Machine Learning and Computational Intelligence
Lecture 10

Sanjeeb Prasad Panday, PhD
Associate Professor
Dept. of Electronics and Computer Engineering
Director (ICTC)
IOE, TU

Derivation of Back Propagation Algorithm

- To derive the equation for updating weights in back propagation algorithm, we use Stochastic gradient descent rule.
- Stochastic gradient descent involves iterating through the training examples one at a time, for each training example **d** descending the gradient of the error ***Ed*** with respect to this single example.
- In other words, for each training example **d** every weight ***wji*** is updated by adding to it Δw_{ij}.
- That is,

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

Where

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

Derivation of Back Propagation Algorithm

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

Where

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

- where ***Ed*** is the error on training example **d**, that is half the squared difference between the target output and the actual output over all output units in the network,

$$E_d(\vec{w}) \equiv \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$

- Here outputs is the set of output units in the network, $\mathbf{t_k}$ is the target value of unit k for training example **d**, and $\mathbf{o_k}$ is the output of unit k given training example **d**.

Derivation of Back Propagation Algorithm

Notation Used:

x_{ji} = the i[th] input to unit j

w_{ji} = the weight associated with the i[th] input to unit j

$net_j = \sum_i w_{ji} X_{ji}$ (the weighted sum of inputs for unit j)

o_j = the output computed by unit j

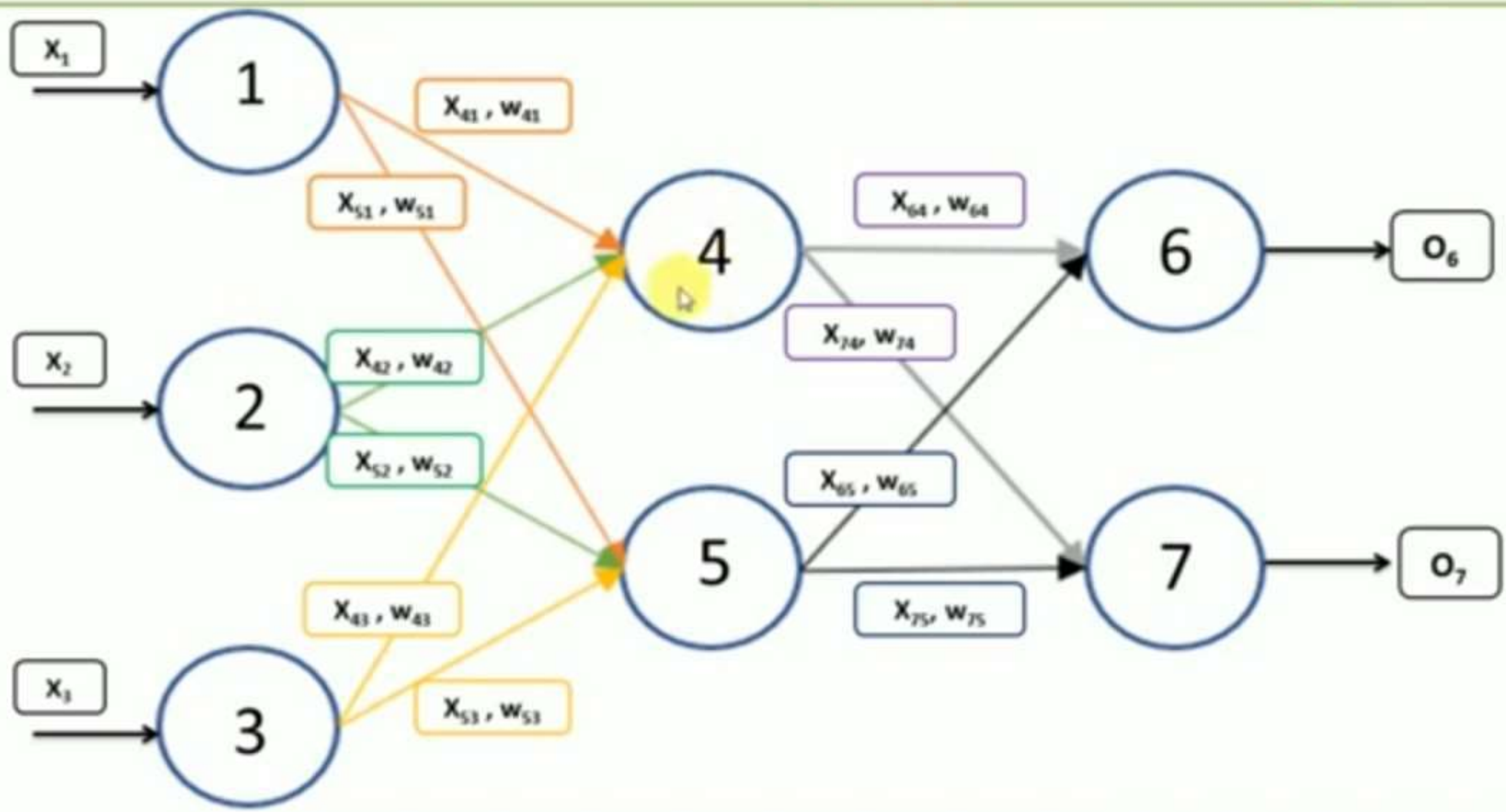
t_j = the target output for unit j

σ = the sigmoid function

outputs = the set of units in the final layer of the network

Downstream(j) = the set of units whose immediate inputs include the output of unit j

Derivation of Back Propagation Algorithm



Derivation of Back Propagation Algorithm

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

Where

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

- To begin, notice that weight ***wji*** can influence the rest of the network only through ***netj***.

  Therefore, we can use the chain rule to write,

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}}$$

$$= \frac{\partial E_d}{\partial net_j} x_{ji}$$

$$net_j = \sum_i w_{ji} x_{ji}$$

$$\frac{\partial net_j}{\partial w_{ji}} = x_{ji}$$

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial net_j} x_{ji}$$

- Our remaining task is to derive a convenient expression for $\frac{\partial E_d}{\partial net_j}$

Derivation of Back Propagation Algorithm

To derive a convenient expression for $\frac{\partial E_d}{\partial net_j}$

We consider two cases in turn:

- Case 1, where unit j is an output unit for the network, and
- Case 2, where unit j is an internal unit of the network.

The neuron

- The sigmoid equation is what is typically used as a transfer function between neurons. It is similar to the step function, but is continuous and differentiable.
- $$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

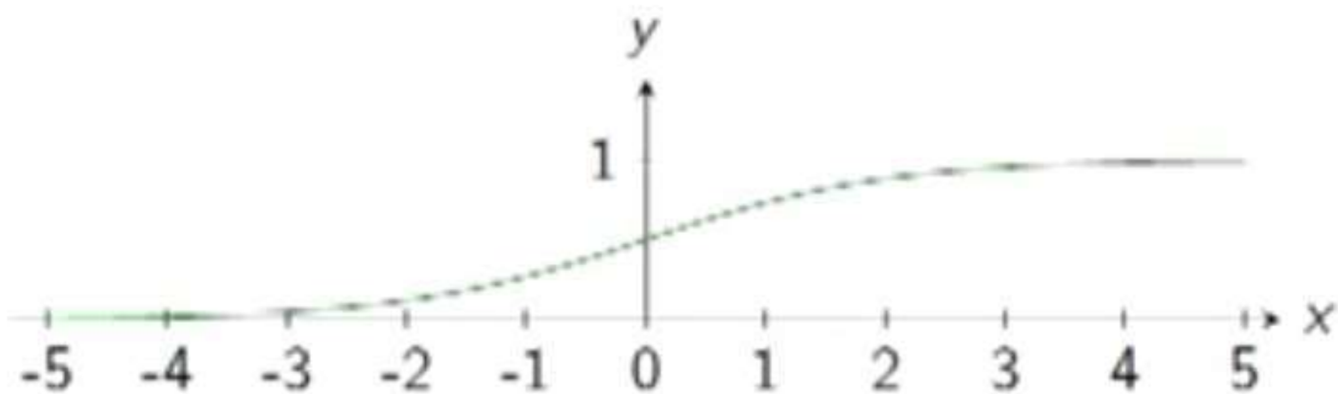


Figure: The Sigmoid Function

- One useful property of this transfer function is the simplicity of computing it's derivative. Let's do that now...

