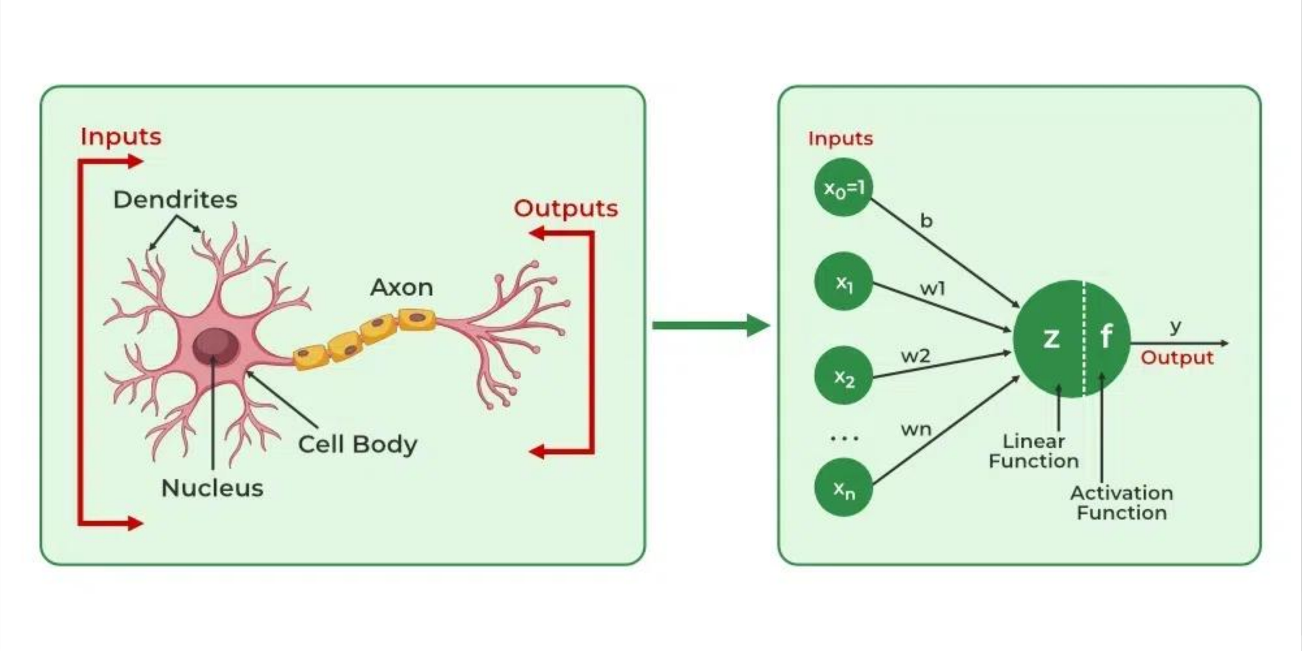
COMP4431 Artificial Intelligence

Neural Networks

Raymond Pang
Department of Computing
The Hong Kong Polytechnic University

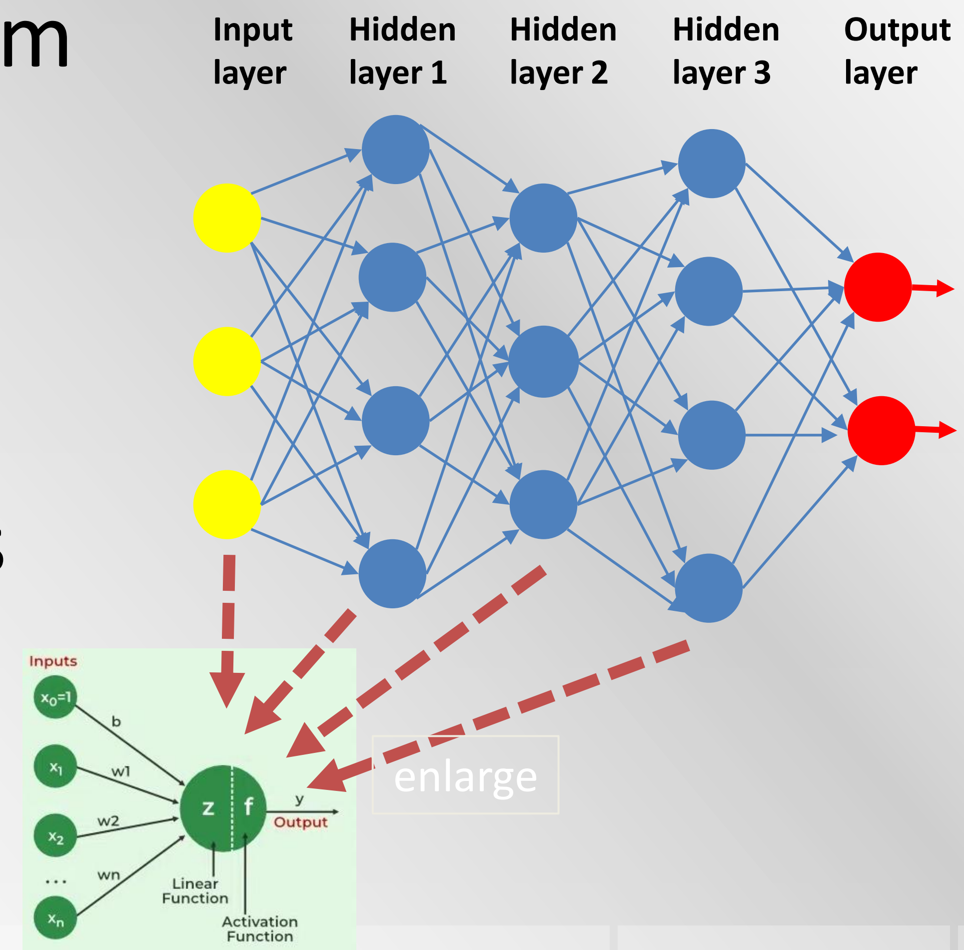
Neural Network

- A neural net consists of a large number of simple processing elements called neurons, units, cells or nodes.
- Neurons contains a number of weights, activation function and they are interconnected



Artificial neural network example

- The neurons are connected and form layers
  - Input layer
  - Hidden layer
  - Output layer
- Each connection is associated with a weight.



