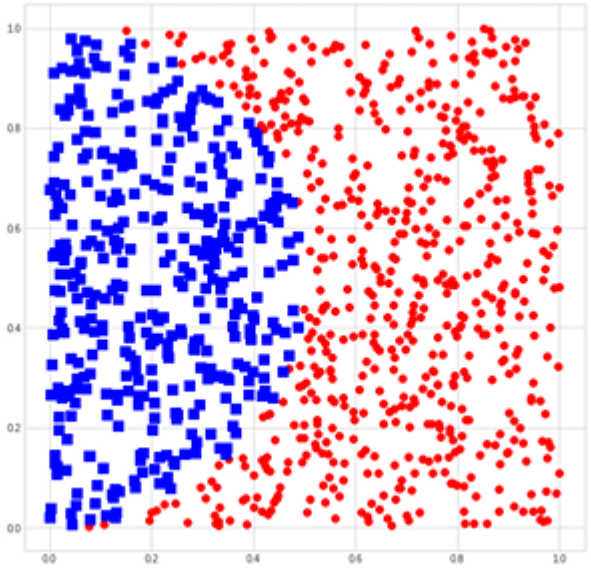
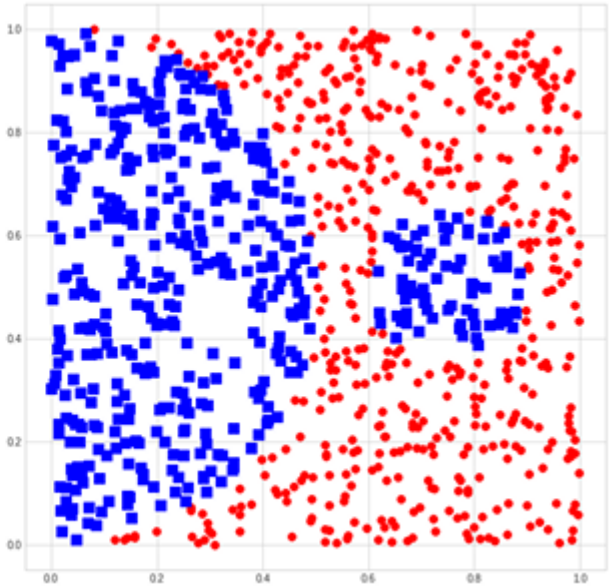
Very similar to linear logistic regression, though perceptron does not imply a particular error or training objective

A key limitation of the single perceptron

Only works for linearly separable data.

IS A PERCEPTRON ENOUGH?

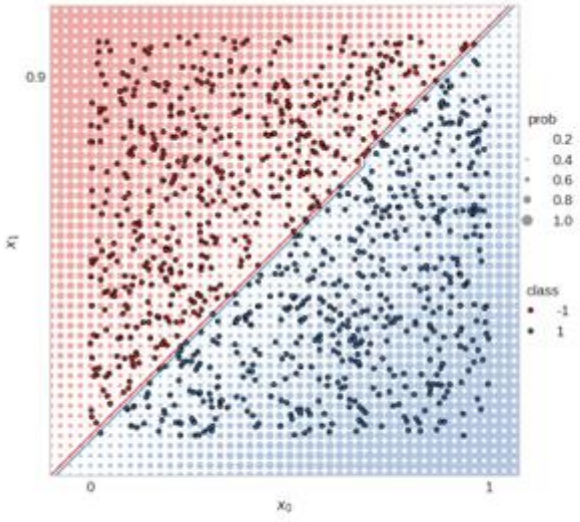
- Which of these can a perceptron solve (fit with zero training error)?

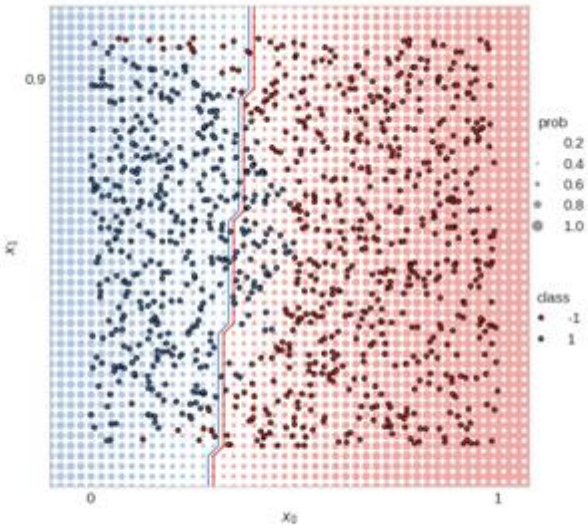
Demo:
https://colab.research.google.com/drive/1nKNJyolqgzW53Rz59M2BZtyQM8bbrExb?usp=sharing

PERCEPTRON IS OFTEN NOT ENOUGH

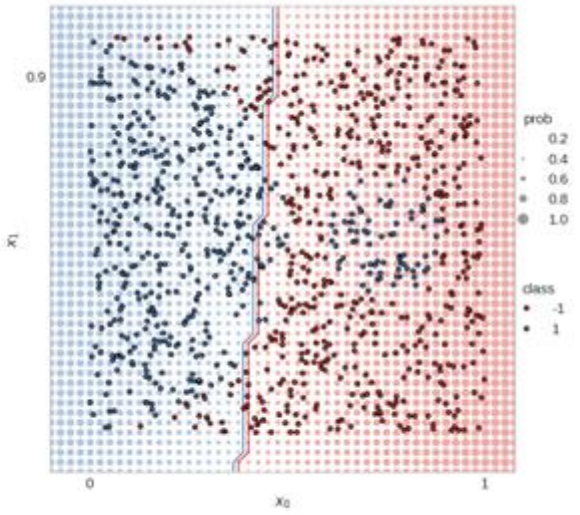
- Perceptron is linear, but we often need a non-linear prediction function



Yes



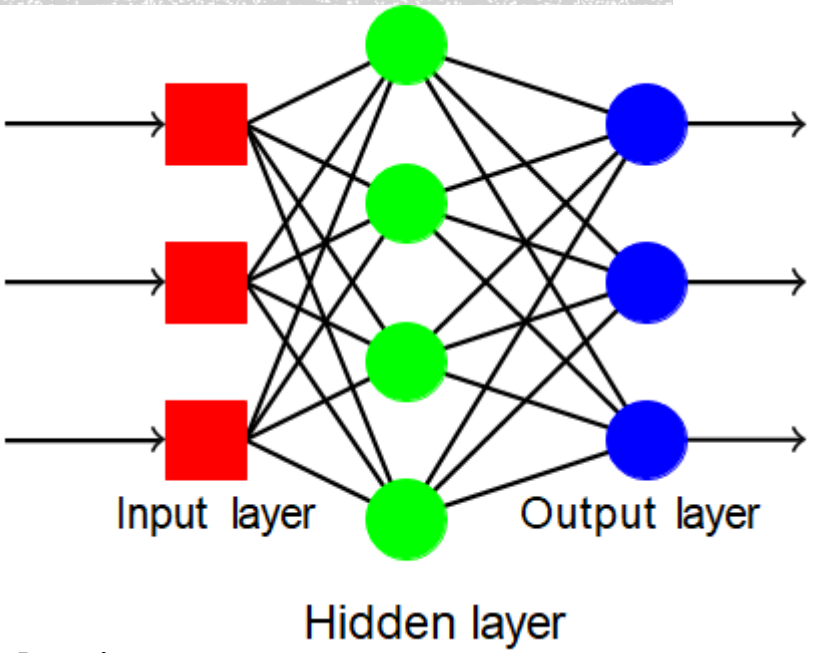
No



Not even close

MULTILAYER PERCEPTRON (MLP)

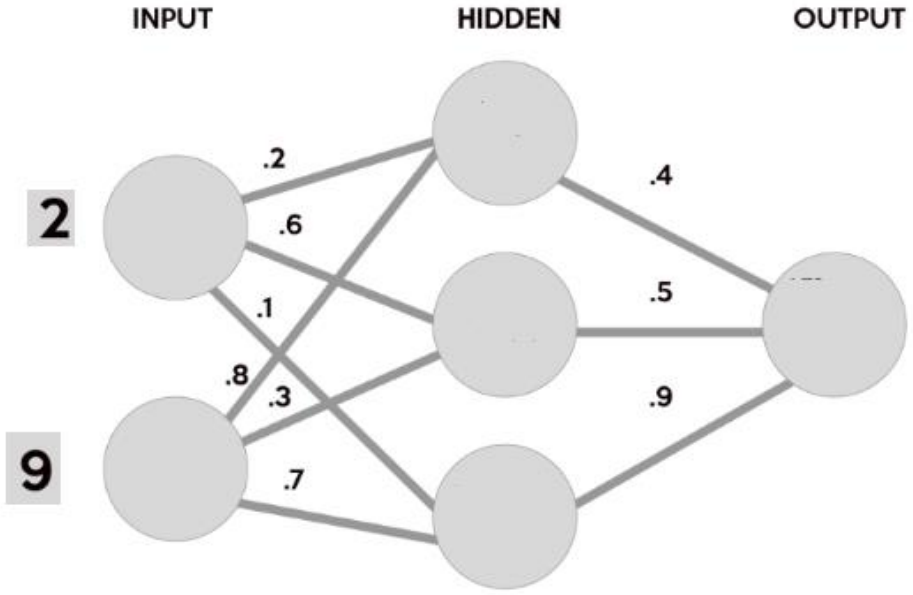
MLPs are **feed-forward** neural networks.



- Organised in layers:
  - One input layer of distribution points.
  - One or more hidden layers of artificial neurons (nodes).
  - One output layer of artificial neurons (nodes).
  - Each node in a layer is connected to all other nodes in the next layer. Each connection has a weight (remember that the weight can be zero).
- MLPs are universal approximators!

HIDDEN NEURONS ROLE

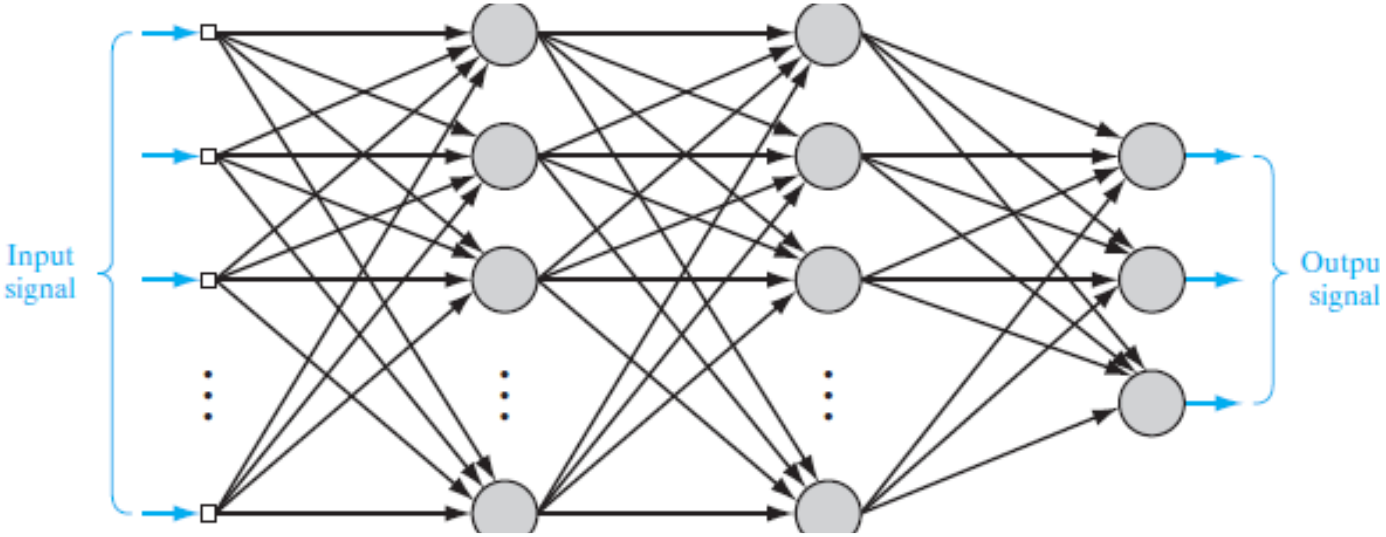
- Not part of the output layer of the network.
- Serves as feature detectors:
- Gradually discover the main features that characterize the data
  - transformation from input space to feature space
  - In this space, data may be more easily separated than the original space.

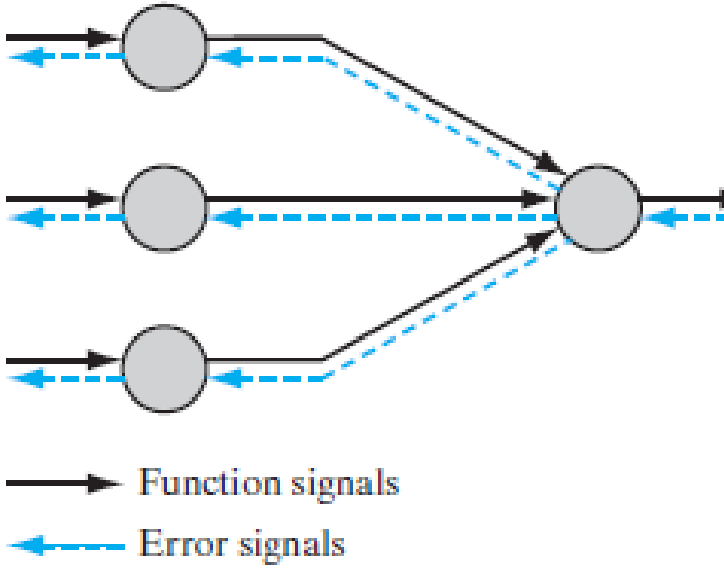


what is the output of the hidden layer ?