The derivative of the sigmoid transfer function

$$
\begin{aligned}
\frac{d}{dx}\sigma(x) &= \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right) \\
&= \frac{e^{-x}}{(1+e^{-x})^2} \\
&= \frac{(1+e^{-x})-1}{(1+e^{-x})^2} \\
&= \frac{1+e^{-x}}{(1+e^{-x})^2} - \left(\frac{1}{1+e^{-x}}\right)^2 \\
&= \sigma(x) - \sigma(x)^2 \\
\sigma' &= \sigma(1-\sigma)
\end{aligned}
$$

Derivation of Back Propagation Algorithm

Case 1: Training Rule for Output Unit Weights

- Just as ***wji*** can influence the rest of the network only through ***net$_j$***, ***net$_j$*** can influence the network only through ***oj.*** Therefore, we can invoke the chain rule again to write,

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}\sum_{k \in outputs}(t_k - o_k)^2$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}(t_j - o_j)^2$$

$$= \frac{1}{2}2(t_j - o_j)\frac{\partial(t_j - o_j)}{\partial o_j}$$

$$= -(t_j - o_j)$$

$$\frac{\partial o_j}{\partial(net_j)} = \frac{\partial\sigma(net_j)}{\partial(net_j)}$$

$$= \sigma(net_j)(1 - \sigma(net_j))$$

$$= o_j(1 - o_j)$$

$$\frac{\partial\sigma(x)}{\partial(x)} = \sigma(x)(1 - \sigma(x))$$

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j)\, o_j(1 - o_j)$$

Derivation of Back Propagation Algorithm

Case 1: Training Rule for Output Unit Weights

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j)\, o_j(1 - o_j)$$

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial net_j} x_{ji}$$

$$\Delta w_{ji} = \eta \underline{(t_j - o_j)\, o_j(1 - o_j)} x_{ji}$$

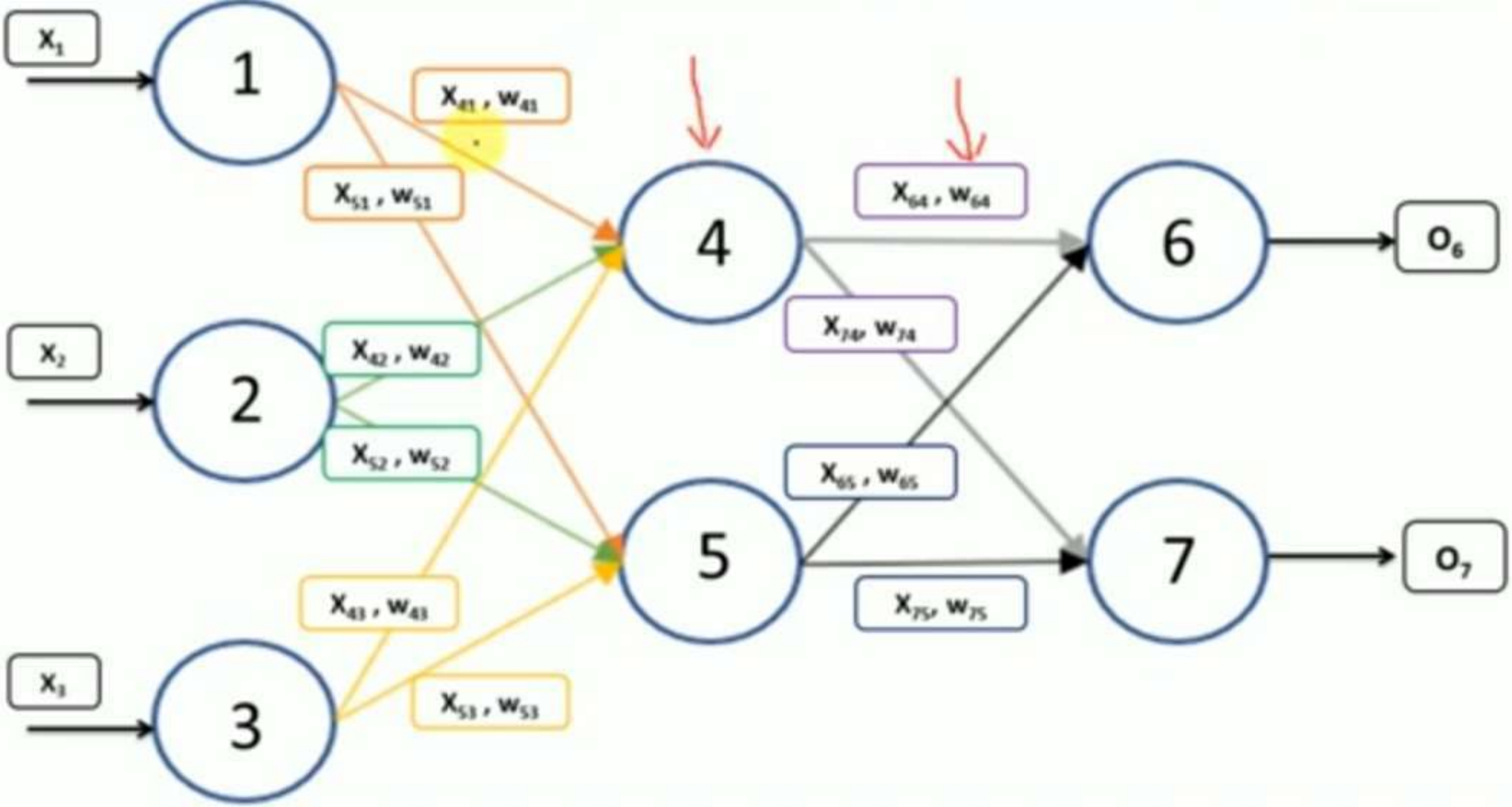
$$\delta_j = (t_j - o_j)\, o_j(1 - o_j)$$

$$\Delta w_{ji} = \eta\, \delta_j\, x_{ji}$$

Derivation of Back Propagation Algorithm

Case 2: Training Rule for Hidden Unit Weights

$$\begin{aligned}
\frac{\partial E_d}{\partial net_j} &= \sum_{k \in Downstream(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \, w_{kj} \frac{\partial o_j}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \, w_{kj} \, o_j(1 - o_j)
\end{aligned}$$



x1
1
x2
2
x3
3
X41 , W41
X51 , W51
X42 , W42
X52 , W52
X43 , W43
X53 , W53
4
5
X64 , W64
X74, W74
X65 , W65
X75, W75
6
7
O6
O7