Neural Network

- Neural Network learns by adjusting the weights so as to be able to correctly classify the training data and hence, after testing phase, to classify unknown data.
- Neural Network needs long time for training.
- Neural Network has a high tolerance to noisy and incomplete data

Training a neural network

- A neural network is trained with *m* training samples

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots\ldots(x^{(m)}, y^{(m)})$$

$x^{(i)}$ is an input vector, $y^{(i)}$ is an output vector

- Training objective: minimize the prediction error (loss)

$$min \sum_{i=1}^{m} (y^{(i)} - f_W(x^{(i)}))^2$$

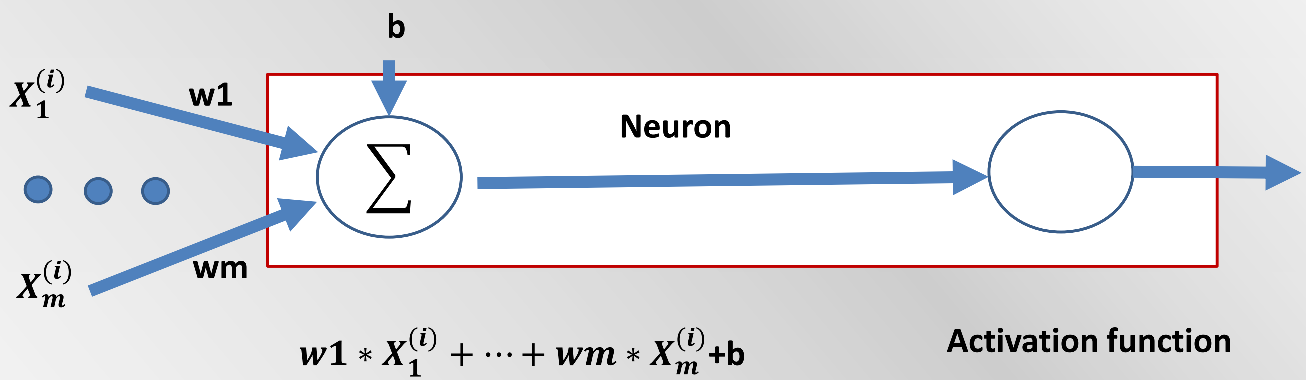
$f_W(x^{(i)})$ is the predicted output vector for the input vector $x^{(i)}$

- Approach: Gradient descent (stochastic gradient descent, batch gradient descent, mini-batch gradient descent).
  - Use error to adjust the weight value to reduce the loss. The adjustment amount is proportional to the contribution of each weight to the loss – Given an error, adjust the weight a little to reduce the error.

Algorithm for learning artificial neural network

- Initialize the weights $\overrightarrow{W} = [W_0\ , W_1\ , \ldots \ldots , W_k\ ]$
- Training
  - For each training data $(x^{(i)}, y^{(i)})$, Using forward propagation to compute the neural network output vector $f_{\overrightarrow{W}}(x^{(i)})$
  - Compute the error E (various definitions)
  - Use backward propagation to compute $\frac{\partial E}{\partial W_k}$ for each weight W_k
  - Update $W_k = W_k - \alpha \frac{\partial E}{\partial W_k}$
  - Repeat until E is sufficiently small.

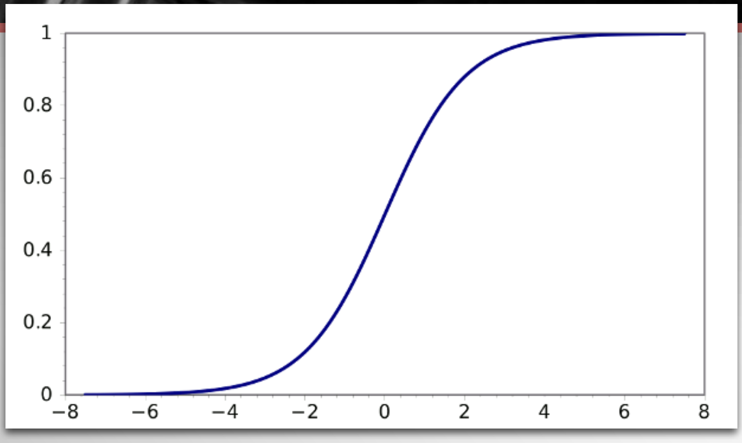
A single neuron

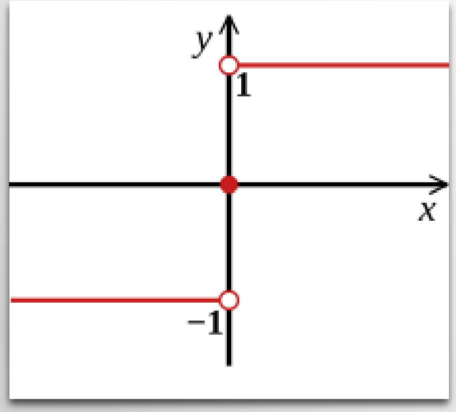


- An artificial neuron has two components:
  - weighted sum and
  - activation function.
- Variety of activation functions: Sigmoid, ReLU, etc.