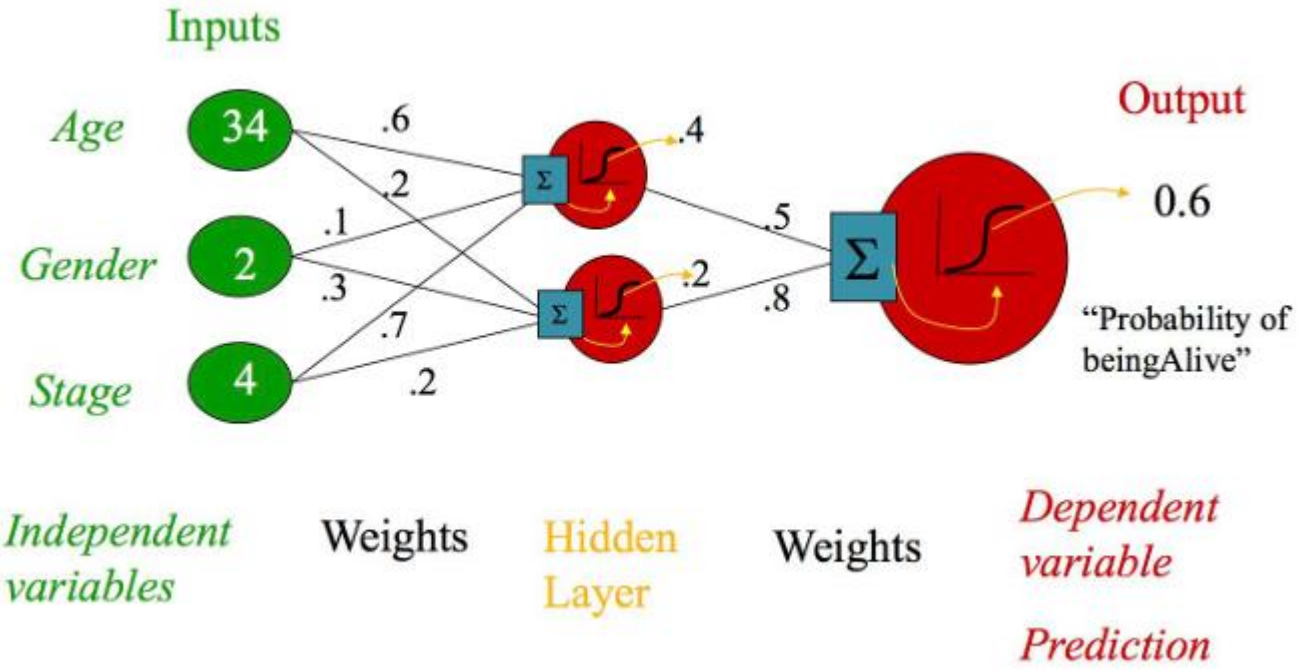
TYPES OF SIGNALS IN MLP

- Two types of signals:
  - Function Signals: comes in at the input end of the network and propagates forward neuron by neuron.
  - Error Signals: Originates at the output of the network and propagates backward layer by layer through the network.

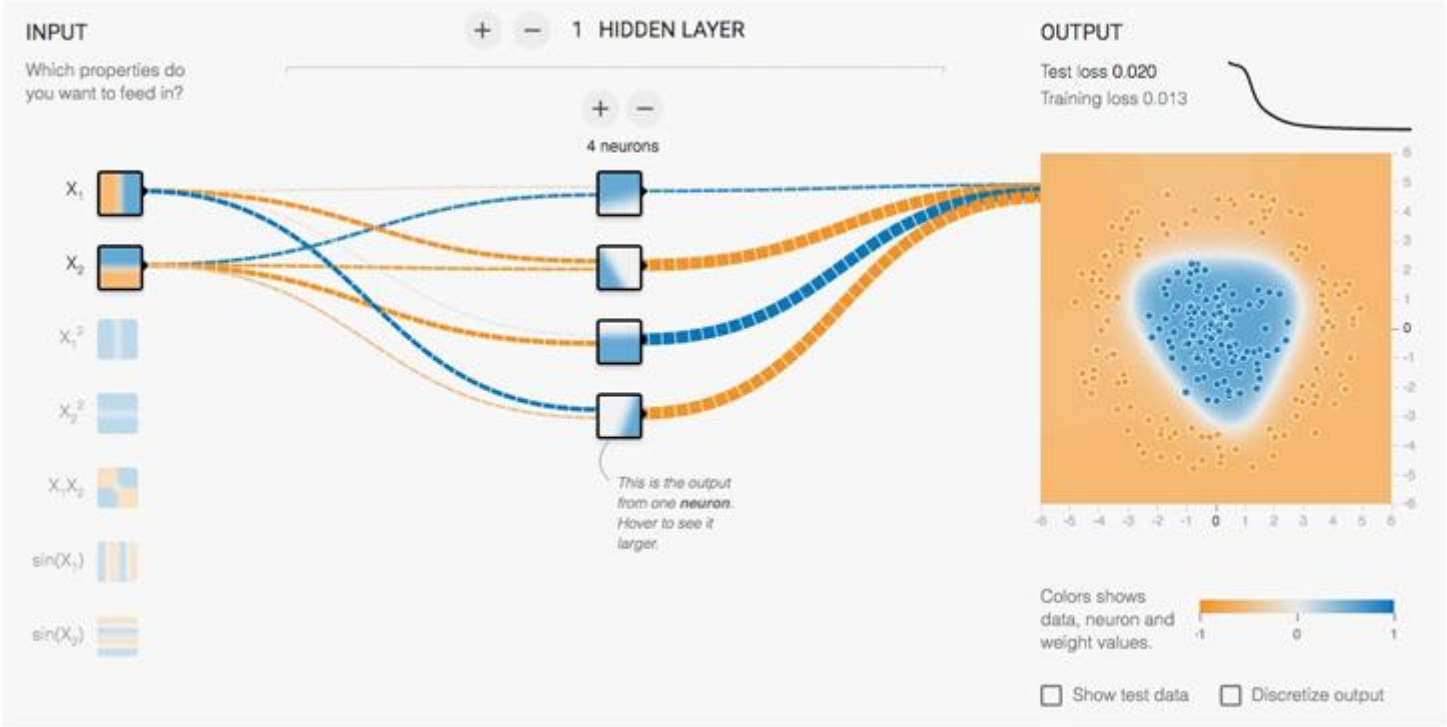




MULTILAYER PERCEPTRON (MLP)



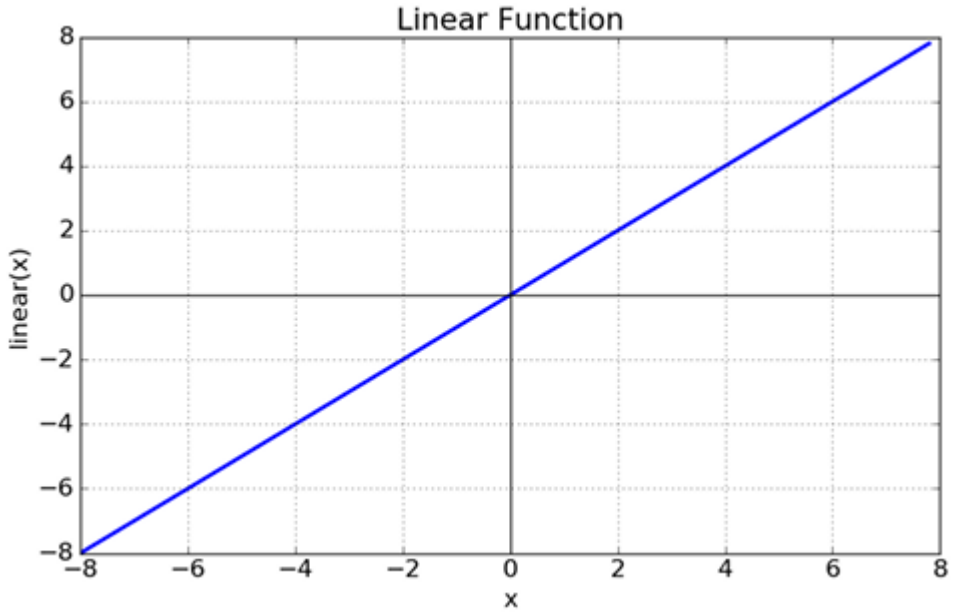
MULTI-LAYER NETWORK ONLINE DEMO



http://playground.tensorflow.org/

LINEAR ACTIVATION

- A no-op activation (i.e. nothing happens)
- Could be used for information compression or data alignment
- Multiple stacked linear layers are equivalent to a single linear layer

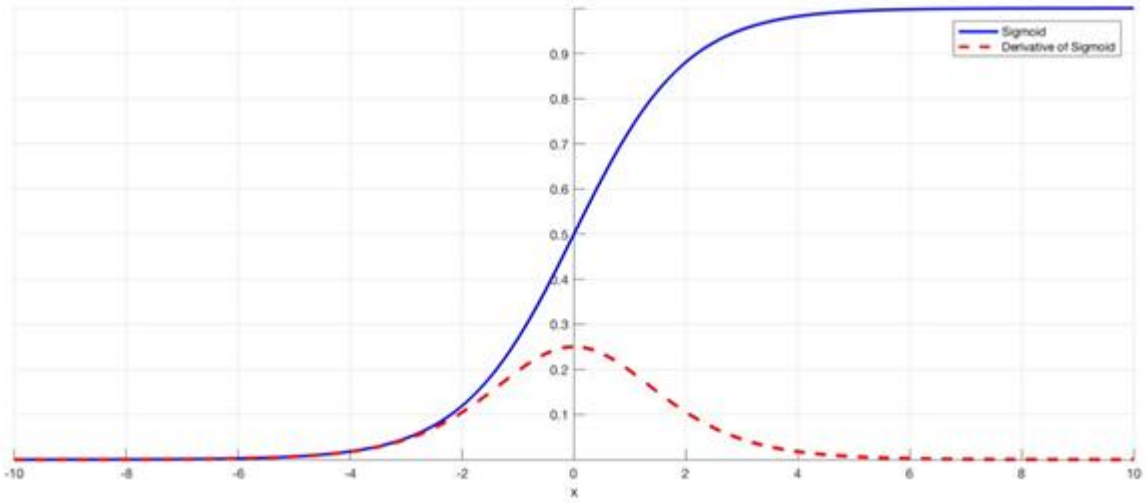


$$f(x) = x$$

$$f'(x) = 1$$

SIGMOID ACTIVATION

- Maps any value to 0 to 1 range
- Traditionally, a common choice for internal layers
- But weak gradients at extremum make it difficult to optimize if there are many layers ("vanishing gradient problem")
- Common choice for output layer to map to a probability



$$f(x) = \frac{1}{1 + \exp(-x)}$$

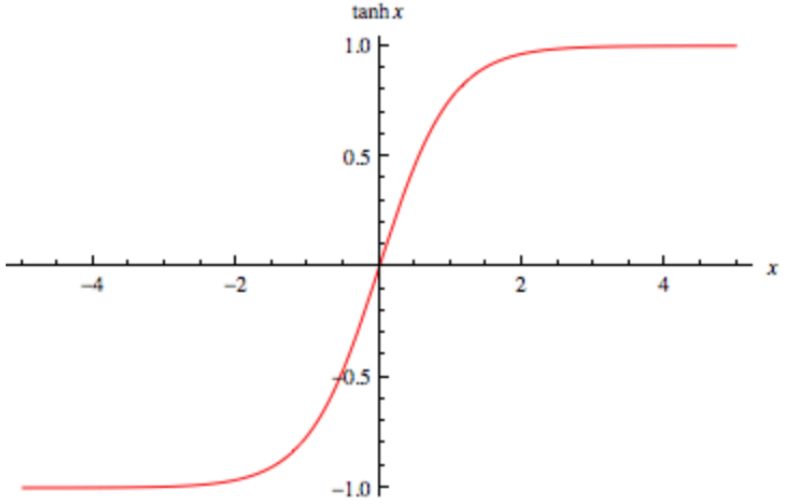
$$f'(x) = f(x)(1 - f(x))$$

- Another version with a

$$f(x) = \frac{1}{1+\exp(-ax)} \rightarrow f'(x) = \mathrm{a}\, f(x)(1 - f(x))$$

HYPERBOLIC TANGENT ACTIVATION

- Like sigmoid
- Maps any value to -1 to 1 range
- differentiable



$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

$$f'(x) = 1 - f(x)^2$$

MLPS: UNIVERSAL APPROXIMATORS

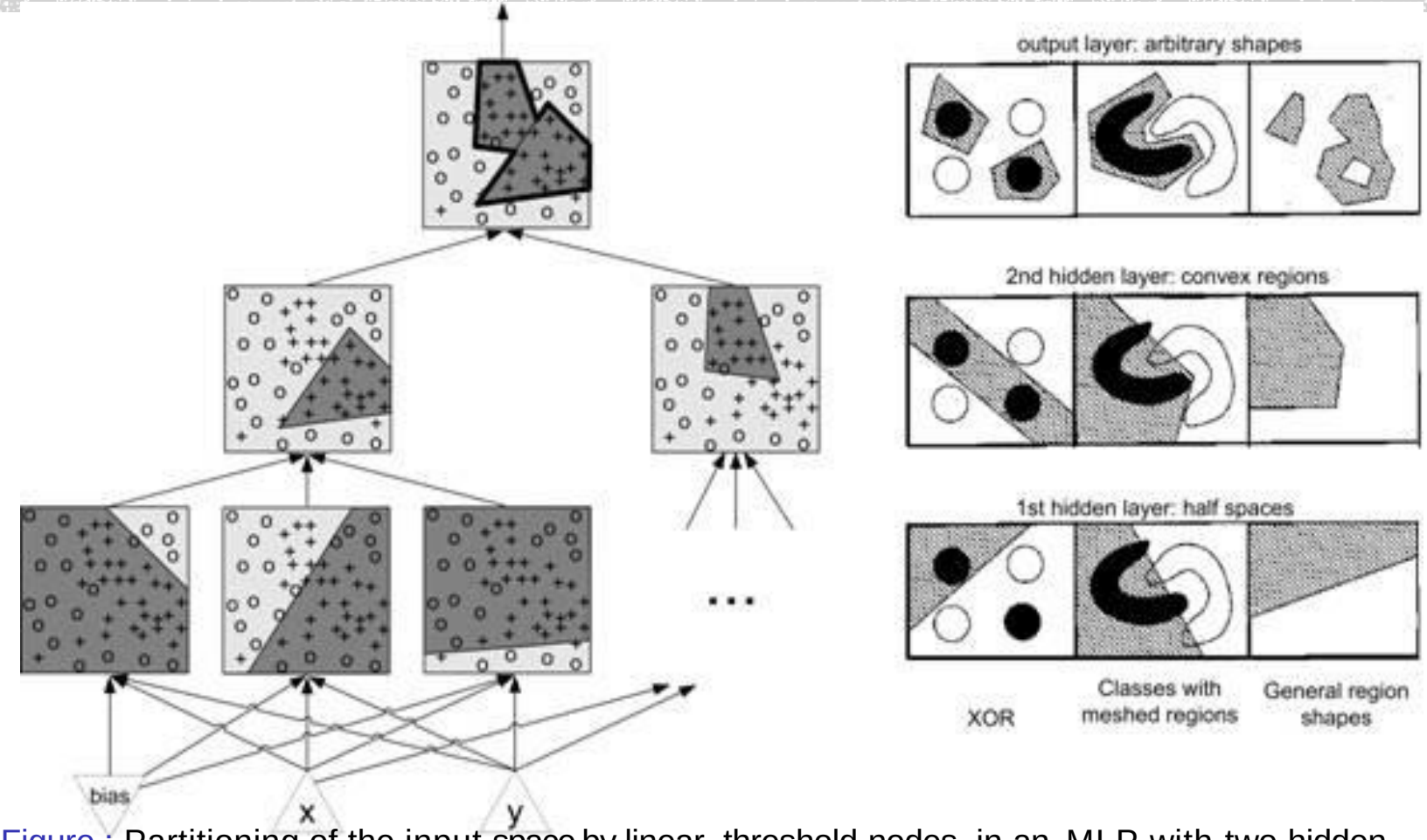
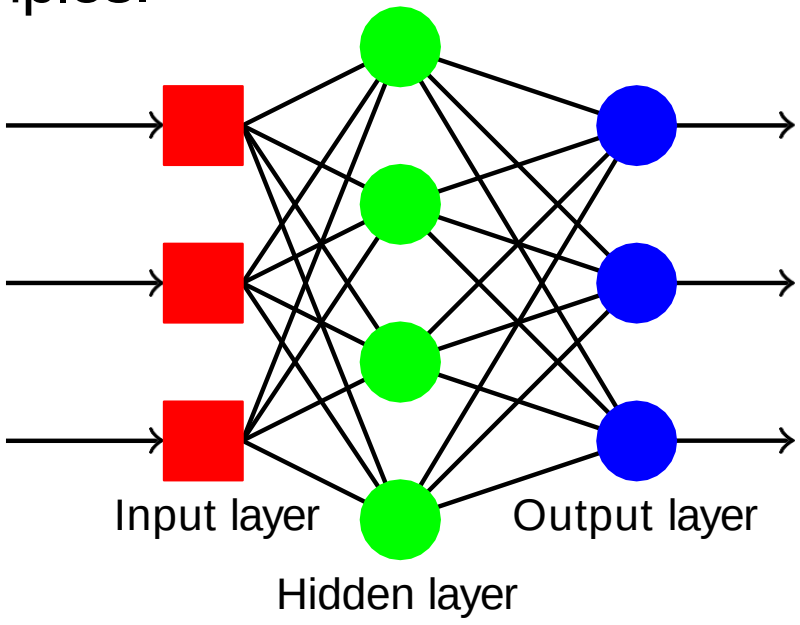


Figure : Partitioning of the input space by linear threshold-nodes in an MLP with two hidden layers and one output node in the output layer and examples of separable decision regions.
Font: http://www.neural-forecasting.com/mlp_neural_nets.htm.