Derivation of Back Propagation Algorithm

Case 2: Training Rule for Hidden Unit Weights

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in Downstream(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \, w_{kj} \frac{\partial o_j}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \, w_{kj} \, o_j(1 - o_j)$$

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j)\, o_j(1 - o_j)$$

$$\delta_j = (t_j - o_j)\, o_j(1 - o_j)$$

$$\frac{\partial \, net_k}{\partial o_j} = \frac{\partial x_{kj} w_{kj}}{\partial o_j} = \frac{\partial o_j w_{kj}}{\partial o_j}$$

$$\frac{\partial o_j}{\partial (net_j)} = \frac{\partial \sigma(net_j)}{\partial (net_j)}$$

$$= \sigma(net_j)(1 - \sigma(net_j))$$

$$= o_j(1 - o_j)$$

Derivation of Back Propagation Algorithm

Case 2: Training Rule for Hidden Unit Weights

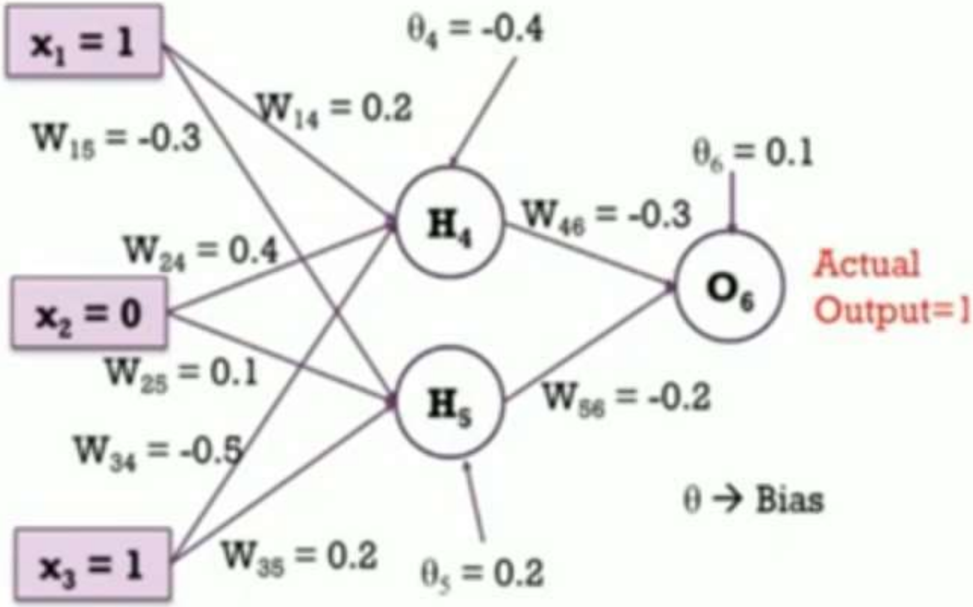
$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial net_j} x_{ji}$$

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in Downstream(j)} -\delta_k \, w_{kj} \, o_j(1 - o_j)$$

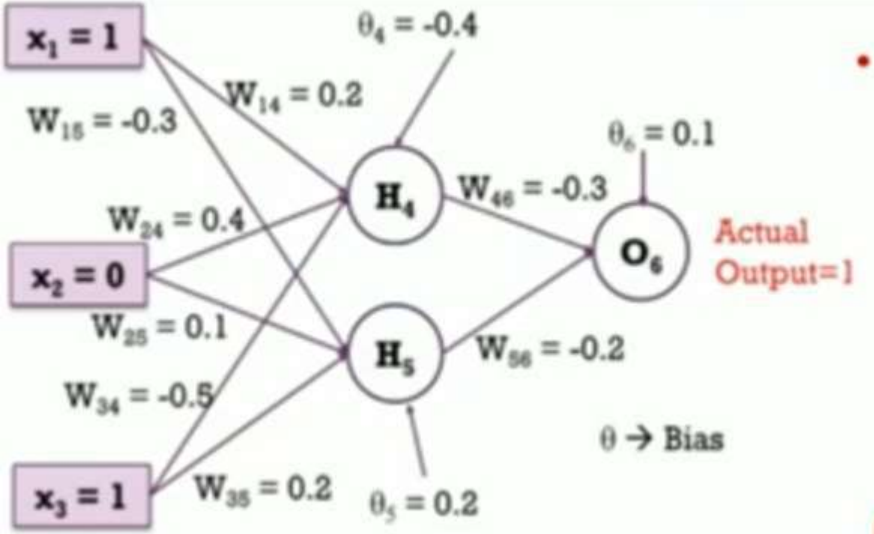
$$\Delta w_{ji} = \eta \, o_j(1 - o_j) \sum_{k \in Downstream(j)} \delta_k \, w_{kj} \, x_{ji}$$

$$\Delta w_{ji} = \eta \, \delta_j \, x_{ji}$$

$$\delta_j = o_j(1 - o_j) \sum_{k \in Downstream(j)} \delta_k \, w_{kj}$$



Assume that the neurons have a sigmoid activation function, perform a forward pass and a backward pass on the network. Assume that the actual output of y is 1 and learning rate is 0.9. Perform another forward pass.



Error $= y_{target} - y_6 = 0.526$

- Forward Pass: Compute output for y4, y5 and y6.

$$a_j = \sum_j (w_{i,j} * x_i) \qquad yj = F(aj) = \frac{1}{1 + e^{-a_j}}$$

$$a_4 = (w_{14} * x_1) + (w_{24} * x_2) + (w_{34} * x_3) + \theta_4$$
$$= (0.2 * 1) + (0.4 * 0) + (-0.5 * 1) + (-0.4) = -0.7$$
$$O(H_4) = y_4 = f(a_4) = 1/(1 + e^{0.7}) = 0.332$$

$$a_5 = (w_{15} * x_1) + (w_{25} * x_2) + (w_{35} * x_3) + \theta_5$$
$$= (-0.3 * 1) + (0.1 * 0) + (0.2 * 1) + (0.2) = 0.1$$
$$O(H_5) = y_5 = f(a_5) = 1/(1 + e^{-0.1}) = 0.525$$

$$a_6 = (w_{46} * H_4) + (w_{56} * H_5) + \theta_6$$
$$= (-0.3 * 0.332) + (-0.2 * 0.525) + 0.1 = -0.105$$
$$O(O_6) = y_6 = f(a_6) = 1/(1 + e^{0.105}) = 0.474$$

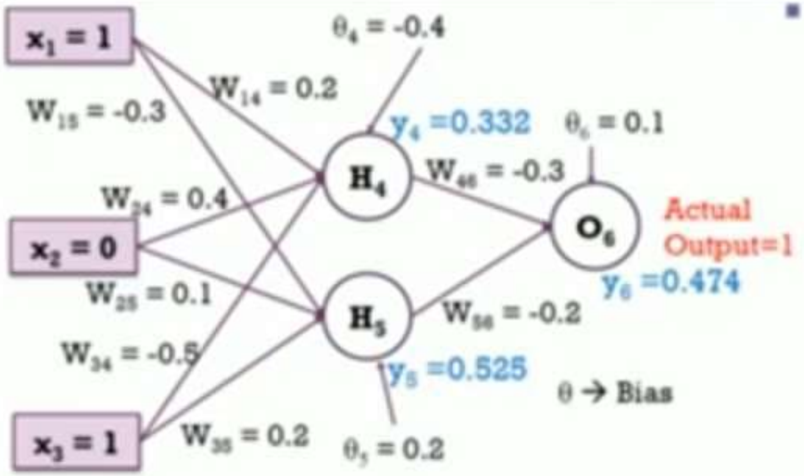
- Each weight changed by:

$$\Delta w_{ji} = \eta \delta_j o_i$$

$$\delta_j = o_j(1 - o_j)(t_j - o_j) \qquad \text{if } j \text{ is an output unit}$$

$$\delta_j = o_j(1 - o_j)\sum_k \delta_k w_{kj} \qquad \text{if } j \text{ is a hidden unit}$$

- where η is a constant called the learning rate
- tj is the correct teacher output for unit j
- δj is the error measure for unit j



- Backward Pass: Compute δ4, δ5 and δ6.