Sigmoid function

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





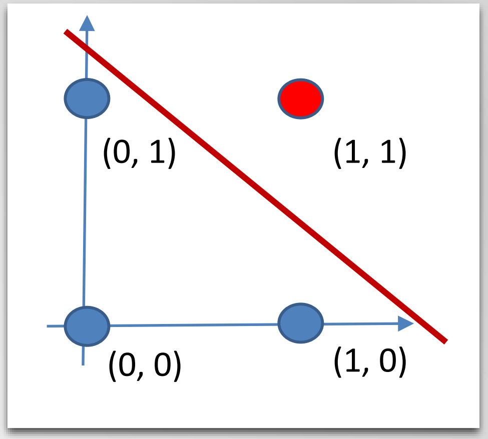
- Turn continuous value to discrete (e.g 0 and 1)
- Similar to a sign function
- But its derivative is well defined instead
- The derivative of sigmoid
  $$\frac{d}{dx} sigmoid(x) = sigmoid(x)\big(1 - sigmoid(x)\big)$$

Training for the logic AND with a single neuron

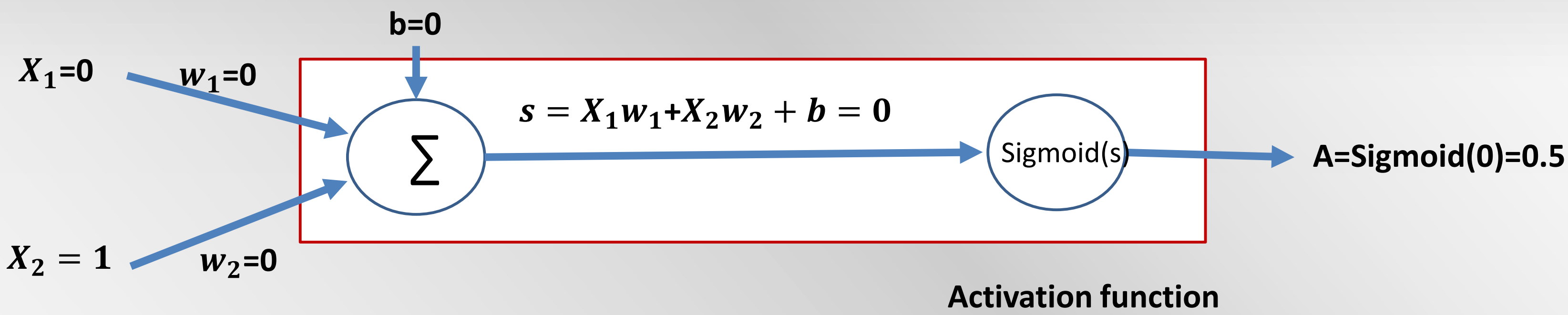
- In general, one neuron can be trained to realize a linear function.
- Logic AND function is a linear function:

$x1$	$x2$	$x1 \land x2$
0	0	0
0	1	0
1	0	0
1	1	1

Logic AND (∧) operation



Training with a single neuron



- Consider training data input (X_1=0, $X_2 = 1$), expected output Y=0.
- A forward propagation of NN,
- weight sum part :

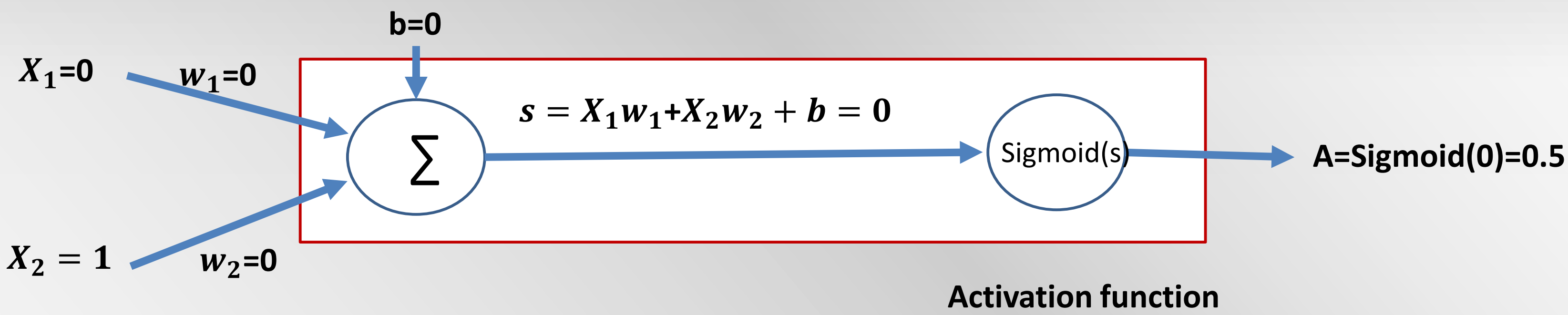
  $\boldsymbol{X_1w_1}$+$\boldsymbol{X_2w_2 + b}$ = 0*0 + 1*0 +0 = 0

  After activation func: A= **Sigmoid(0)** = 0.5
- Error: $E = \frac{1}{2}(Y - A)^2 = 0.125$

Chain Rules

- To update w_1, w_2, and b, gradient descent needs to compute $\frac{\partial E}{\partial w_1}$, $\frac{\partial E}{\partial w_2}$, and $\frac{\partial E}{\partial b}$
- If a variable *z* depends on the variable *y*, which itself depends on the variable *x*, then *z* depends on *x* as well, via the intermediate variable *y*.
- The **chain rule** is a formula that expresses the derivative as : $\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$
- $\frac{\partial E}{\partial W_1} = \frac{\partial E}{\partial A}\frac{\partial A}{\partial s}\frac{\partial s}{\partial W_1}$

Training with a single neuron



$$\frac{\partial E}{\partial W_1} = \frac{\partial E}{\partial A}\frac{\partial A}{\partial s}\frac{\partial s}{\partial W_1}$$

$$\frac{\partial E}{\partial A} = \frac{\partial(\frac{1}{2}(Y-A)^2)}{\partial A} = \mathrm{A} - \mathrm{Y} = 0.5 - 0 = 0.5$$