MLPS: UNIVERSAL APPROXIMATORS

- As with single perceptron, finding the right weights is very hard!
- Solution technique: **learning!**
- As with perceptron, learning means adjusting the weights based on training examples.



SUPERVISED LEARNING

General idea

1. Send the MLP an input pattern, x, from the **training set**.
2. Get the output from the MLP, y.
3. Compare y with the "right answer", or target t, to get the **error quantity**.
4. Use the error quantity to modify the weights, so next time y will be closer to t.
5. Repeat with another x from the training set.

- Of course, when updating weights after seeing x, the network doesn't just change the way it deals with x, but other inputs too...
- Inputs it has not seen yet!
- The ability to deal accurately with unseen inputs is called **generalisation**.

LEARNING AND ERROR MINIMISATION

Recall: the **perceptron learning rule**

Try to minimise the difference between the actual and desired outputs:

$$w_i' = w_i + \alpha(t - y)x_i$$

We define an **error function** to represent such a difference over a set of inputs.

Typical error function: mean squared error (MSE)

For example, the mean squared error can be defined as:

$$E(\vec{w}) = \frac{1}{2N}\sum_{p=1}^{N}(t^p - o^p)^2$$

More than one output neuron

$$E(\vec{w}) = \frac{1}{2N}\sum_{p=1}^{N}\sum_{j=1}^{m}(t_j^p - o_j^p)^2$$

where

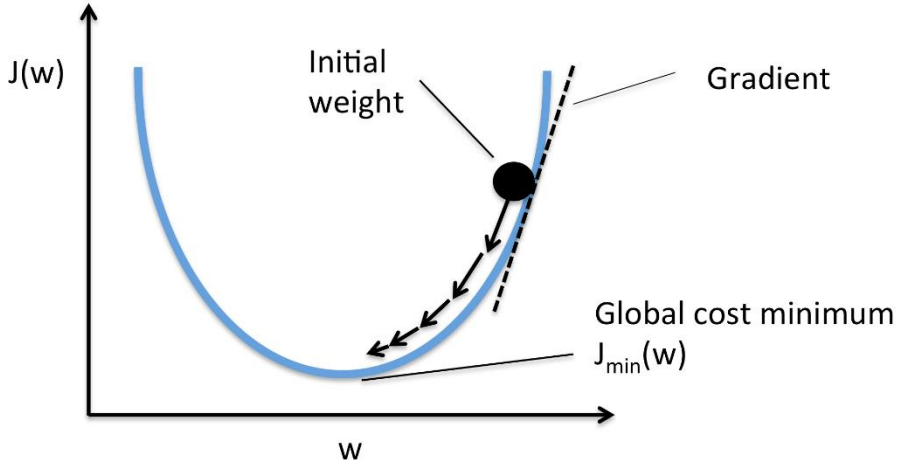
- N is the number of patterns,
- t^p is the target output for the pattern p,
- o^p is the output obtained for the pattern p, and
- The 2 makes little difference, but is inserted to make life easier later on!

This tells us how "good" the neural network is.

Learning aims to **minimise** the error $E(\vec{w})$ by adjusting the weights $\vec{w}$.

GRADIENT DESCENT

- One technique that can be used for minimising functions is **gradient descent**.
- Can we use this on our error function E?



We would like a learning rule that tells us how to update weights, like this:

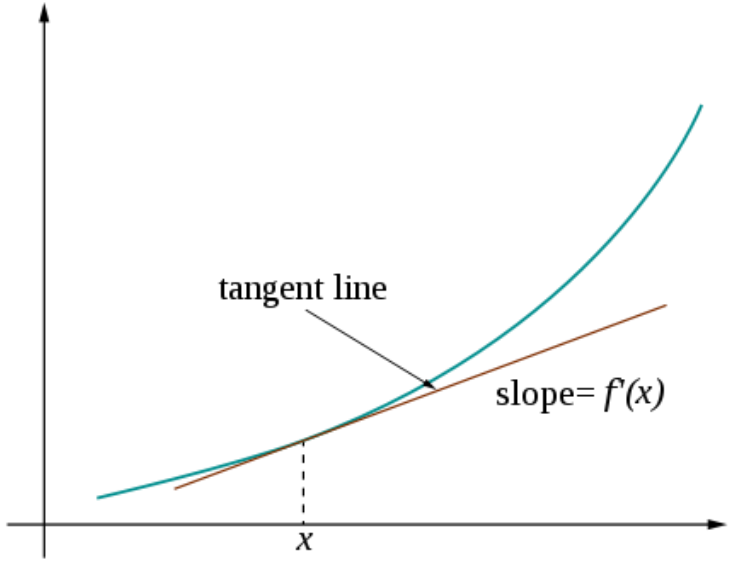
$$w'_{ij} = w_{ij} + \Delta w_{ij}$$

But what should Δw_{ij} be?

GRADIENT AND DERIVATIVES: THE IDEA

- The **derivative** is a measure of the rate of change of a function, as its input changes

- Consider a function $y = f(x)$.
- The derivative $\frac{dx}{dy}$ indicates how much y changes in response to changes in x.
- If x and y are real numbers, and if the graph of y is plotted against x, the derivative measures the **slope** or **gradient** of the line at each point, i.e., it describes the steepness or incline



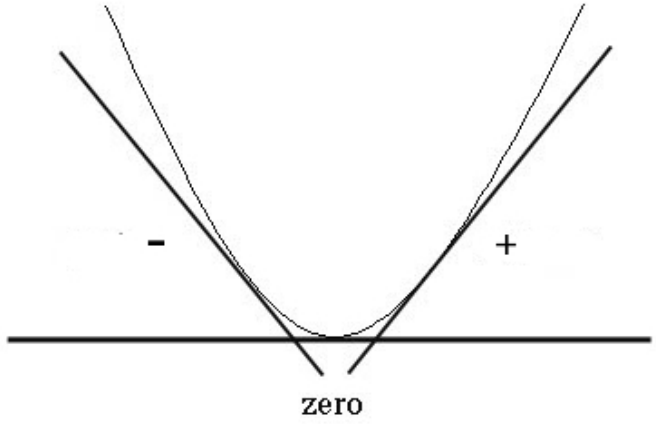
GRADIENT AND DERIVATIVES: THE IDEA

- $\frac{dx}{dy} > 0$ implies that y increases as x increases.

  If we want to find the minimum y, we should reduce x.

- $\frac{dx}{dy} < 0$ implies that y decreases as x increases.

  If we want to find the minimum y, we should increase x.



- $\frac{dx}{dy} = 0$ implies that we are at a minimum or maximum or a plateau.
- To get closer to the minimum: $x_{new} = x_{old} - \eta \frac{dy}{dx}$
- η indicates **how much** we would like to reduce or increase x and
- $\frac{dx}{dy}$ tells us the correct **direction** to go.

GRADIENT AND DERIVATIVES: THE IDEA

- **one weight** → simple derivative to adjust this weight.
- **several weights** to adjust → **partial derivatives**

- A partial derivative of a function of several variables is its derivative with respect to one of those variables, with the others held constant.

Example

If $y = f(x_1, x_2)$, then we can have $\frac{\partial y}{\partial x_1}$ and $\frac{\partial y}{\partial x_2}$.

In our learning rule case, if we can work out the partial derivatives, we can use this rule to update the weights:

Learning rule

$$w'_{ij} = w_{ij} + \Delta w_{ij}$$

where $\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$.

HOW TO CALCULATE DERIVATIVES

Recall the **mean squared error** function E, which we want to minimise:

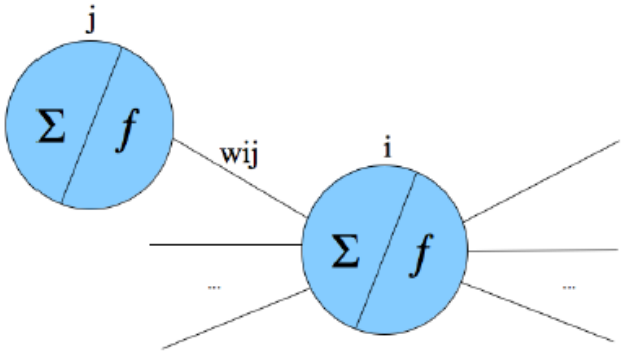
$$E(\vec{w}) = \frac{1}{2N}\sum_{p=1}^{N}(t^p - o^p)^2$$

- If we use a sigmoid activation function f, then the output of neuron i for pattern p is

$$o_i^p = f(u_i) = \frac{1}{1 + e^{-au_i}}$$

- where a is a predefined constant.
- And u_i is the result of the input function in neuron i:

$$u_i = \sum_j w_{ij}x_{ij}$$



For the pth pattern and the ith neuron, we use gradient descent on the error function:

$$\Delta w_{ij} = -\eta\frac{\partial E_p}{\partial w_{ij}} = \eta(t_i^p - o_i^p)f'(u_i)x_{ij}$$

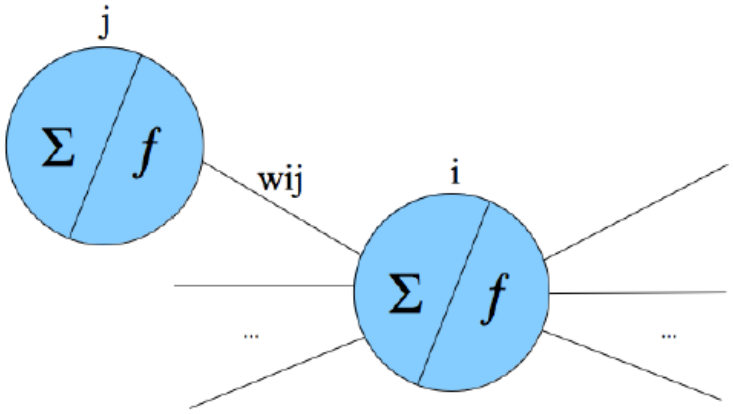
where $f'(u_i) = \frac{df}{du_i}$ is the derivative of f with respect to u_i.
If f is the sigmoid function, $f'(u_i) = af(u_i)(1 - f(u_i))$.

USING GRADIENT DESCENT TO MINIMISE THE ERROR

- So, we can update weights after processing each pattern, using this rule:

$$\Delta w_{ij} = \eta \ (t_i^p - o_i^p) f'(u_i) \ x_{ij}$$

$$\Delta w_{ij} = \eta \ \delta_i^p \ x_{ij}$$



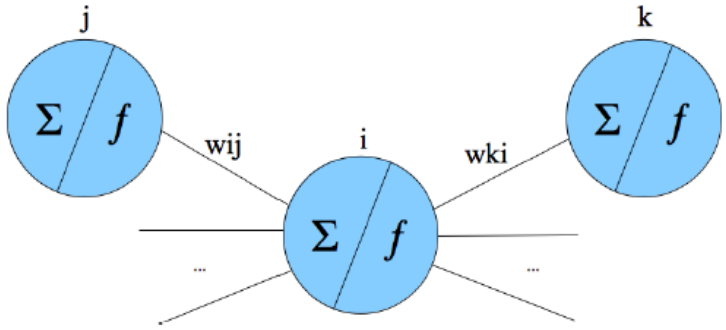
- This is known as the **generalised delta rule**.
- Note that we need to use the derivative of the activation function f.
- So, f **must** be **differentiable**
  - The threshold activation function is not continuous, thus not differentiable.
  - The sigmoid activation function has a derivative which is easy to calculate.

UPDATING OUTPUT VS HIDDEN NEURONS

- We can update output neurons using the generalised delta rule:

$$\Delta w_{ij} = \eta \ \delta_i^p \ x_{ij}$$

$$\delta_i^p = f'(u_i)(t_i^p - o_i^p)$$

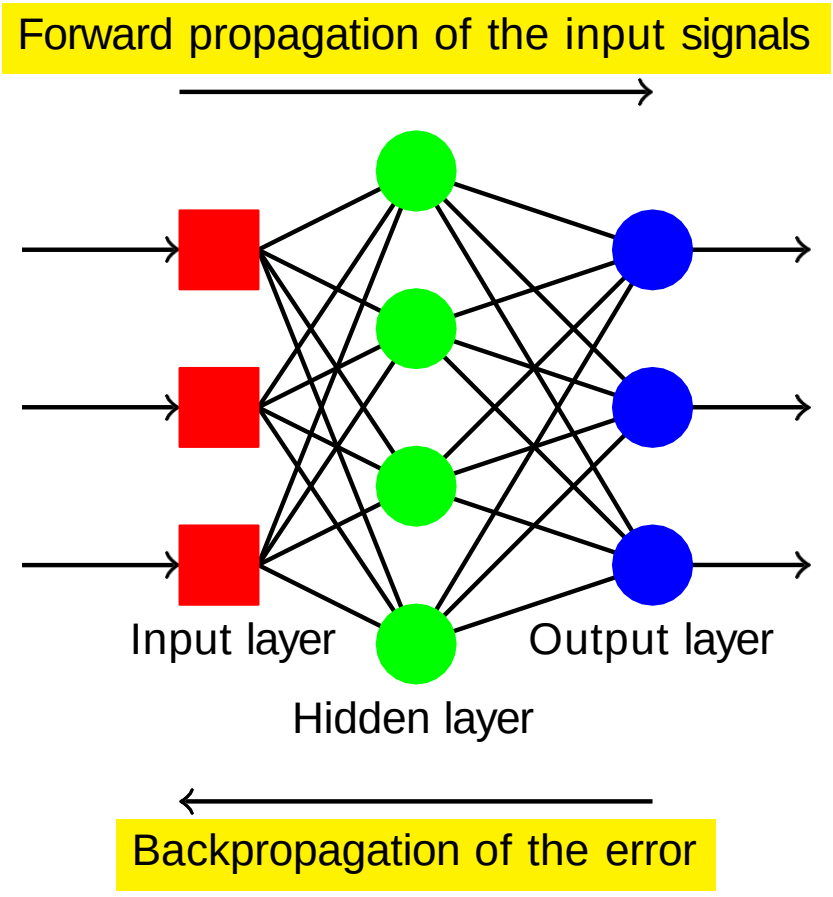


- This δ_i^p is only good for the output neurons, since it relies on target outputs.
- How about hidden neurons that don't have target outputs?

$$\delta_i^p = f'(u_i) \sum_k w_{ki} \delta_k$$

- This rule propagates error back from output neurons to hidden neurons.