For output unit:

$$\delta_6 = y_6(1-y_6)\,(y_{target} - y_6)$$
$$= 0.474*(1-0.474)*(1-0.474) = 0.1311$$

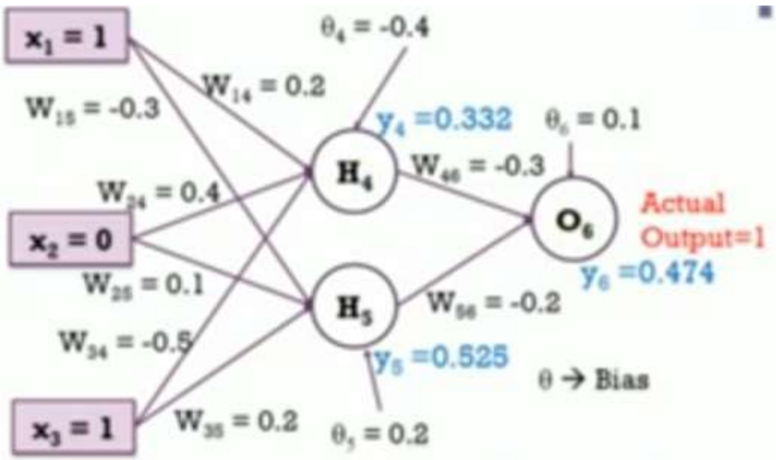
- Each weight changed by:

$$\Delta w_{ji} = \eta \delta_j o_i$$

$$\delta_j = o_j(1 - o_j)(t_j - o_j) \qquad \text{if } j \text{ is an output unit}$$

$$\delta_j = o_j(1 - o_j)\sum_k \delta_k w_{kj} \qquad \text{if } j \text{ is a hidden unit}$$

- where η is a constant called the learning rate
- tj is the correct teacher output for unit j
- δj is the error measure for unit j



- Backward Pass: Compute δ4, δ5 and δ6.

For output unit:

$\delta_6 = y_6(1-y_6)\,(y_{target} - y_6)$
$= 0.474*(1-0.474)*(1-0.474) = 0.1311$

For hidden unit:

$\delta_5 = y_5(1-y_5)\, w_{56} * \delta_6$
$= 0.525*(1 - 0.525)*(-0.2 * 0.1311) = -0.0065$

$= y_4(1-y_4)\, w_{46} * \delta_6$
$= 0.332*(1 - 0.332)*(-0.3 * 0.1331) = -0.0087$

Compute new weights

$$\Delta w_{ji} = \eta \delta_j o_i$$

$\Delta w_{46} = \eta\, \delta_6\, y_4 = 0.9 * 0.1311 * 0.332 = 0.03917$
$w_{46}\text{ (new)} = \Delta w_{46} + w_{46}\text{(old)} = 0.03917 + (-0.3) = -0.261$

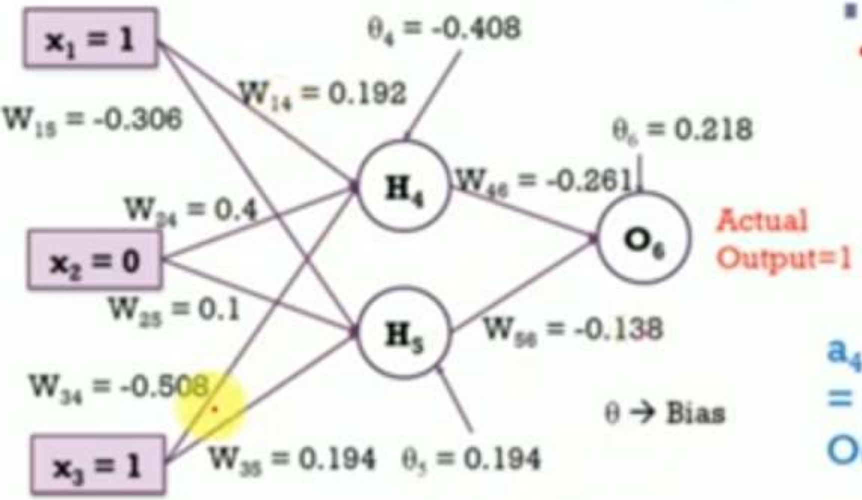
$\Delta w_{14} = \eta\, \delta_4\, x_1 = 0.9 * -0.0087 * 1 = -0.0078$
$w_{14}\text{ (new)} = \Delta w_{14} + w_{14}\text{(old)} = -0.0078 + 0.2 = 0.192$

- Similarly, update all other weights

i	j	w_{ij}	δ_j	x_i	η	Updated w_{ij}
4	6	-0.3	0.1311	0.332	0.9	-0.261
5	6	-0.2	0.1311	0.525	0.9	-0.138
1	4	0.2	-0.0087	1	0.9	0.192
1	5	-0.3	-0.0065	1	0.9	-0.306
2	4	0.4	-0.0087	0	0.9	0.4
2	5	0.1	-0.0065	0	0.9	0.1
3	4	-0.5	-0.0087	1	0.9	-0.508
3	5	0.2	-0.0065	1	0.9	0.194

- Similarly, update bais weights

θ_j	Previous θ_j	δ_j	η	Updated θ_j
Θ_6	0.1	0.1311	0.9	0.218
Θ_5	0.2	-0.0065	0.9	0.194
Θ_4	-0.4	-0.0087	0.9	-0.408



- Forward Pass: Compute output for y4, y5 and y6.