$\frac{\partial A}{\partial s} = \frac{\partial(sigmoid(s))}{\partial s}$ = sigmoid(s) (1-sigmoid(s)) = 0.5 (1-0.5) = 0.25,

$$\frac{\partial s}{\partial W_1} = \frac{\partial(X_1w_1+X_2w_2+b)}{\partial W_1} = X_1 = 0$$

- To update w_1: $w_1 = w_1 - rate * \frac{\partial E}{\partial W_1}$
  = 0 – 0.1*0.5*0.25*0 = 0

Assume rate = 0.1

Training with a single neuron

Similarly,

$$\frac{\partial E}{\partial W_2} = \frac{\partial E}{\partial A}\frac{\partial A}{\partial s}\frac{\partial s}{\partial W_2} \qquad \frac{\partial E}{\partial A} = \frac{\partial(\frac{1}{2}(Y-A)^2)}{\partial A} = 0.5$$

$\frac{\partial A}{\partial s} = \frac{\partial(sigmoid(s))}{\partial s}$ = sigmoid(s) (1-sigmoid(s)) = 0.5 (1-0.5) = 0.25,

$\frac{\partial s}{\partial W_2} = \frac{\partial(X_1 w_1 + X_2 w_2 + b)}{\partial W_2} = X_2$ = 1

- To update w_2: $w_2 = w_2 - rate * \frac{\partial E}{\partial W_2}$
  = 0 – 0.1*0.5*0.25*1
  = -0.0125

Training with a single neuron

Finally, with respect to b (bias term)

$$\frac{\partial E}{\partial b} = \frac{\partial E}{\partial A}\frac{\partial A}{\partial s}\frac{\partial s}{\partial b} \qquad \frac{\partial E}{\partial A} = \frac{\partial(\frac{1}{2}(Y-A)^2)}{\partial A} = 0.5$$

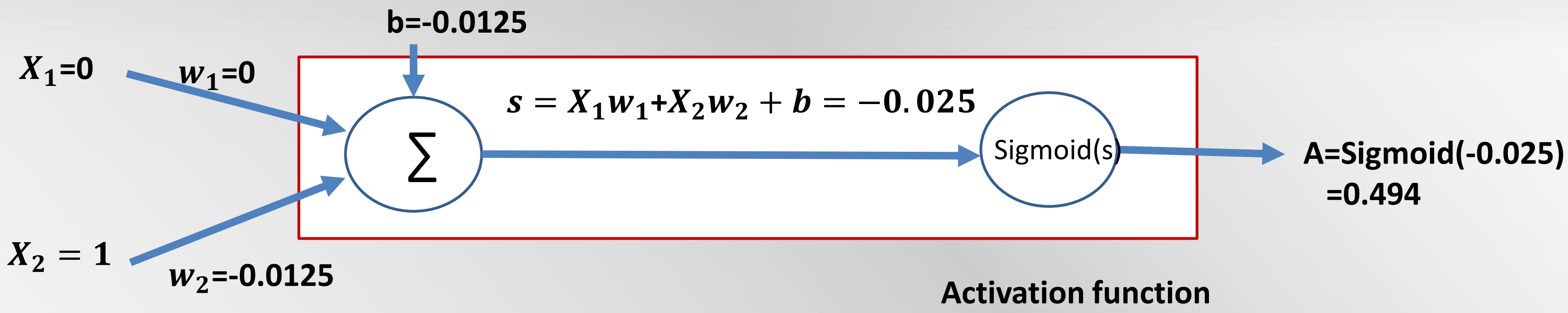
$\frac{\partial A}{\partial s} = \frac{\partial(sigmoid(s))}{\partial s}$ = sigmoid(s) (1-sigmoid(s))

= 0.5 (1-0.5) = 0.25,

$\frac{\partial s}{\partial b} = \frac{\partial(X_1 w_1 + X_2 w_2 + b)}{\partial b} = 1$

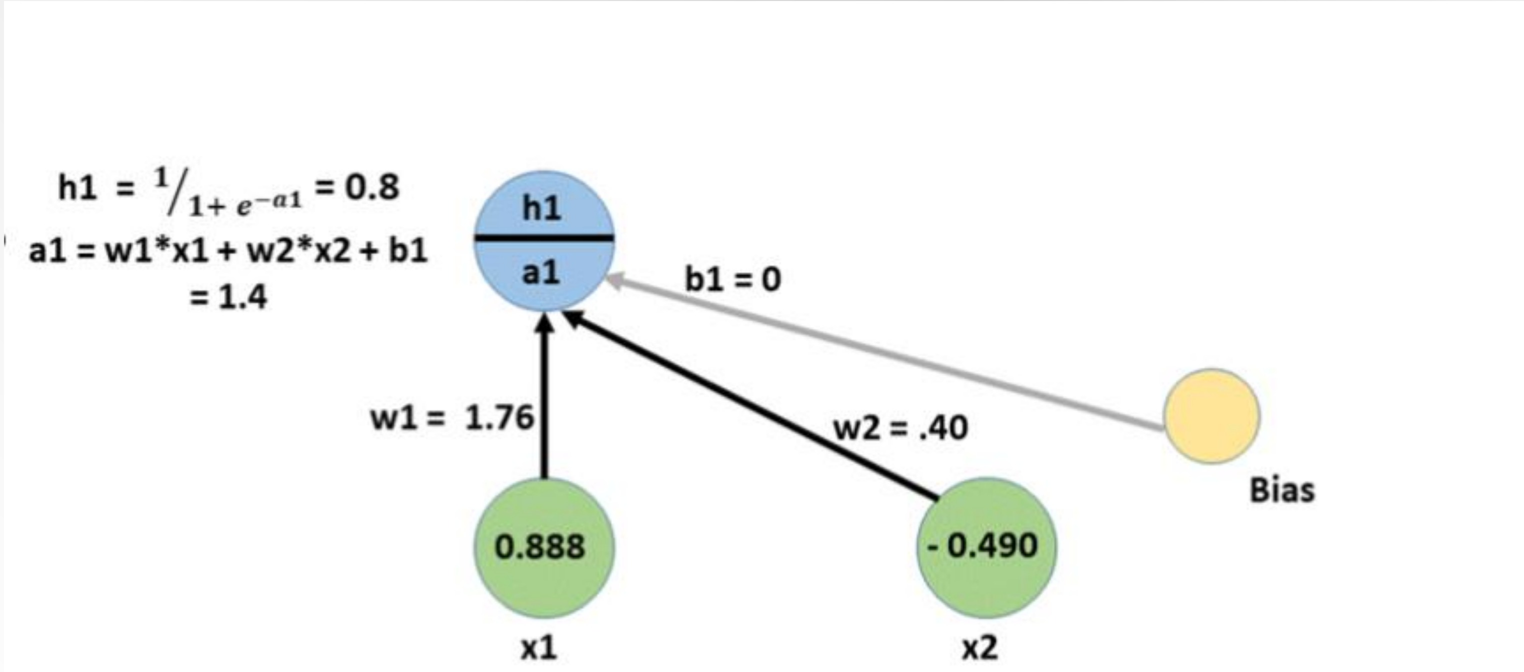
- To update b: b$= b - rate * \frac{\partial E}{\partial b}$ = 0 – 0.1*0.5*0.25*1 = -0.0125