BACKPROPAGATION

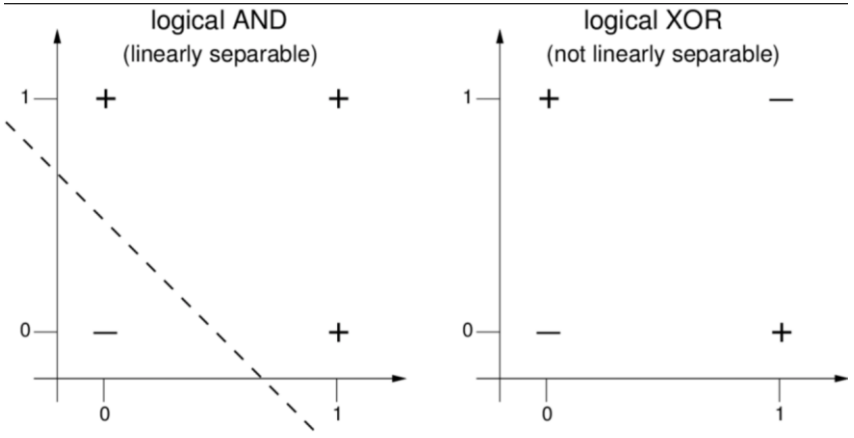


EXAMPLES

XOR PROBLEM

- The Rosenblatt's single layer perceptron was unable to solve problems that are linearly inseparable such as the XOR.

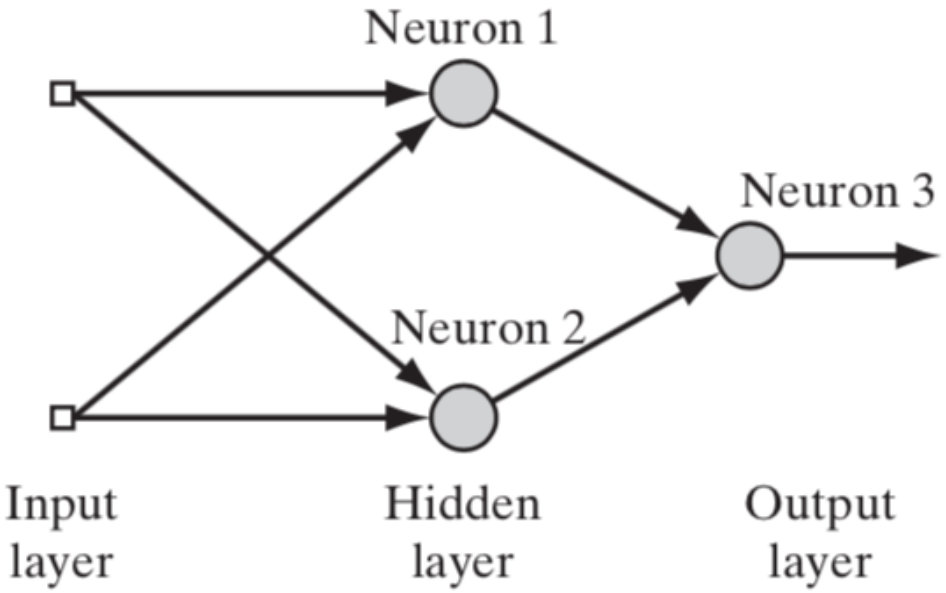
- The XOR can be solved using a single hidden layer
  - Each neuron is represented by a McColluch-Pitts model which uses a threshold as activation function.
  - Bits 0 and 1 are represented by 0 and +1 respectively (Binary Representation)

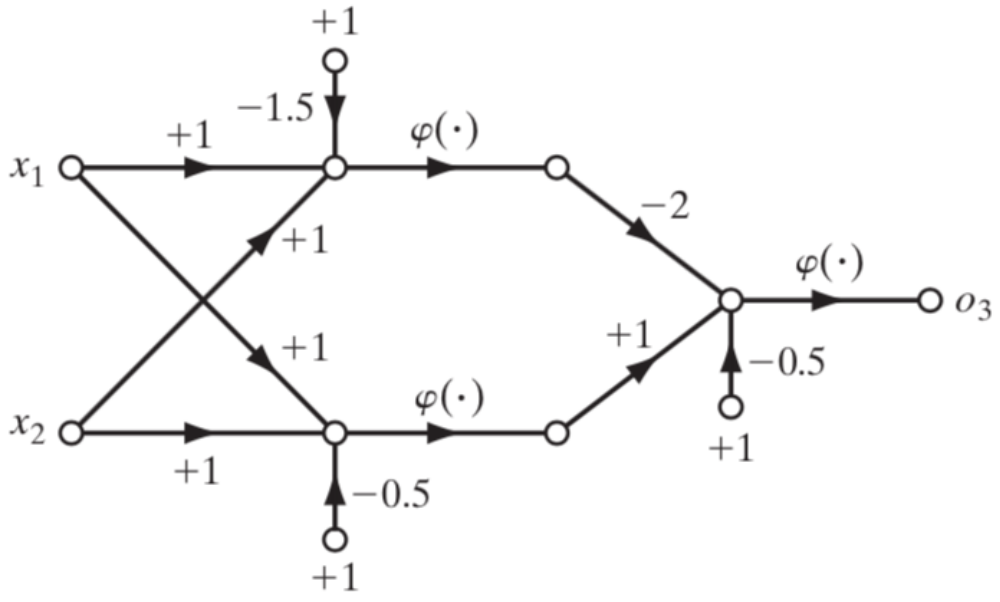


Input $_1$	Input $_2$	Output
1	1	1
1	0	1
0	1	1
0	0	0

Input $_1$	Input $_2$	Output
1	1	0
1	0	1
0	1	1
0	0	0

XOR PROBLEM





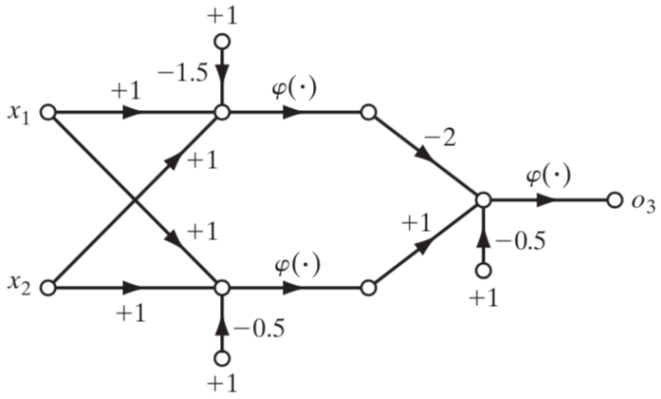
After Training we got the following weights:

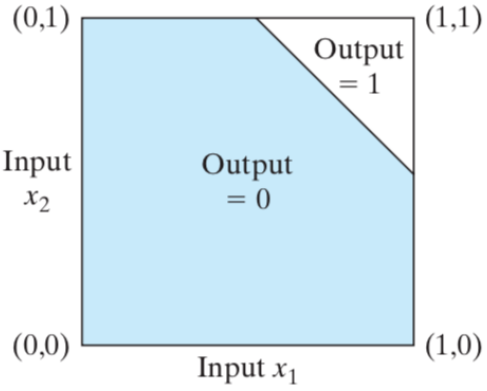
- Hidden Neuron 1
- $w_{11} = w_{12} = +1$
- $b_1 = -\frac{3}{2}$

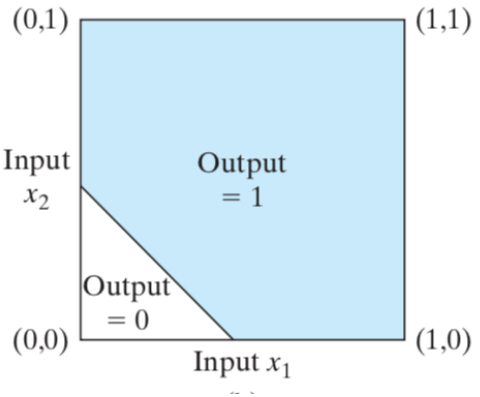
- Hidden Neuron 2
- $w_{21} = w_{22} = +1$
- $b_2 = -\frac{3}{2}$

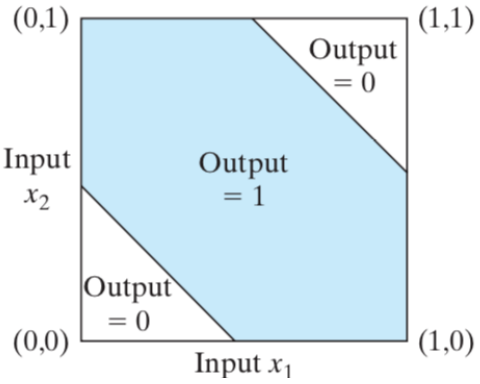
- Output Layer
- $w_{31} = -2\, w_{32} = +1$
- $b_3 = -\frac{1}{2}$

XOR PROBLEM



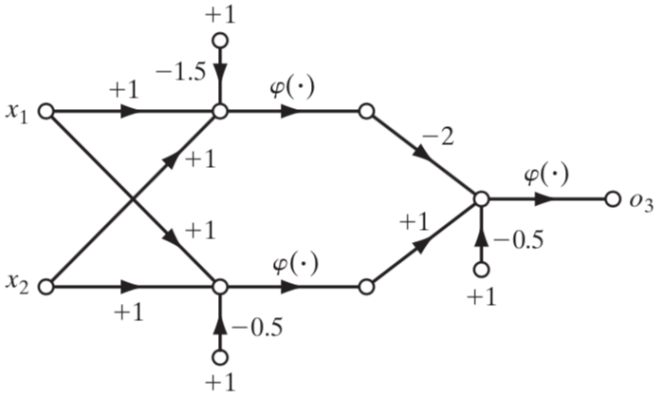


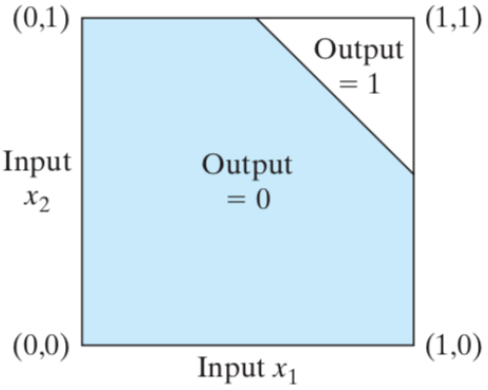


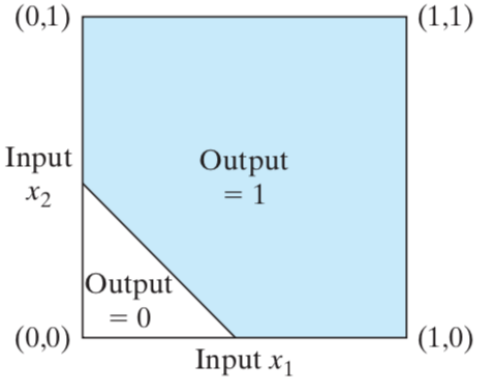


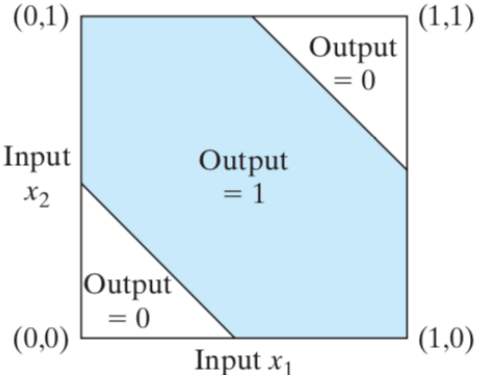
- The function of the output neuron is to construct a linear combination of the decision boundaries by the two hidden neurons.
  - Result is shown in the last image on the right (above)
- The bottom neuron is excitatory (positive connection) to the output.
- The top neuron is inhibitory (negative connection) to the output.
- When input pattern $(0,0)$ is introduced both hidden neurons are off and the output is off
- When input pattern $(1,1)$ is introduced both hidden neurons are on and the output is off

XOR PROBLEM





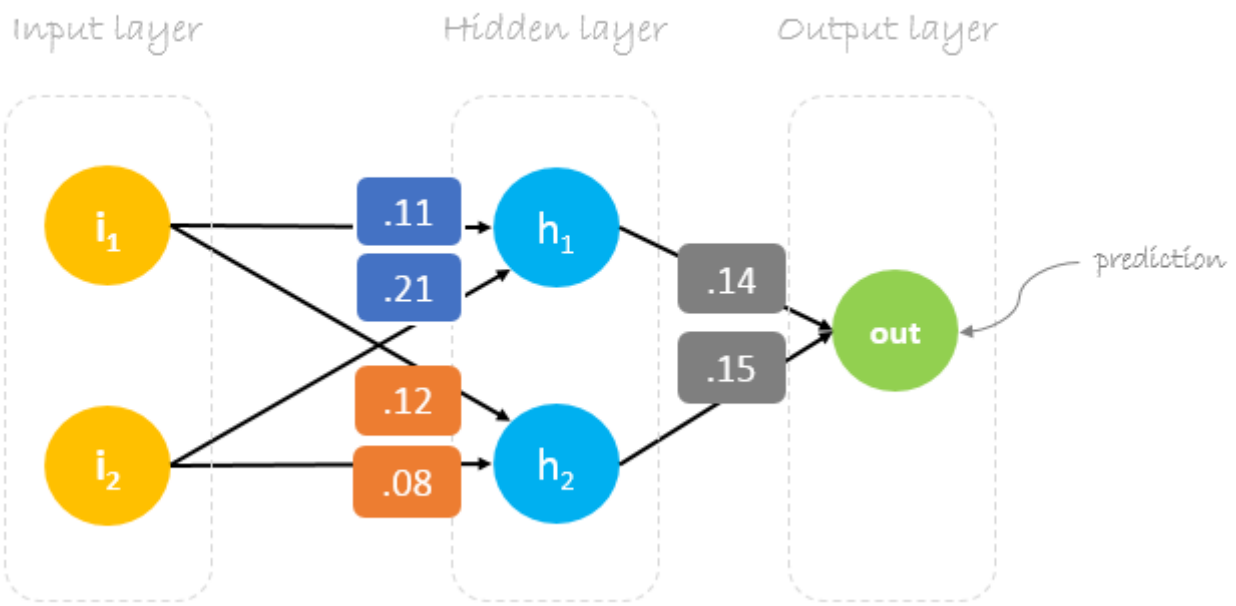




- When input pattern (0,0) is introduced both hidden neurons are off and the output is off
- When input pattern (1,1) is introduced the output will be remain off because the large inhibitory weight of the top node overwhelms that of the bottom node.
- The case of (1,0) and (0,1), the output is switched on because of the excitation of the positive weight of the bottom hidden neuron