$$a_j = \sum_j (w_{i,j} * x_i) \qquad yj = F(aj) = \frac{1}{1 + e^{-a_j}}$$

Error $= y_{target} - y_6 = 0.485$

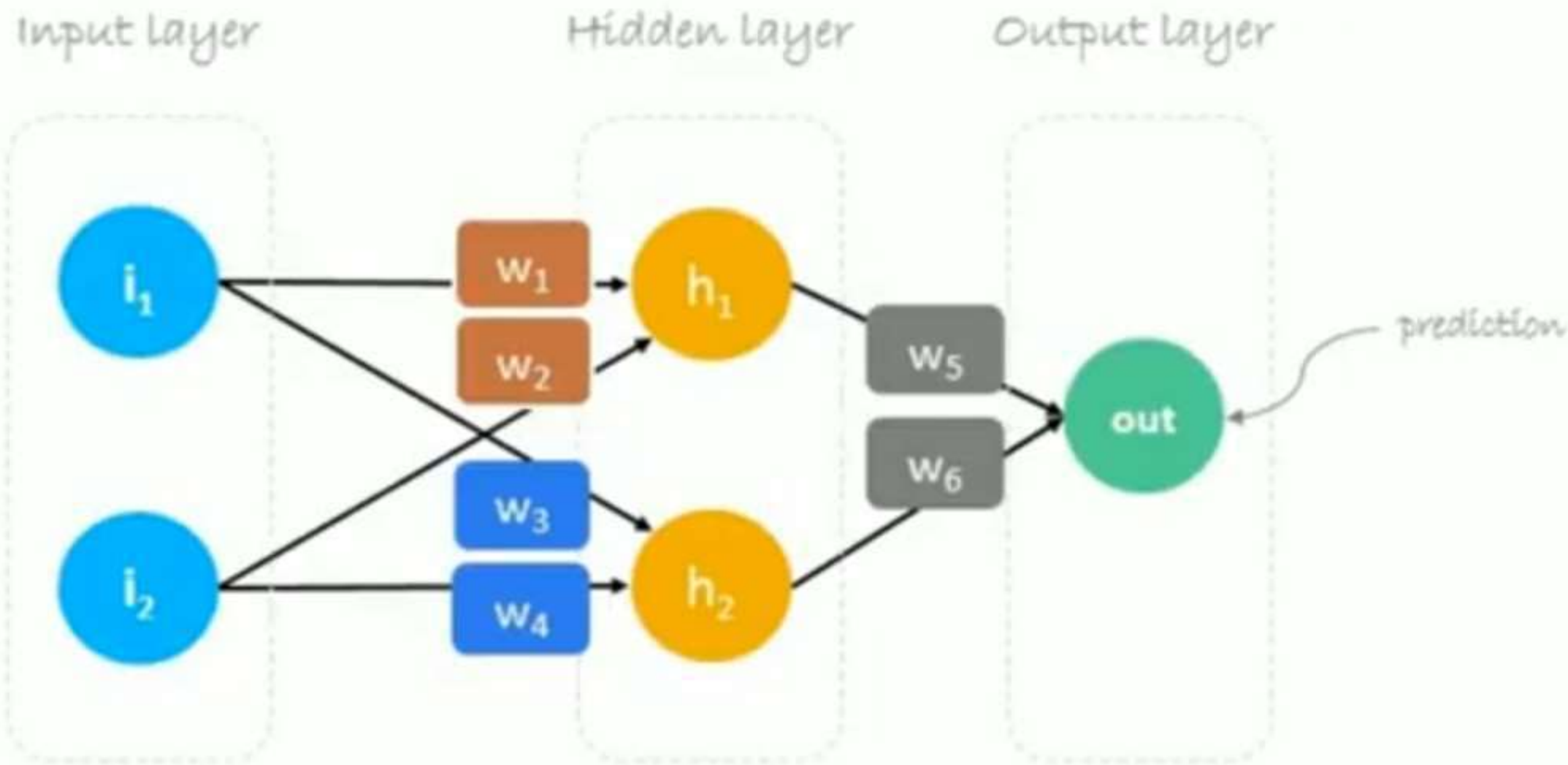
$a_4 = (w_{14} * x_1) + (w_{24} * x_2) + (w_{34} * x_3) + \theta_4$
$= (0.192 * 1) + (0.4 * 0) + (-0.508 * 1) + (-0.408) = -0.724$
$O(H_4) = y_4 = f(a_4) = 1/(1 + e^{0.724}) = 0.327$

$a_5 = (w_{15} * x_1) + (w_{25} * x_2) + (w_{35} * x_3) + \theta_5$
$= (-0.306 * 1) + (0.1 * 0) + (0.194 * 1) + (0.194) = 0.082$
$O(H_5) = y_5 = f(a_5) = 1/(1 + e^{-0.082}) = 0.520$

$a_6 = (w_{46} * H_4) + (w_{56} * H_5) + \theta_6$
$= (-0.261 * 0.327) + (-0.138 * 0.520) + 0.218 = 0.061$
$O(O_6) = y_6 = f(a_6) = 1/(1 + e^{-0.061}) = 0.515$ (Network Output)

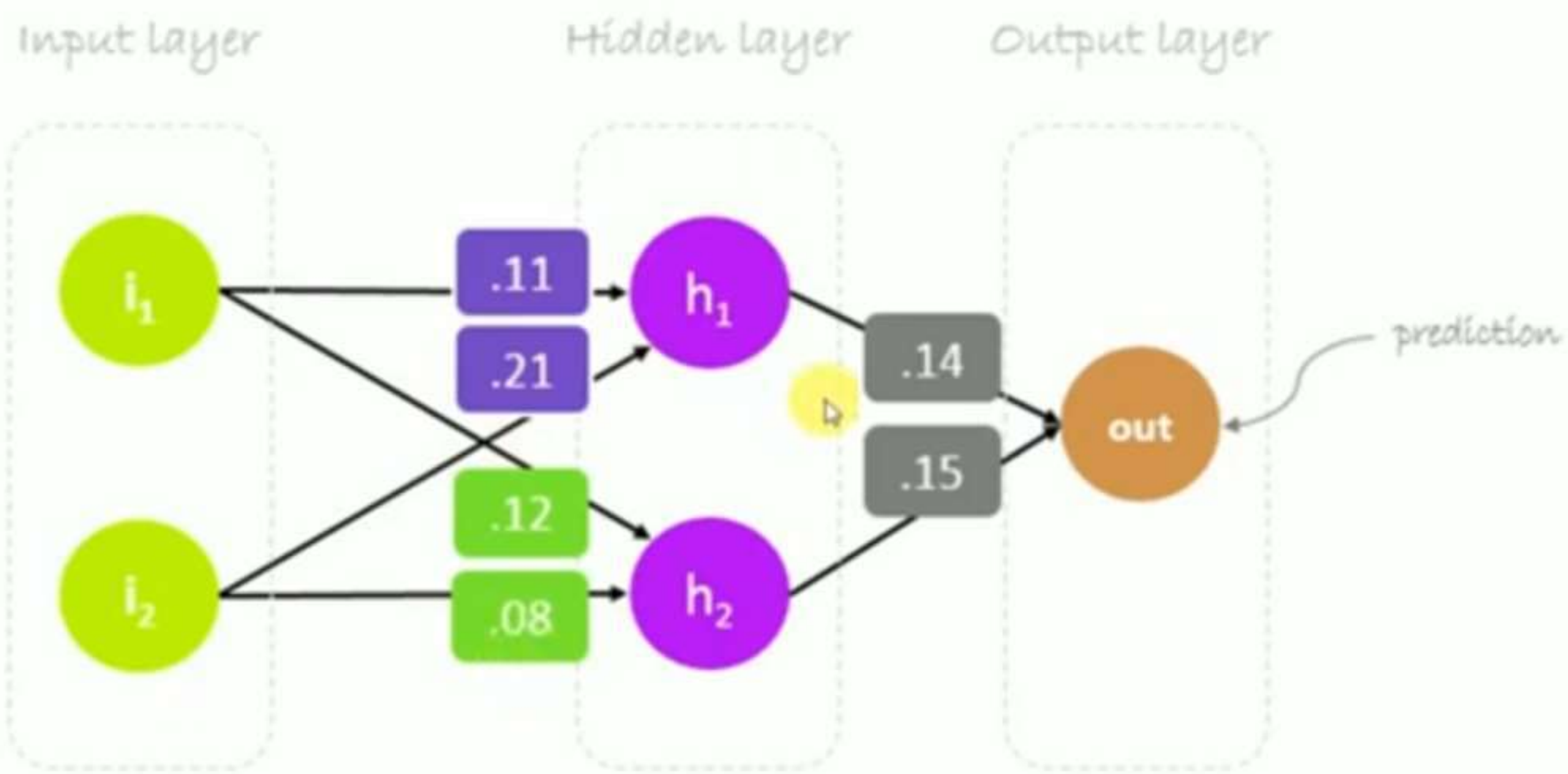
First, we will build a neural network with three layers:

- **Input** layer with two inputs neurons
- One **hidden** layer with two neurons
- **Output** layer with a single neuron


Input layer
Hidden layer
Output layer
i_1
i_2
w_1
w_2
w_3
w_4
h_1
h_2
w_5
w_6
out
prediction

Weights

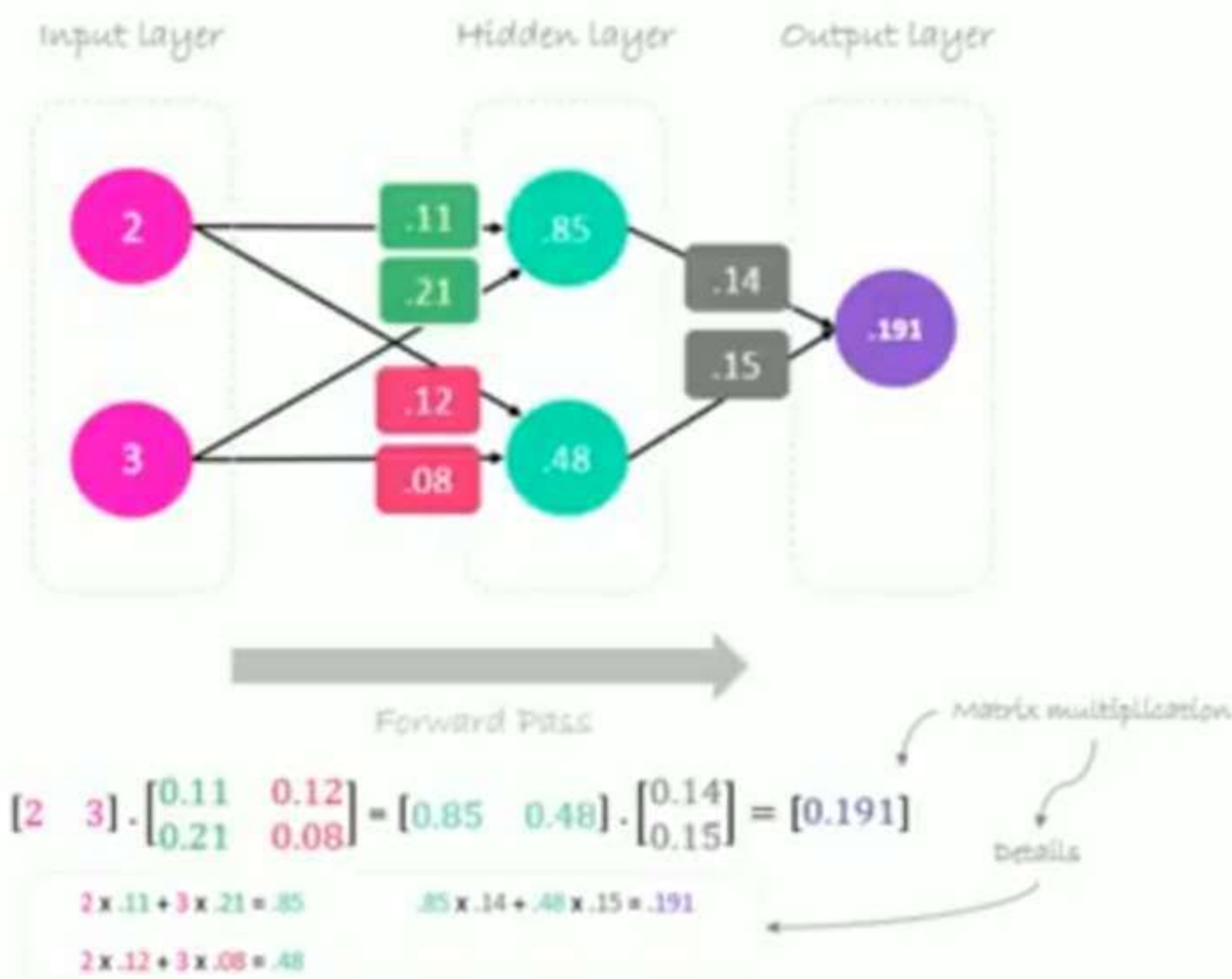
- Neural network training is about finding weights that minimize prediction error.
- We usually start our training with a set of randomly generated weights.
- Then, backpropagation is used to update the weights in an attempt to correctly map arbitrary inputs to outputs.
- Our initial weights will be as following: $w1 = 0.11$, $w2 = 0.21$, $w3 = 0.12$, $w4 = 0.08$, $w5 = 0.14$ and $w6 = 0.15$


Input layer
Hidden layer
Output layer
i1
i2
.11
.21
.12
.08
h1
h2
.14
.15
out
prediction