Training with a single neuron

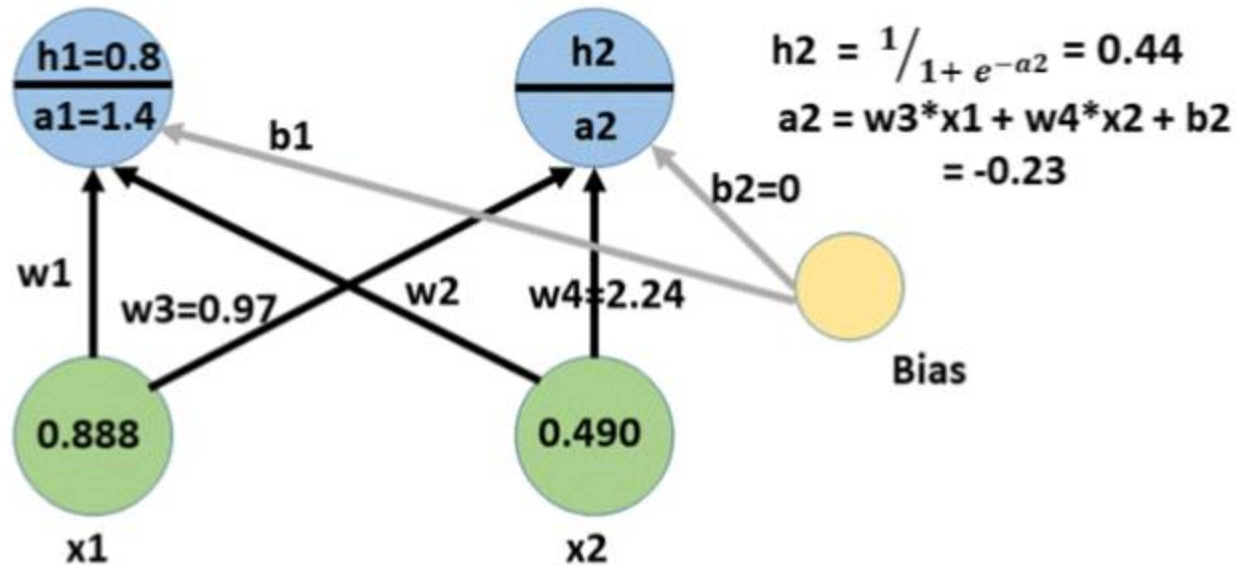


- This process is repeated until the error is sufficiently small
- Do a forward propagation of NN with updated weight, it become -0.494, closer to 0
- The initial weight should be randomized.
- Gradient descent can get stuck in the local optimal.