EXAMPLE 2

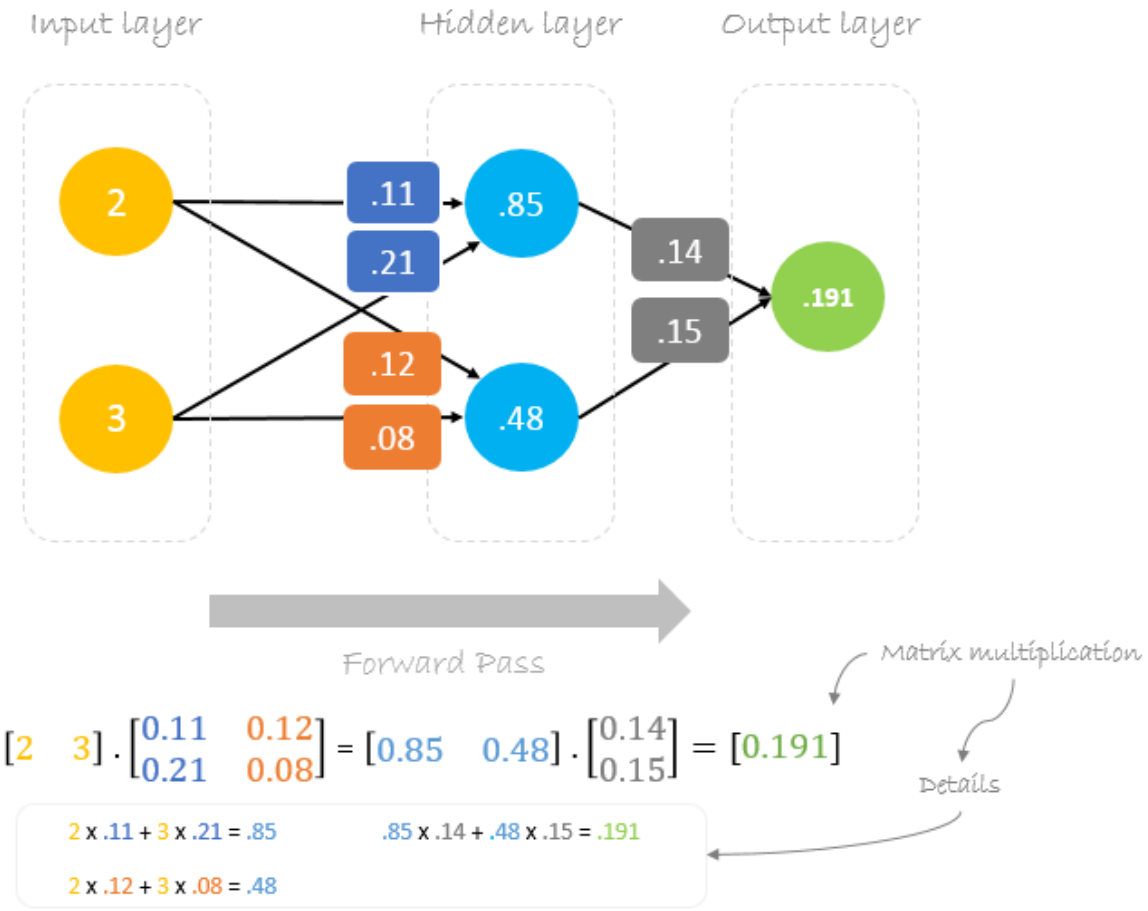
Network with initial weights



One sample from the dataset

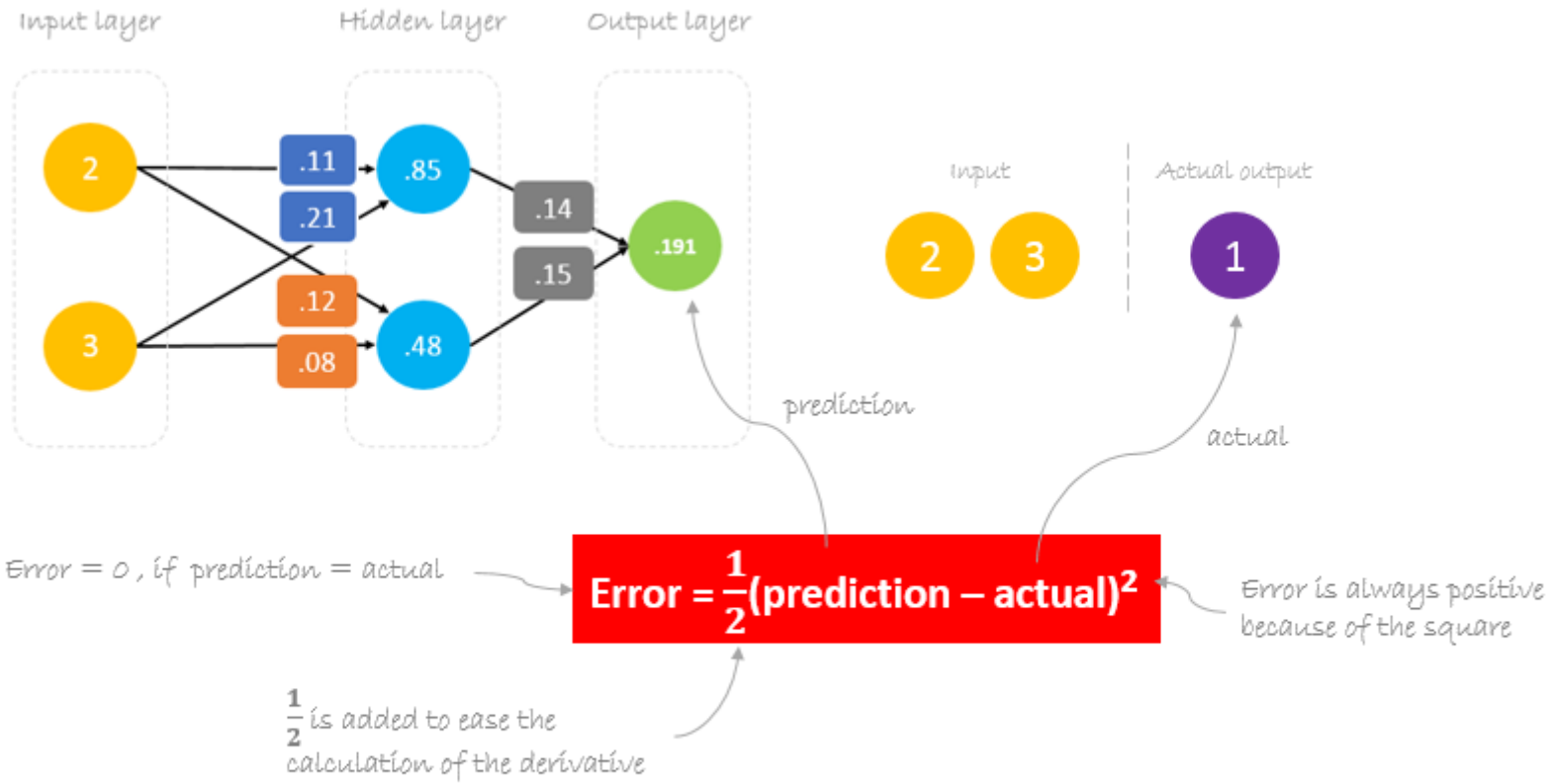
EXAMPLE 2

- Step 1: Forward pass

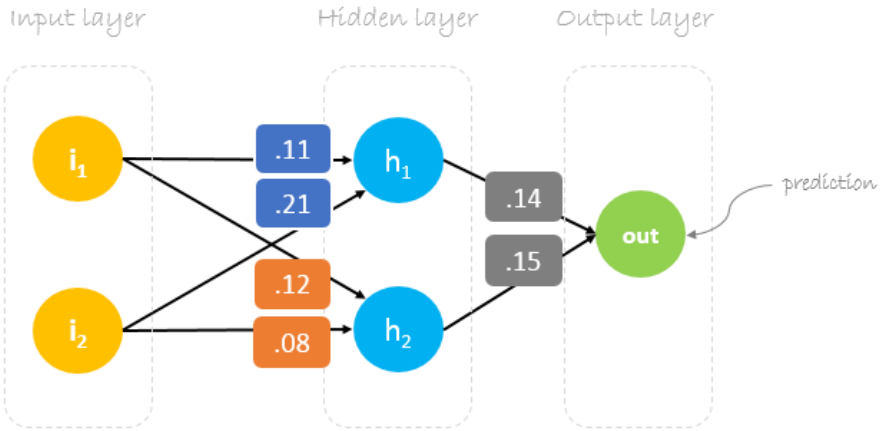


EXAMPLE 2

- Step 2: Calculate error



EXAMPLE 2



- Step 3: Reducing error by updating weights using BP

$$^{*}W_x = W_x - a\left(\frac{\partial Error}{\partial W_x}\right)$$

Old weight — Derivative of Error with respect to weight — New weight — Learning rate

$$prediction = out$$

$$prediction = (h_1)\, w_5 + (h_2)\, w_6$$

$h_1 = i_1 w_1 + i_2 w_2$

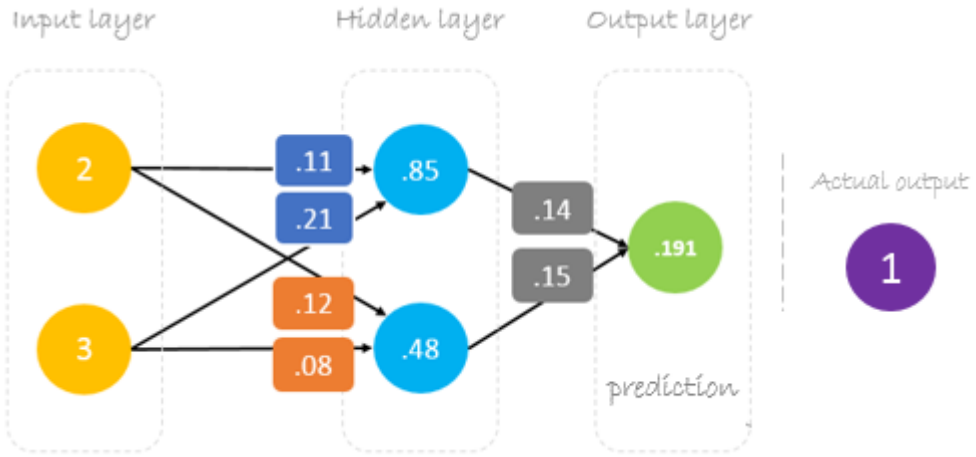
$h_2 = i_1 w_3 + i_2 w_4$

$$prediction = (i_1 w_1 + i_2 w_2)\, w_5 + (i_1 w_3 + i_2 w_4)\, w_6$$

to change ***prediction*** value, we need to change ***weights***

$$^{*}W_6 = W_6 - a\left(\frac{\partial Error}{\partial W_6}\right)$$

EXAMPLE 2



- Step 3: Reducing error by updating weights using BP (Output layer)

$$\frac{\partial Error}{\partial W_6} = \frac{\partial Error}{\partial prediction} * \frac{\partial prediction}{\partial W_6}$$ ← chain rule

Error = $\frac{1}{2}$(prediction – actual)^2

prediction = $(i_1 w_1 + i_2 w_2) w_5 + (i_1 w_3 + i_2 w_4) w_6$

$$\frac{\partial Error}{\partial W_6} = \frac{\frac{1}{2}(predictoin - actula)^2}{\partial prediciton} * \frac{\partial \left(i_1 w_1 + i_2 w_2\right) w_5 + \left(i_1 w_3 + i_2 w_4\right) w_6}{\partial W_6}$$

$$\frac{\partial Error}{\partial W_6} = 2 * \frac{1}{2}(\mathbf{predictoin - actula}) \frac{\partial (predictoin - actula)}{\partial prediciton} * (i_1 w_3 + i_2 w_4)$$ ← $h_2 = i_1 w_3 + i_2 w_4$

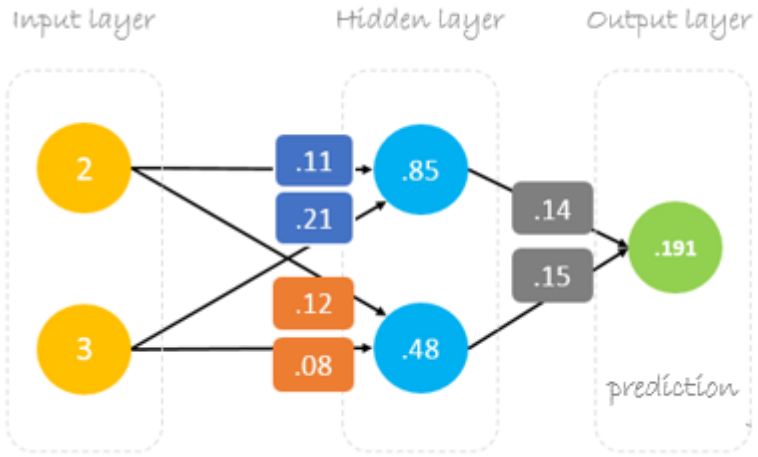
$$\frac{\partial Error}{\partial W_6} = (\mathbf{predictoin - actula}) * (h_2)$$