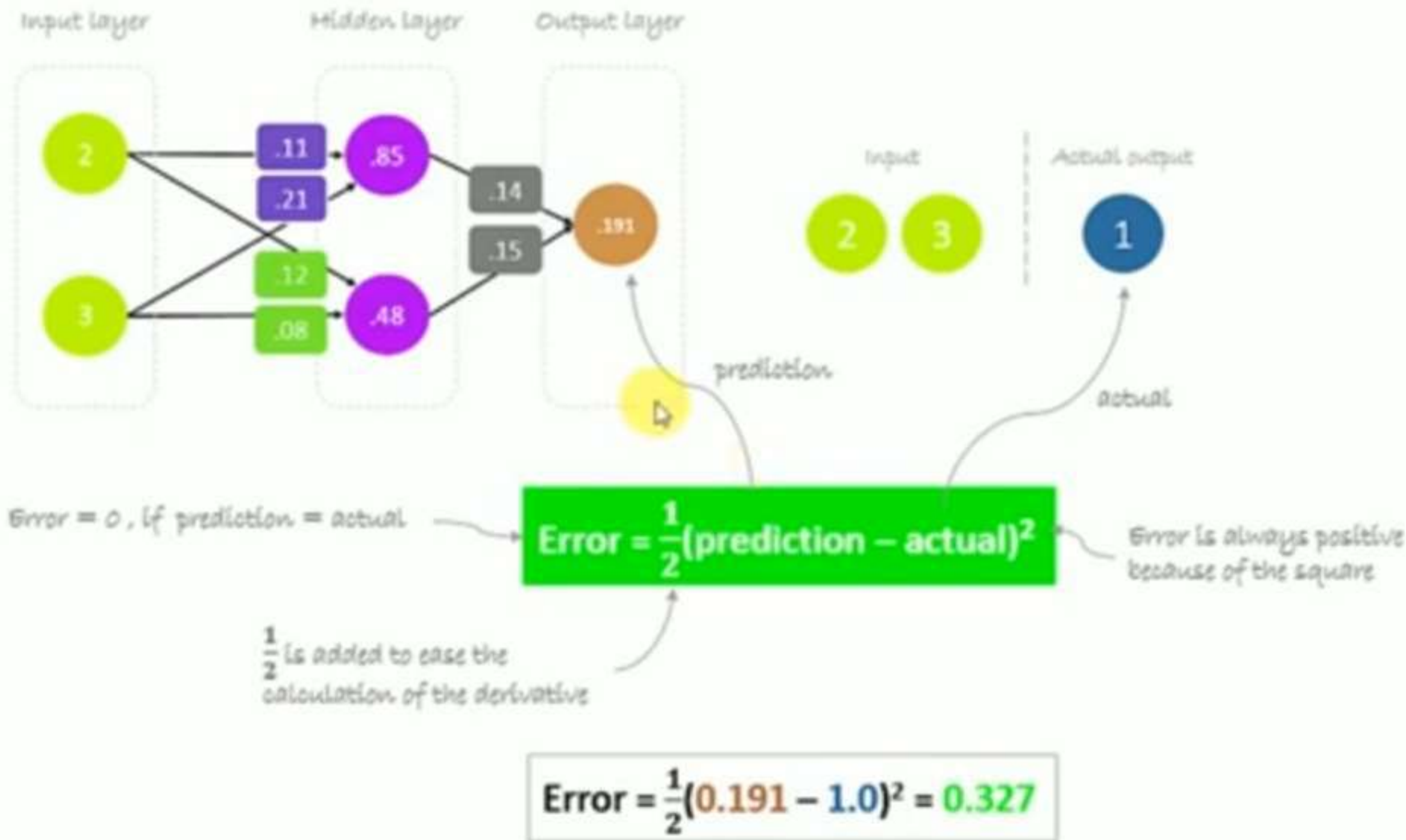
Dataset

Our single sample is as following inputs=[2, 3] and output=[1].

Forward Pass



Calculating Error



Reducing Error

- Our main goal of the training is to reduce the error.
- Since actual output is constant, the only way to reduce the error is to change prediction value.
- The question now is, how to change prediction value?
- By decomposing prediction into its basic elements we can find that weights are the variable elements affecting prediction value.
- In other words, in order to change prediction value, we need to change weights values.

Reducing Error

$$\text{prediction} = out$$

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

$h_1 = i_1 w_1 + i_2 w_2$

$h_2 = i_1 w_3 + i_2 w_4$

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

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

Backpropagation

- **Backpropagation**, short for "backward propagation of errors", is a mechanism used to update the **weights** using gradient descent.
- It calculates the gradient of the error function with respect to the neural network's weights.
- The calculation proceeds backwards through the network.

Backpropagation

Derivative of Error
with respect to weight

Old weight

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

New weight

Learning
rate

For example, to update w6, we take the current w6 and subtract the partial derivative of **error** function with respect to w6.

Backpropagation

Updated weights

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

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - a\,\Delta \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - \begin{bmatrix} a h_1 \Delta \\ a h_2 \Delta \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - a\,\Delta \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} . \begin{bmatrix} w_5 & w_6 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - \begin{bmatrix} a\, i_1 \Delta w_5 & a\, i_1 \Delta w_6 \\ a\, i_2 \Delta w_5 & a\, i_2 \Delta w_6 \end{bmatrix}$$

Backward Pass

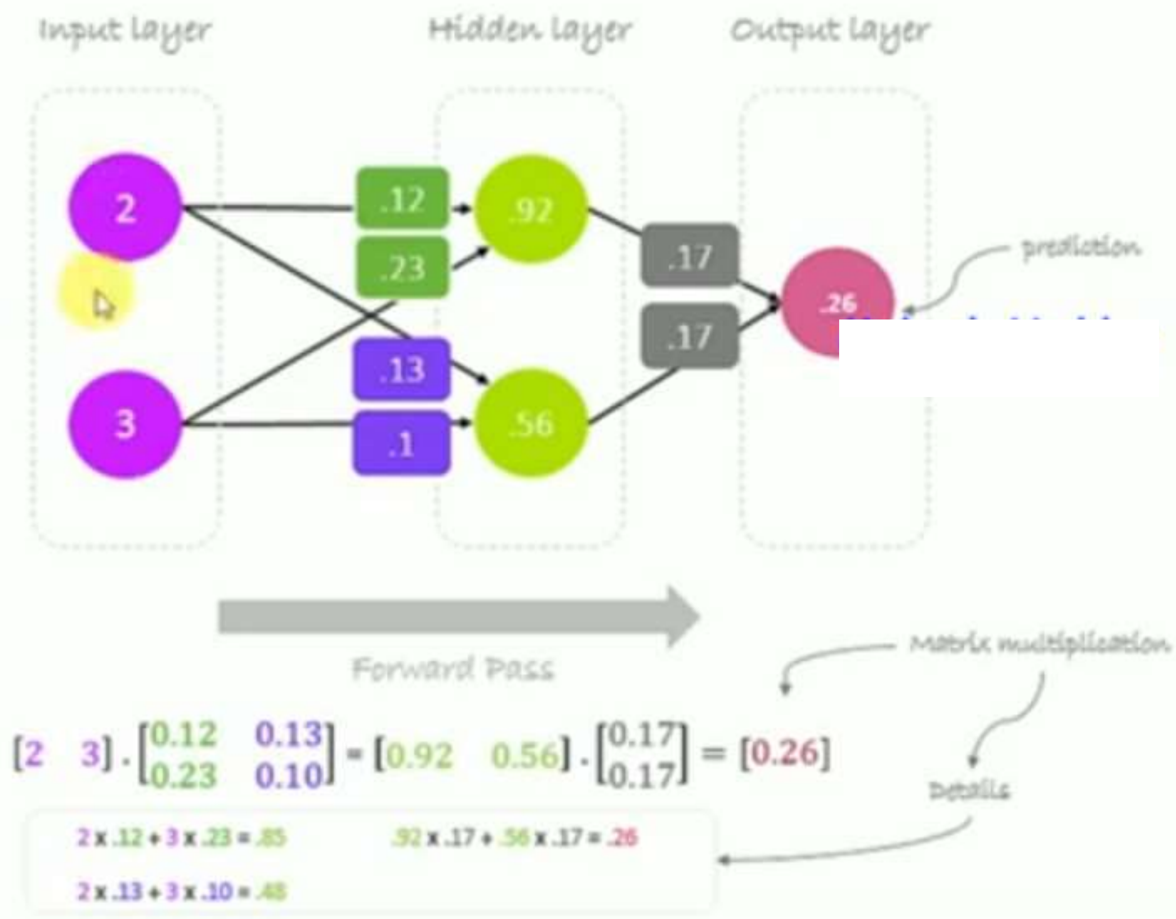
$\Delta = 0.191 - 1 = -0.809$ ← **Delta** = prediction - actual