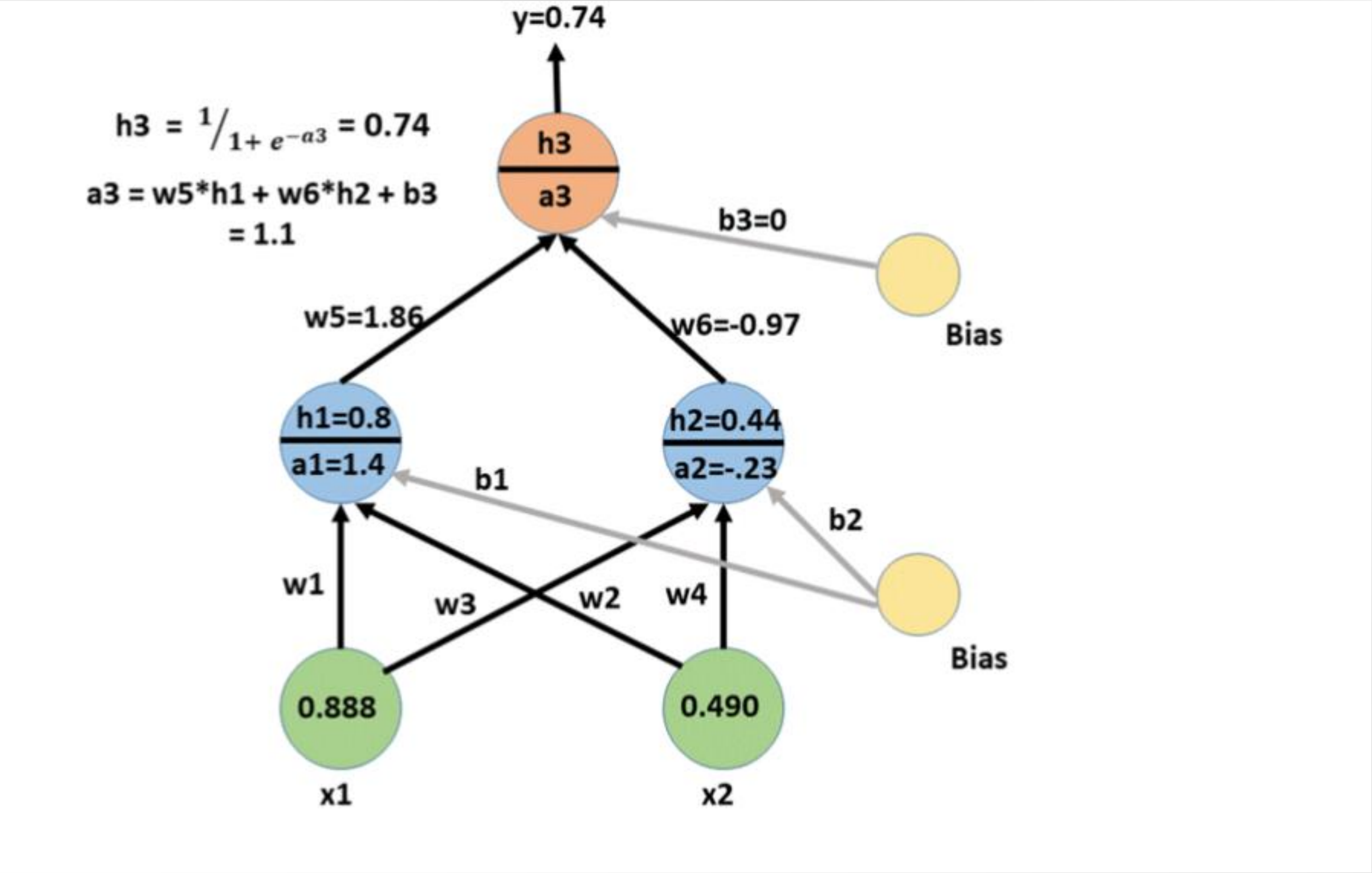
Forward Propagation Example with multiple Neurons