Δ = prediction – actual ← delta

$$\frac{\partial Error}{\partial W_6} = \Delta h_2$$

$$^*W_6 = W_6 - a\,\Delta h_2 \qquad ^*W_5 = W_5 - a\,\Delta h_1$$

EXAMPLE 2



- Step 3: Reducing error by updating weights using BP (hidden layer)

$$\frac{\partial Error}{\partial W_1} = \frac{\partial Error}{\partial prediction} * \frac{\partial prediction}{\partial h_1} * \frac{\partial h_1}{\partial W_1}$$ ← chain rule

$$\frac{\partial Error}{\partial W_1} = \frac{\partial \frac{1}{2}(predictoin - actula)^2}{\partial prediciton} * \frac{\partial (h_1)\, w_5 + (h_2)\, w_6}{\partial h_1} * \frac{\partial\, i_1 w_1 + i_2 w_2}{\partial w_1}$$

$$\frac{\partial Error}{\partial W_1} = 2 * \frac{1}{2}(\mathbf{predictoin - actula}) \frac{\partial (predictoin - actula)}{\partial prediciton} * (w_5) * (i_1)$$

$$\frac{\partial Error}{\partial W_1} = (\mathbf{predictoin - actula}) * (w_5 i_1)$$

$$\frac{\partial Error}{\partial W_1} = \Delta\, w_5 i_1$$

Error = $\frac{1}{2}$(prediction – actual)²

prediction = $(h_1)\, w_5 + (h_2)\, w_6$

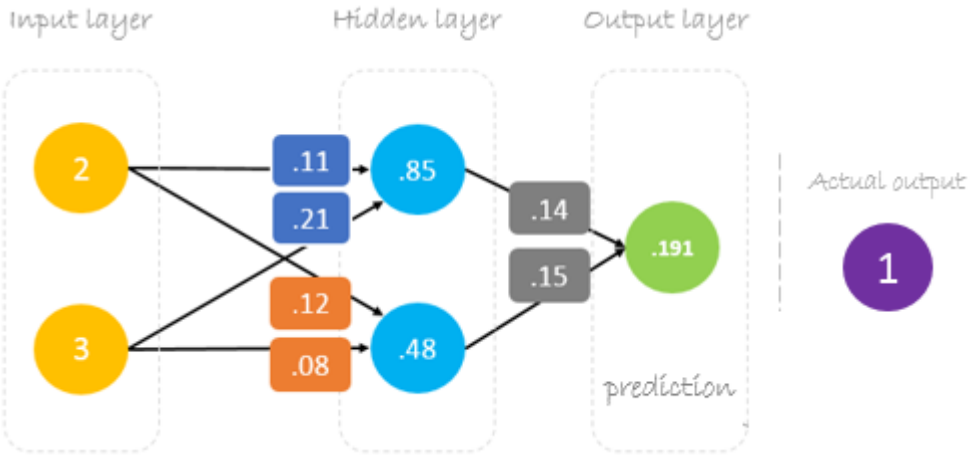
$h_1 = i_1 w_1 + i_2 w_2$

Δ = prediction – actual ← delta

updated weights →

$$^*w_6 = w_6 - a\,(h_2 \,.\, \Delta)$$
$$^*w_5 = w_5 - a\,(h_1 \,.\, \Delta)$$
$$^*w_4 = w_4 - a\,(i_2 \,.\, \Delta w_6)$$
$$^*w_3 = w_3 - a\,(i_1 \,.\, \Delta w_6)$$
$$^*w_2 = w_2 - a\,(i_2 \,.\, \Delta w_5)$$
$$^*w_1 = w_1 - a\,(i_1 \,.\, \Delta w_5)$$

EXAMPLE 2



- Step 3: Reducing error by updating weights using BP (hidden layer)

$\Delta = 0.191 - 1 = -0.809$ ← **Delta** = prediction - actual

$a = 0.05$ ← **Learning rate,** we smartly guess this number

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - 0.05(-0.809) \begin{bmatrix} 0.85 \\ 0.48 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - \begin{bmatrix} -0.034 \\ -0.019 \end{bmatrix} = \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix}$$

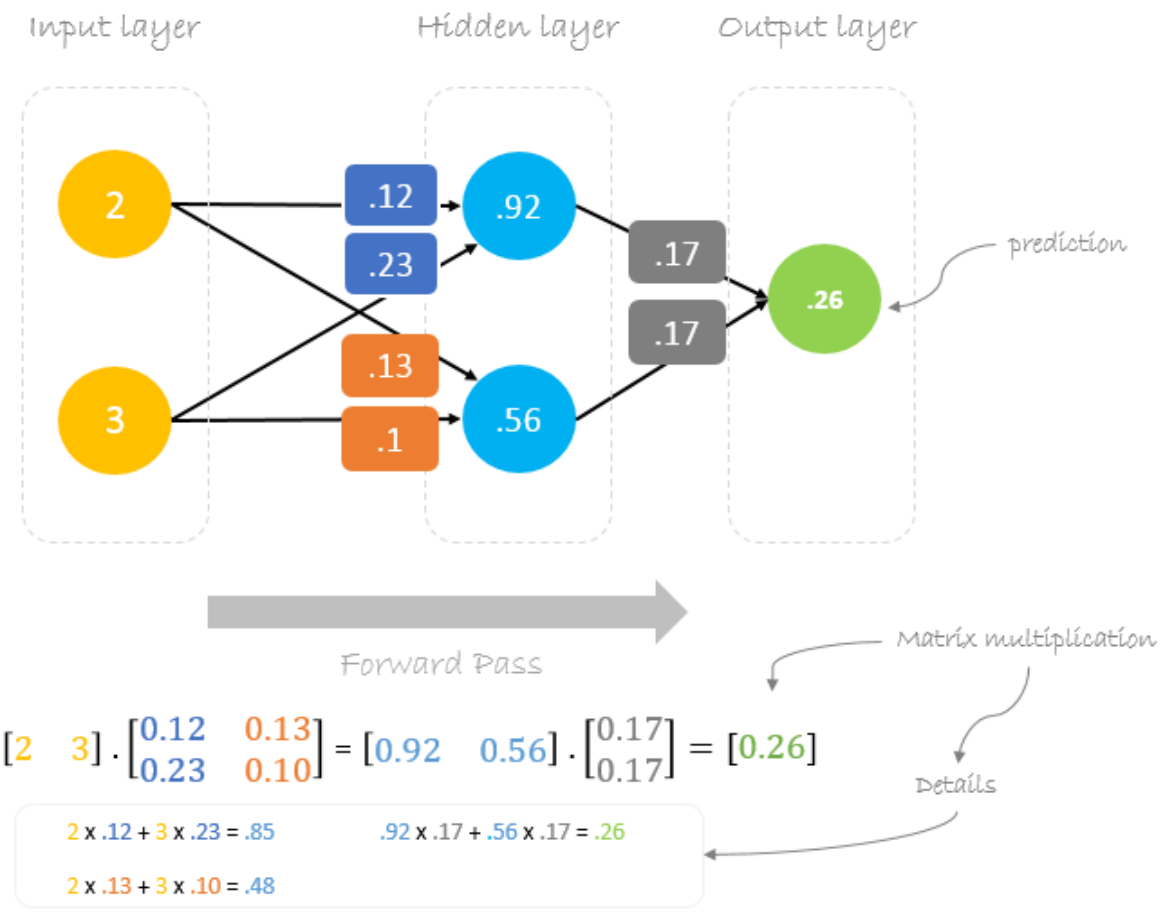
$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - 0.05(-0.809) \begin{bmatrix} 2 \\ 3 \end{bmatrix} . [0.14 \quad 0.15] = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - \begin{bmatrix} -0.011 & -0.012 \\ -0.017 & -0.018 \end{bmatrix} = \begin{bmatrix} .12 & .13 \\ .23 & .10 \end{bmatrix}$$

updated weights

$$^*w_6 = w_6 - a\,(h_2 \,.\, \Delta)$$
$$^*w_5 = w_5 - a\,(h_1 \,.\, \Delta)$$
$$^*w_4 = w_4 - a\,(i_2 \,.\, \Delta w_6)$$
$$^*w_3 = w_3 - a\,(i_1 \,.\, \Delta w_6)$$
$$^*w_2 = w_2 - a\,(i_2 \,.\, \Delta w_5)$$
$$^*w_1 = w_1 - a\,(i_1 \,.\, \Delta w_5)$$

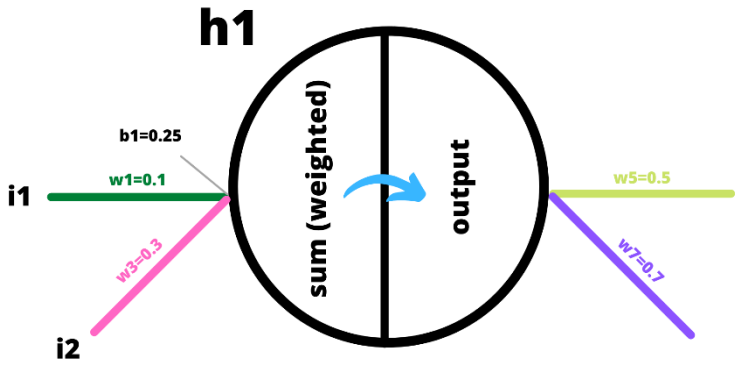
EXAMPLE 2

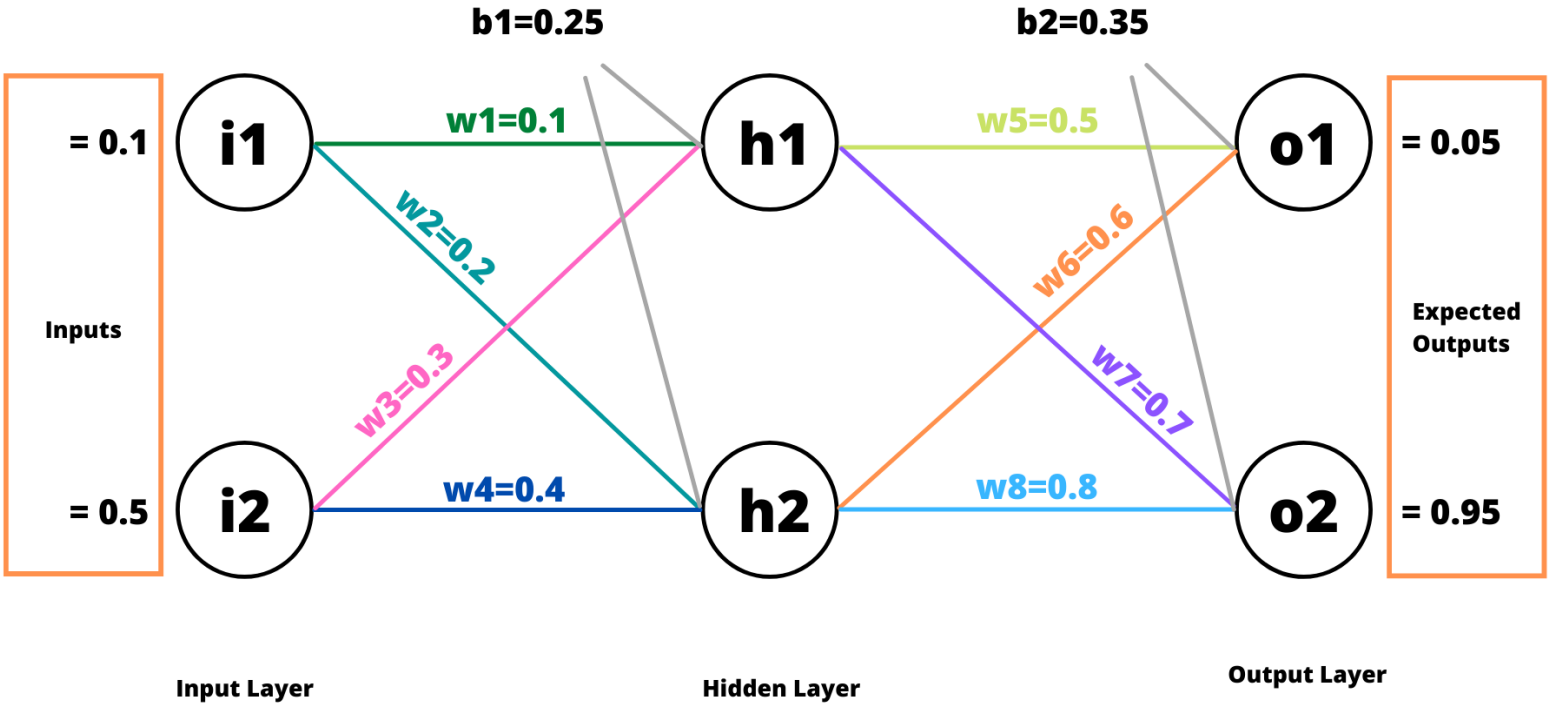
- Using the updated weights, what happens if we feed the same sample again ?
- Prediction is closer



EXAMPLE 3

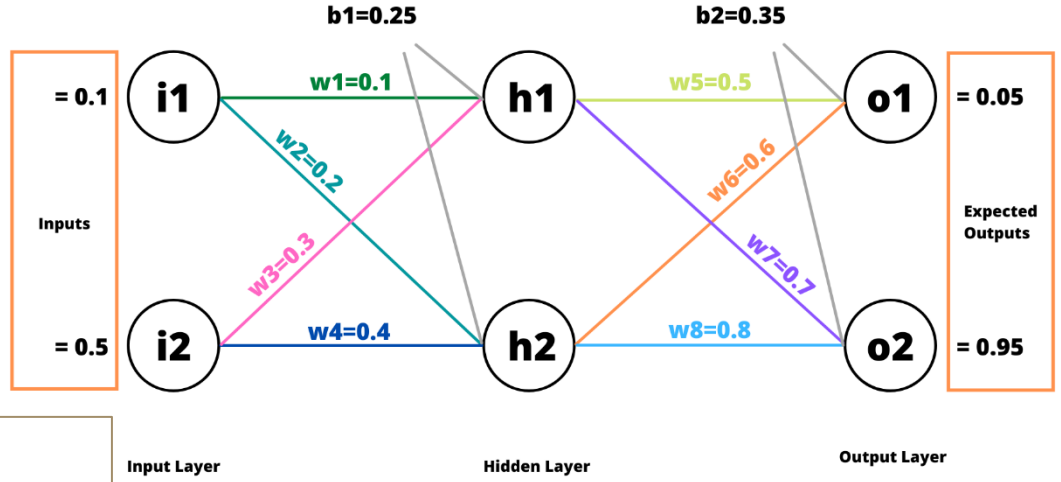
Here we use more than one output neuron
+ activation function not linear