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

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

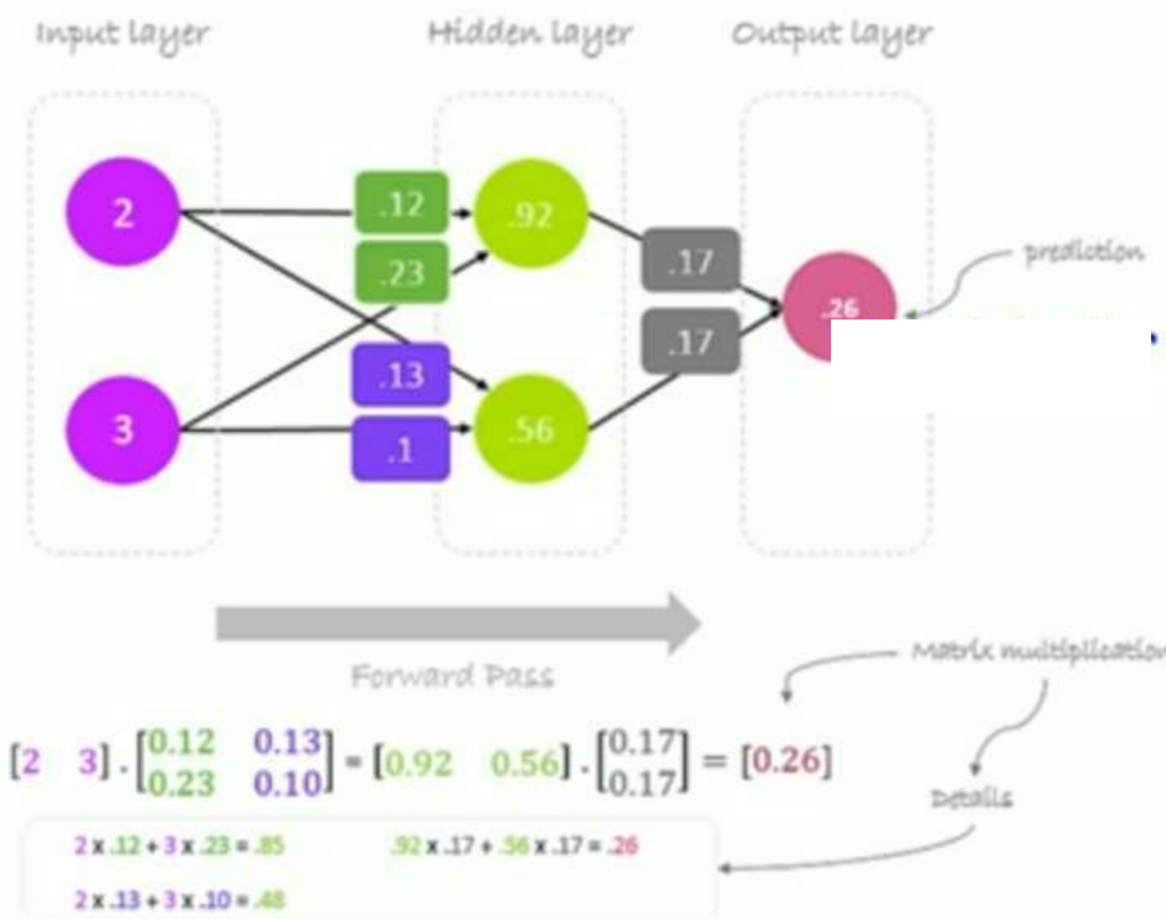
$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - 0.05(-0.809)\begin{bmatrix} 2 \\ 3 \end{bmatrix} . \begin{bmatrix} 0.14 & 0.15 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - \begin{bmatrix} -0.011 & -0.012 \\ -0.017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

Backward Pass



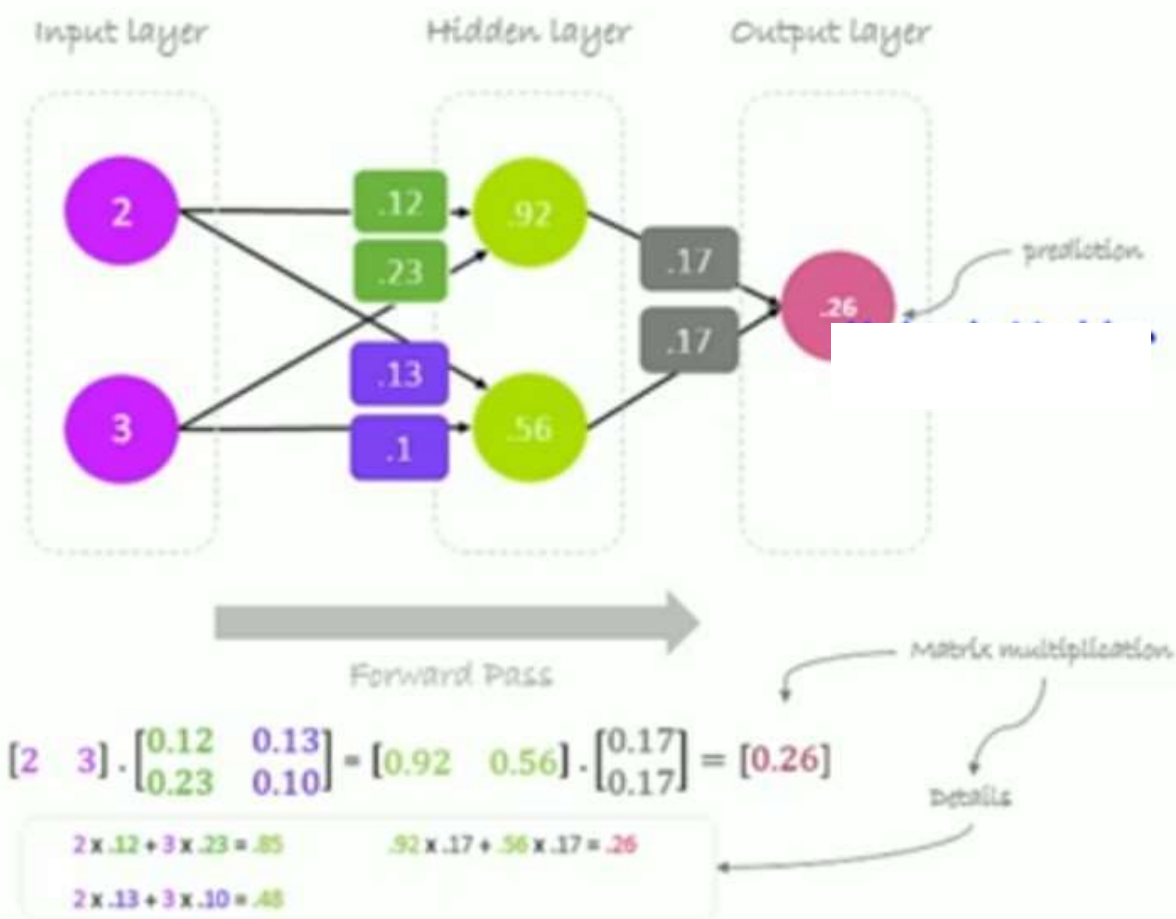
Backward Pass

- We can notice that the **prediction** 0.26 is a little bit closer to **actual output** than the previously predicted one 0.191.



Backward Pass

- We can notice that the **prediction** 0.26 is a little bit closer to **actual output** than the previously predicted one 0.191.
- We can repeat the same process of backward and forward pass until **error** is close or equal to zero.



Thank you for your attention