Forward Propagation Example with multiple Neurons

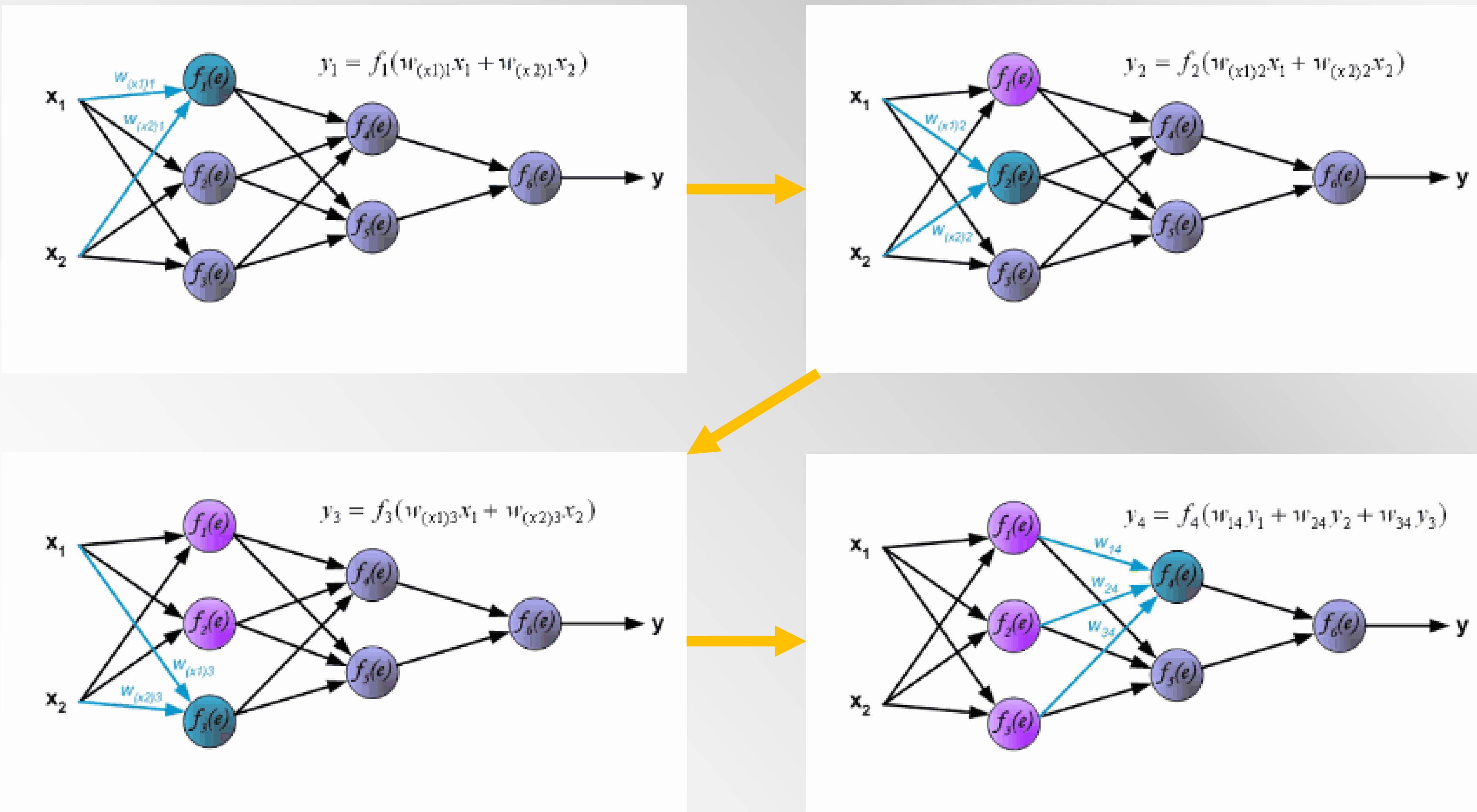


Forward Propagation Example with multiple Neurons

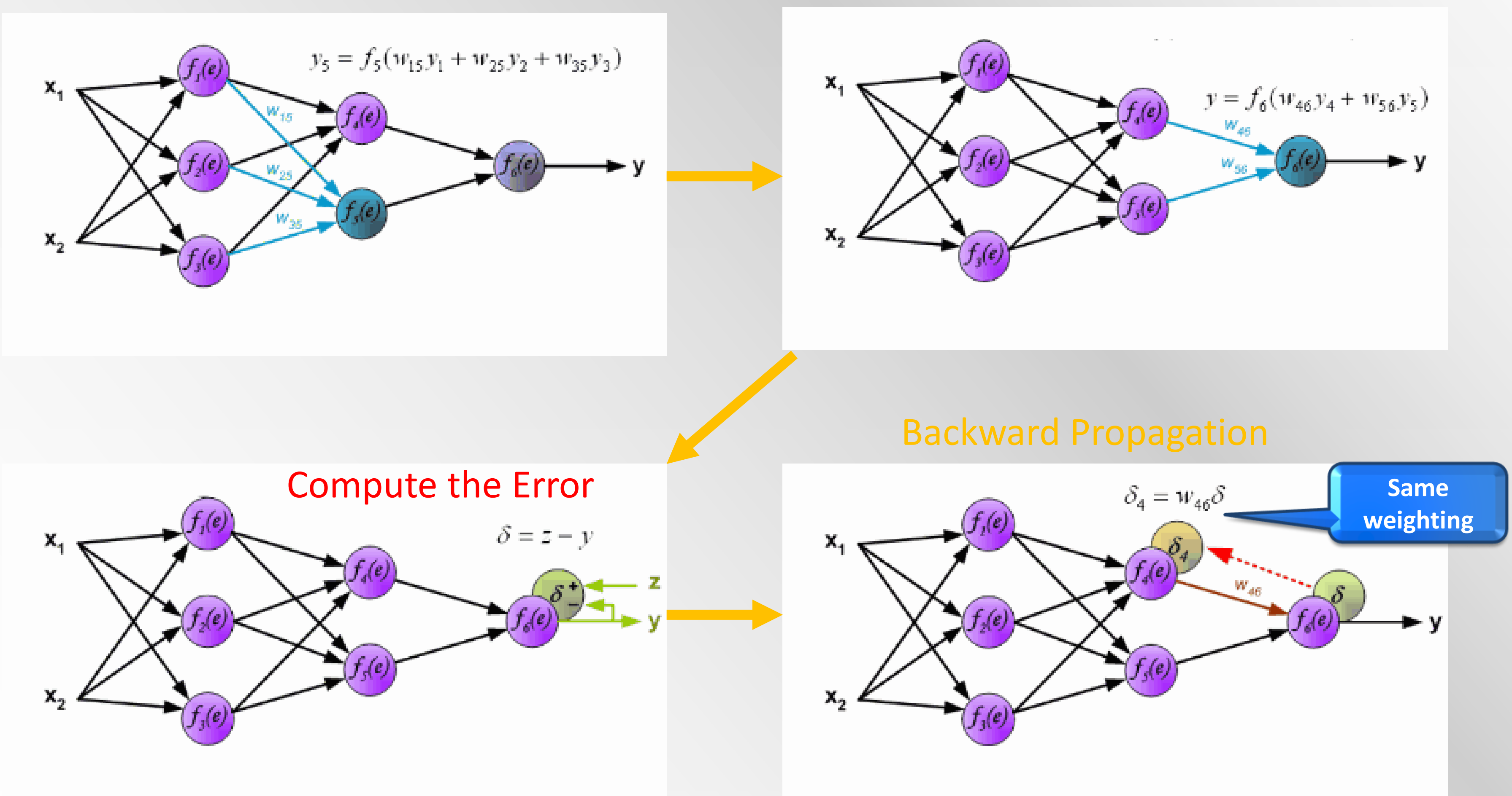


Forward and backward Propagation Illustrated

Forward Propagation