EXAMPLE 3

Step 1: Forward pass



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

$$sum_{h1} = 0.1 * 0.1 + 0.5 * 0.3 + 0.25 = 0.41$$

$$output_{h1} = \frac{1}{1 + e^{-sum_{h1}}}$$

$$output_{h1} = \frac{1}{1 + e^{-0.41}} = 0.60108$$

$$sum_{h2} = i_1 * w_2 + i_2 * w_4 + b_1 = 0.47$$

$$output_{h2} = \frac{1}{1 + e^{-sum_{h2}}} = 0.61538$$

$$sum_{o1} = output_{h1} * w_5 + output_{h2} * w_6 + b_2 = 1.01977$$

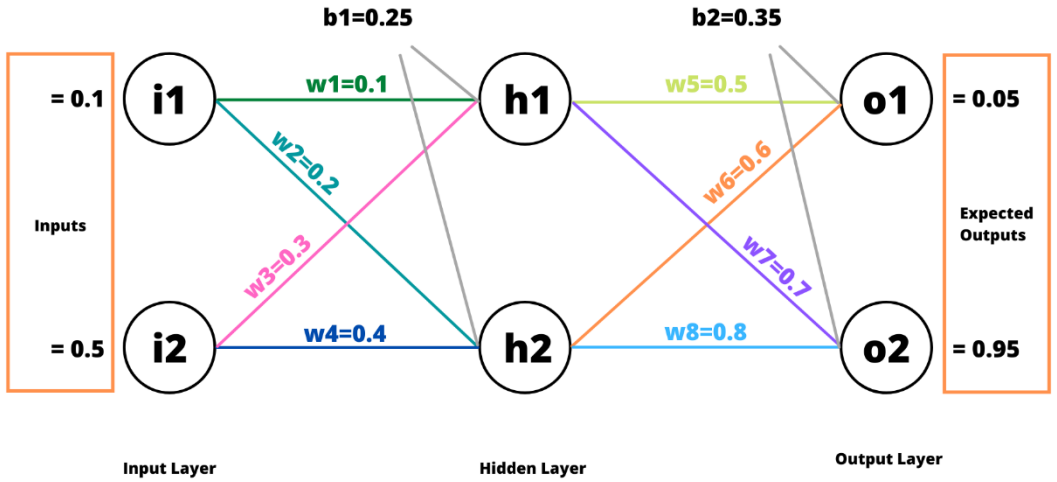
$$output_{o1} = \frac{1}{1 + e^{-sum_{o1}}} = 0.73492$$

$$sum_{o2} = output_{h1} * w_7 + output_{h2} * w_8 + b_2 = 1.26306$$

$$output_{o2} = \frac{1}{1 + e^{-sum_{o2}}} = 0.77955$$

EXAMPLE 3

Step 2: Error Calculation



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

$$E_1 = \frac{1}{2}(target_1 - output_{o1})^2$$

$$E_1 = \frac{1}{2}(0.05 - 0.73492)^2 = 0.23456$$

$$E_2 = \frac{1}{2}(target_2 - output_{o2})^2$$

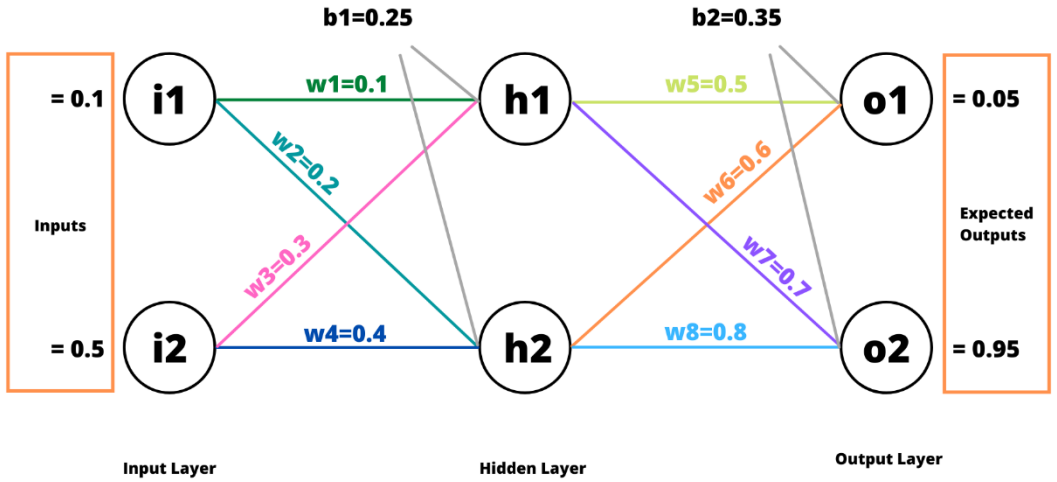
$$E_2 = \frac{1}{2}(0.95 - 0.77955)^2 = 0.01452$$

$$E_{total} = E_1 + E_2 = 0.24908$$

EXAMPLE 3

Step 3: Backpropagation

Weights at output layer: w5, w6, w7, w8



$$\frac{\partial E_{total}}{\partial w5} = \frac{\partial E_{total}}{\partial output_{o1}} * \frac{\partial output_{o1}}{\partial sum_{o1}} * \frac{\partial sum_{o1}}{\partial w5}$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\frac{\mathrm{d}}{\mathrm{dx}}\sigma(x) = \sigma(x)(1-\sigma(x))$$

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

$$E_{total} = \frac{1}{2}(target_1 - output_{o1})^2 + \frac{1}{2}(target_2 - output_{o2})^2$$

$$\frac{\partial output_{o1}}{\partial sum_{o1}} = output_{o1}(1 - output_{o1})$$

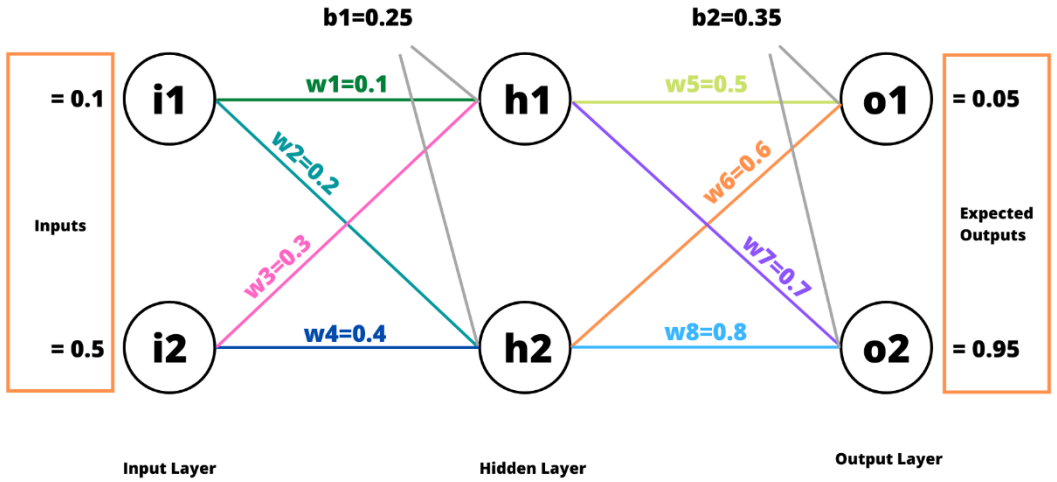
$$\frac{\partial E_{total}}{\partial output_{o1}} = 2 * \frac{1}{2} * (target_1 - output_{o1}) * -1$$

$$= output_{o1} - target_1$$

$$sum_{o1} = output_{h1} * w_5 + output_{h2} * w_6 + b_2$$

$$\frac{\partial sum_{o1}}{\partial w5} = output_{h1}$$

EXAMPLE 3



Step 3: Backpropagation

Weights at output layer: w5, w6, w7, w8

$$\frac{\partial E_{total}}{\partial w5} = \frac{\partial E_{total}}{\partial output_{o1}} * \frac{\partial output_{o1}}{\partial sum_{o1}} * \frac{\partial sum_{o1}}{\partial w5}$$

$$\frac{\partial E_{total}}{\partial w5} = [output_{o1} - target_1] * [output_{o1}(1 - output_{o1})] * [output_{h1}]$$

$$\frac{\partial E_{total}}{\partial w5} = 0.68492 * 0.19480 * 0.60108$$

$$\frac{\partial E_{total}}{\partial w5} = 0.08020$$

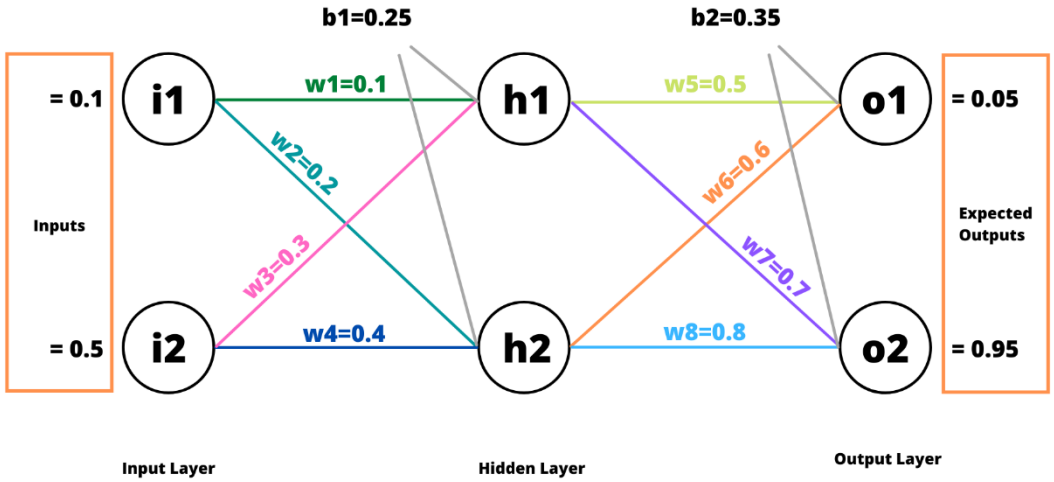
$new\_w_5 = w5 - n * \frac{\partial E_{total}}{\partial w5}$, where n is learning rate.

$$new\_w_5 = 0.5 - 0.6 * 0.08020$$

EXAMPLE 3

Step 3: Backpropagation

Weights at output layer: w5, w6, w7, w8



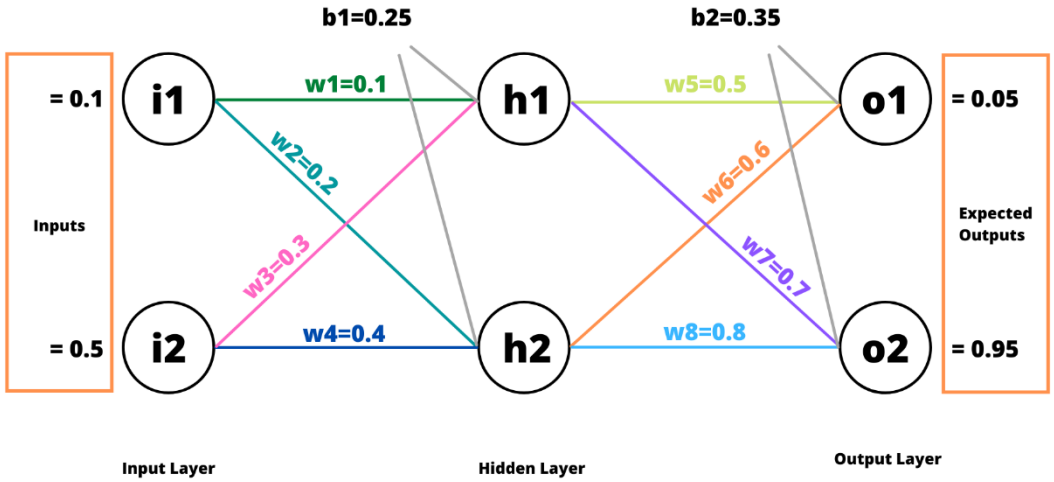
$$\frac{\partial E_{total}}{\partial w6} = \frac{\partial E_{total}}{\partial output_{o1}} * \frac{\partial output_{o1}}{\partial sum_{o1}} * \frac{\partial sum_{o1}}{\partial w6}$$

The first two components of this chain have already been calculated. The last component $\frac{\partial sum_{o1}}{\partial w6} = output_{h2}$.

$$\frac{\partial E_{total}}{\partial w6} = 0.68492 * 0.19480 * 0.61538 = 0.08211$$

$$new\_w_6 = w6 - n * \frac{\partial E_{total}}{\partial w6}$$

EXAMPLE 3



Step 3: Backpropagation

Weights at output layer: w5, w6, w7, w8

$$\frac{\partial E_{total}}{\partial w7} = \frac{\partial E_{total}}{\partial output_{o2}} * \frac{\partial output_{o2}}{\partial sum_{o2}} * \frac{\partial sum_{o2}}{\partial w7}$$

$$\frac{\partial E_{total}}{\partial output_{o2}} = output_{o2} - target_2$$

$$\frac{\partial output_{o2}}{\partial sum_{o2}} = output_{o2}(1 - output_{o2})$$

$$\frac{\partial sum_{o2}}{\partial w7} = output_{h1}$$

$$\frac{\partial E_{total}}{\partial w7} = [output_{o2} - target_2] * [output_{o2}(1 - output_{o2})] * [output_{h1}]$$

$$\frac{\partial E_{total}}{\partial w7} = -0.17044 * 0.17184 * 0.60108$$

$$\frac{\partial E_{total}}{\partial w7} = -0.01760$$

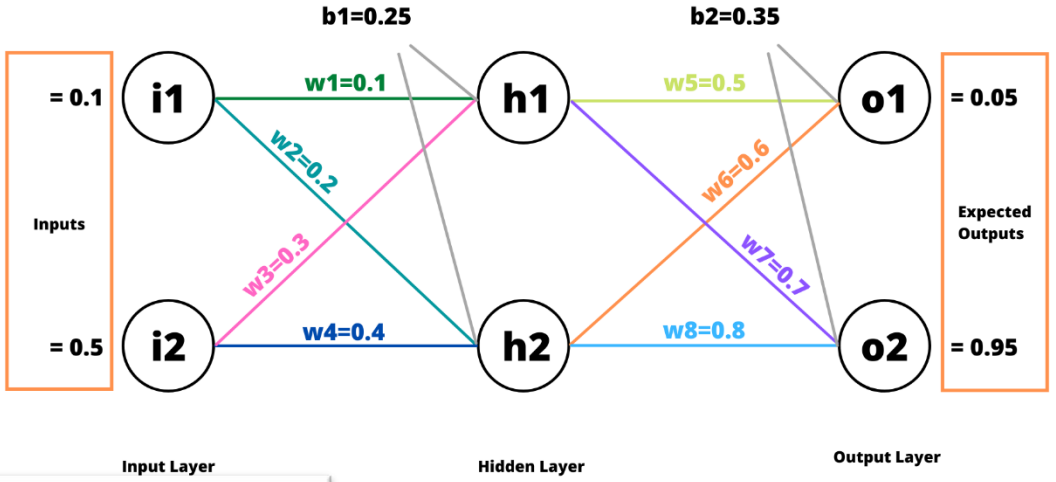
$$new\_w_7 = w7 - n * \frac{\partial E_{total}}{\partial w7}$$

$$new\_w_7 = 0.7 - 0.6 * -0.01760$$

$$new\_w_7 = 0.71056$$

$new\_w_8 = 0.81081$ (with $\frac{\partial E_{total}}{\partial w8} = -0.01802$).

EXAMPLE 3



$$\frac{\partial E_1}{\partial w1} = \frac{\partial E_1}{\partial output_{o1}} * \frac{\partial output_{o1}}{\partial sum_{o1}} * \frac{\partial sum_{o1}}{\partial output_{h1}} * \frac{\partial output_{h1}}{\partial sum_{h1}} * \frac{\partial sum_{h1}}{\partial w1}$$

$$\frac{\partial E_1}{\partial output_{o1}} = output_{o1} - target$$

$$sum_{o1} = output_{h1} * w_5 + output_{h2} * w_6 + b_2$$

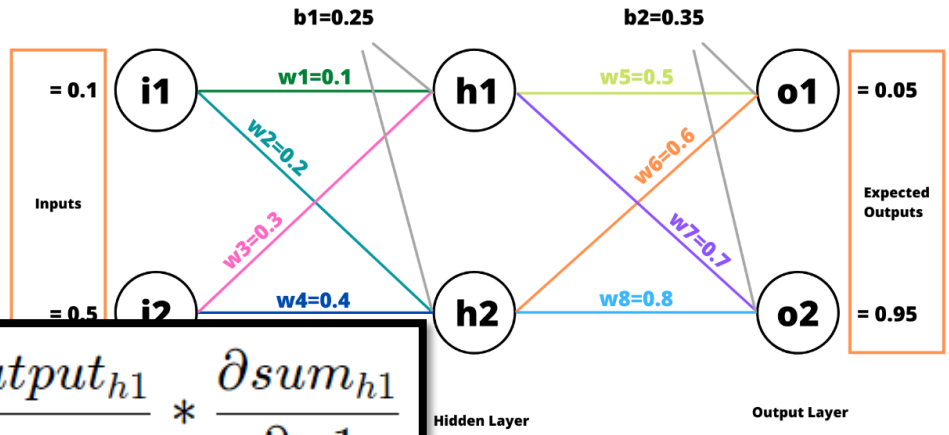
$$\frac{\partial sum_{o1}}{\partial output_{h1}} = w5$$

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

$$\frac{\partial sum_{h1}}{\partial w1} = i_1$$

$$\frac{\partial output_{h1}}{\partial sum_{h1}} = output_{h1} * (1 - output_{h1})$$

EXAMPLE 3



$$\frac{\partial E_1}{\partial w1} = \frac{\partial E_1}{\partial output_{o1}} * \frac{\partial output_{o1}}{\partial sum_{o1}} * \frac{\partial sum_{o1}}{\partial output_{h1}} * \frac{\partial output_{h1}}{\partial sum_{h1}} * \frac{\partial sum_{h1}}{\partial w1}$$

$$\frac{\partial E_1}{\partial w1} = 0.68492 * 0.19480 * 0.5 * 0.23978 * 0.1 = 0.00159$$

$$\frac{\partial E_2}{\partial w1} = \frac{\partial E_2}{\partial output_{o2}} * \frac{\partial output_{o2}}{\partial sum_{o2}} * \frac{\partial sum_{o2}}{\partial output_{h1}} * \frac{\partial output_{h1}}{\partial sum_{h1}} * \frac{\partial sum_{h1}}{\partial w1}$$

$$\frac{\partial E_2}{\partial output_{o2}} = output_{o2} - target_2$$

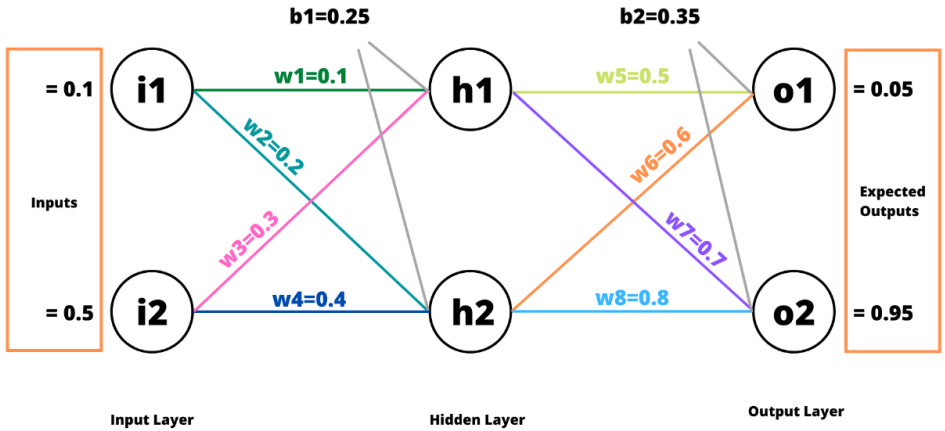
$$sum_{o2} = output_{h1} * w_7 + output_{h2} * w_8 + b_2$$

$$\frac{\partial sum_{o2}}{\partial output_{h1}} = w7$$

$$\frac{\partial E_2}{\partial w1} = \frac{\partial E_2}{\partial output_{o2}} * \frac{\partial output_{o2}}{\partial sum_{o2}} * \frac{\partial sum_{o2}}{\partial output_{h1}} * \frac{\partial output_{h1}}{\partial sum_{h1}} * \frac{\partial sum_{h1}}{\partial w1}$$

$$\frac{\partial E_2}{\partial w1} = -0.17044 * 0.17184 * 0.7 * 0.23978 * 0.1 = -0.00049$$

EXAMPLE 3



$$\frac{\partial E_{total}}{\partial w1} = 0.00159 + (-0.00049) = 0.00110.$$

$$new\_w_1 = w1 - n * \frac{\partial E_{total}}{\partial w1}$$

$$new\_w_1 = 0.1 - 0.6 * 0.00110$$

$$new\_w_1 = 0.09933$$

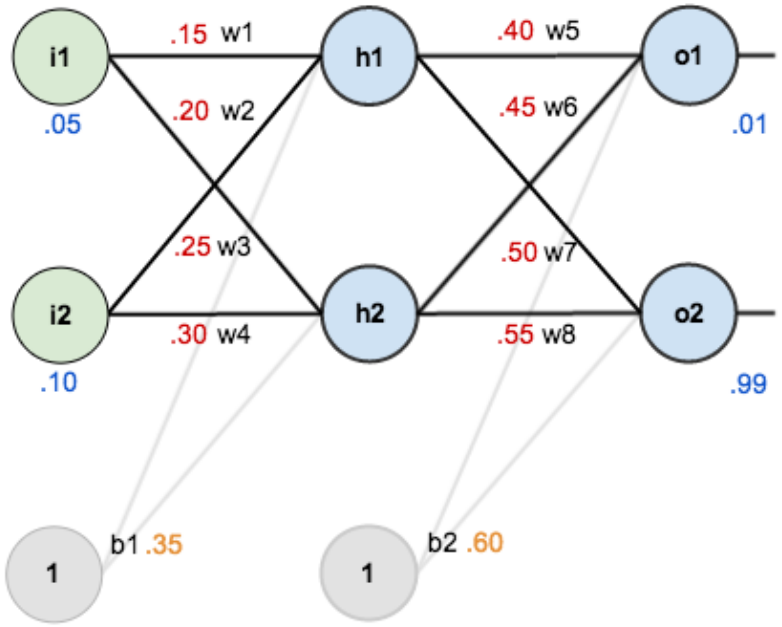
$$new\_w_2 = 0.19919$$

$$new\_w_3 = 0.29667$$

$$new\_w_4 = 0.39597$$

BACKPROPAGATION: EXAMPLE4

- Given the below one hidden layer neural network with i1 & i2 as training patterns, initial weights and the desired outputs o1 and o2
- Suppose using the logistic activation function



1.1 - FORWARD PASS

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

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

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

$$out_{h2} = 0.596884378$$

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

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

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

$$out_{o2} = 0.772928465$$

1.2 – TOTAL ERROR

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

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

$$E_{o2} = 0.023560026$$

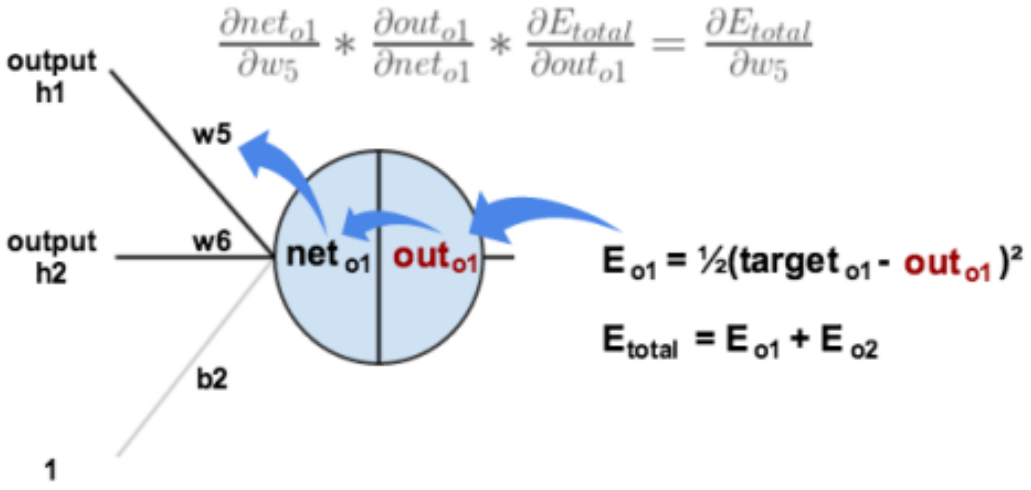
$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

2 - BACKWARD PASS

- Update the weights in reverse manner in order to get closer to the desired output.
  - For example how much updating w_5 affects the output → minimize error

$\frac{\partial E_{total}}{\partial w_5}$

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



2 – BACKWARD PASS

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

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

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

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

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$$

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

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

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

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

2 – BACKWARD PASS

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

η → is called learning rate

Some references use α while others use ϵ

$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

$$w_8^+ = 0.561370121$$

2 – BACKWARD PASS

$$\frac{\partial E_{total}}{\partial w_5} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1}) * out_{h1}$$

Alternatively, we have $\frac{\partial E_{total}}{\partial out_{o1}}$ and $\frac{\partial out_{o1}}{\partial net_{o1}}$ which can be written as $\frac{\partial E_{total}}{\partial net_{o1}}$, aka δ_{o1} (the Greek letter delta) aka the *node delta*. We can use this to rewrite the calculation above:

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

$$\delta_{o1} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1})$$

Therefore:

$$\frac{\partial E_{total}}{\partial w_5} = \delta_{o1} out_{h1}$$

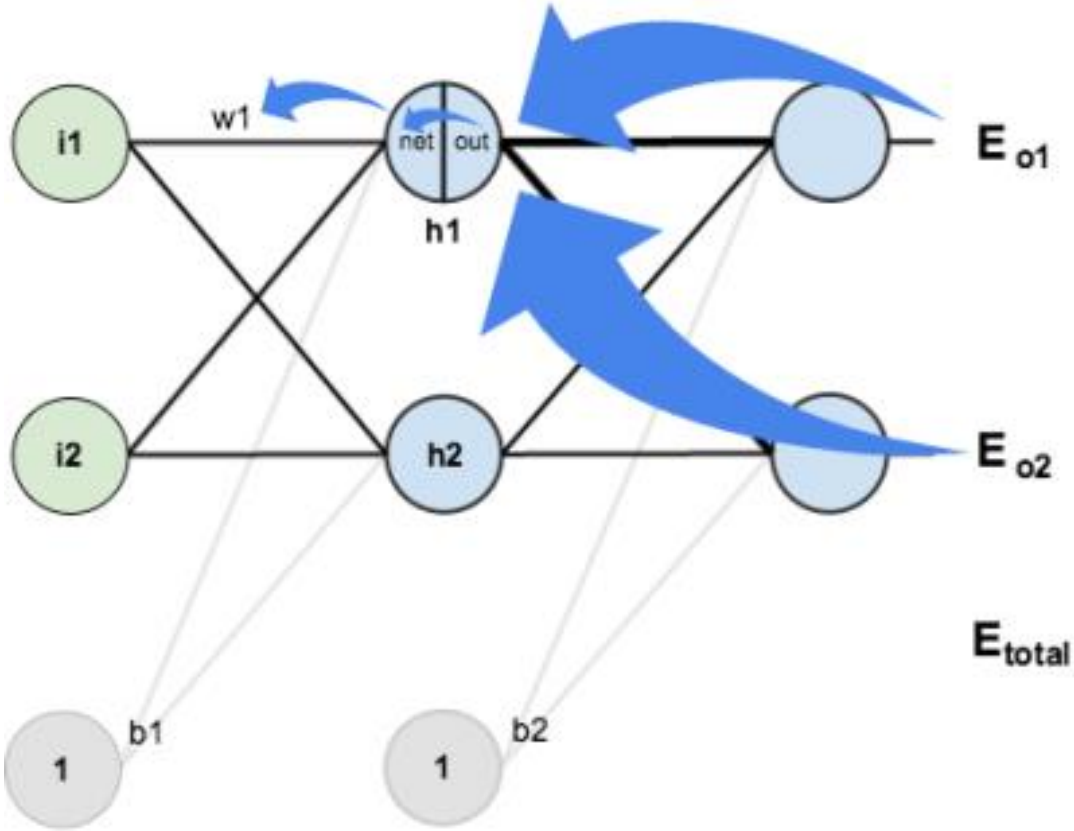
Some sources extract the negative sign from δ so it would be written as:

$$\frac{\partial E_{total}}{\partial w_5} = -\delta_{o1} out_{h1}$$

2 – BACKWARD PASS

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

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



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

2 — BACKWARD PASS

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

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:

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

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

2 — BACKWARD PASS

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

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

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

Therefore:

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}} = 0.055399425 + -0.019049119 = 0.036350306$$

Now that we have $\frac{\partial E_{total}}{\partial out_{h1}}$, we need to figure out $\frac{\partial out_{h1}}{\partial net_{h1}}$ and then $\frac{\partial net_{h1}}{\partial w}$ for each weight:

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

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

2 — BACKWARD PASS

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

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

Putting it all together:

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

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

You might also see this written as:

$$\frac{\partial E_{total}}{\partial w_1} = \left(\sum_o \frac{\partial E_{total}}{\partial out_o} * \frac{\partial out_o}{\partial net_o} * \frac{\partial net_o}{\partial out_{h1}}\right) * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = \left(\sum_o \delta_o * w_{ho}\right) * out_{h1}(1 - out_{h1}) * i_1$$

$$\frac{\partial E_{total}}{\partial w_1} = \delta_{h1} i_1$$

2 – BACKWARD PASS

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

Repeating this for w_2, w_3, and w_4

$$w_2^+ = 0.19956143$$

$$w_3^+ = 0.24975114$$

$$w_4^+ = 0.29950229$$

- Error was → 0.298371109
- After this first round of backpropagation → error is now down to 0.291027924.
- After 10,000 times → error is down to 0.000035085
- The output at this time is 0.015912196 (vs 0.01 target) and 0.984065734 (vs 0.99 target).