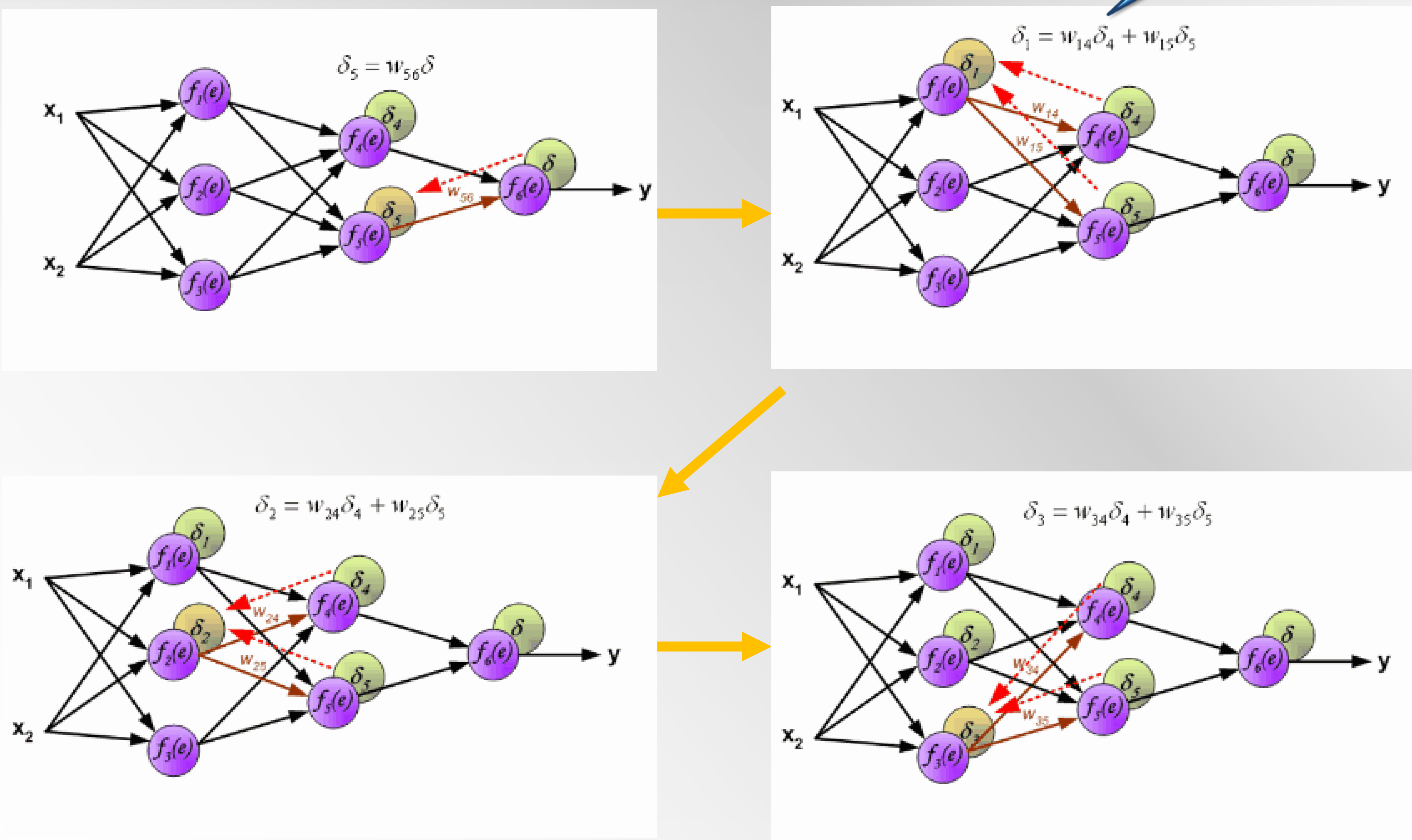


Forward and backward Propagation Illustrated

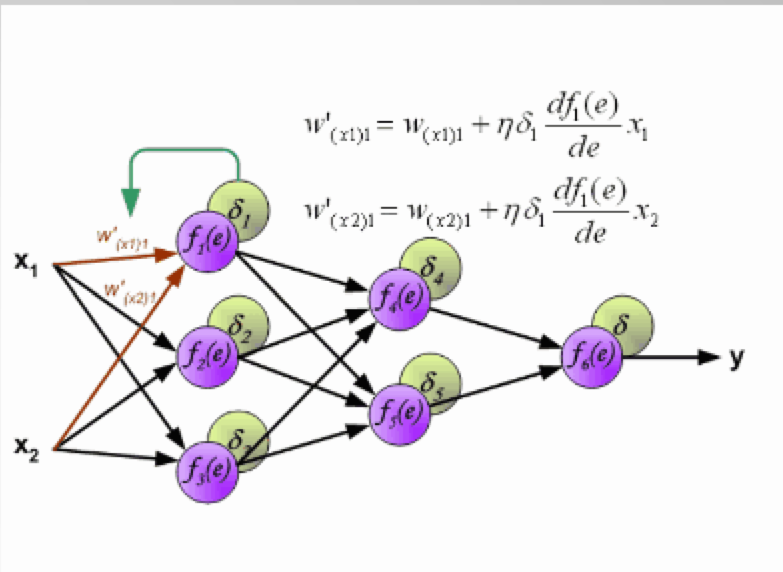


Forward and backward Propagation Illus

Error from all output neurons

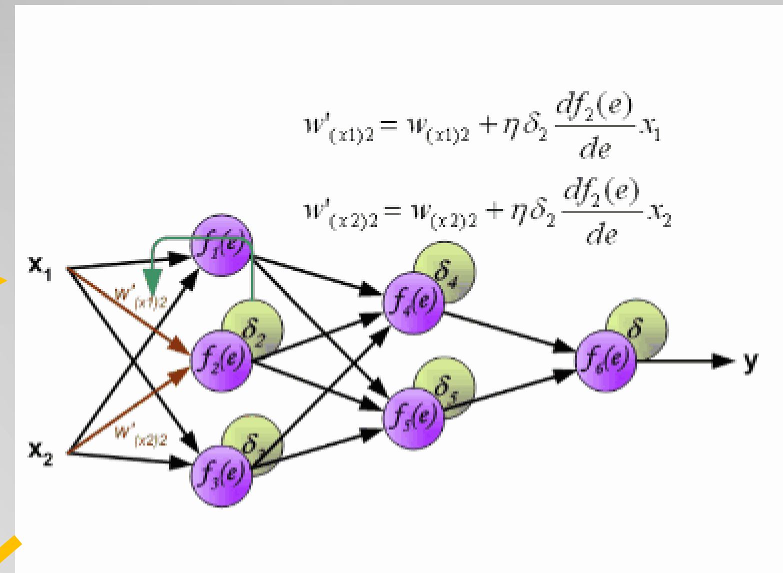


Update Weights



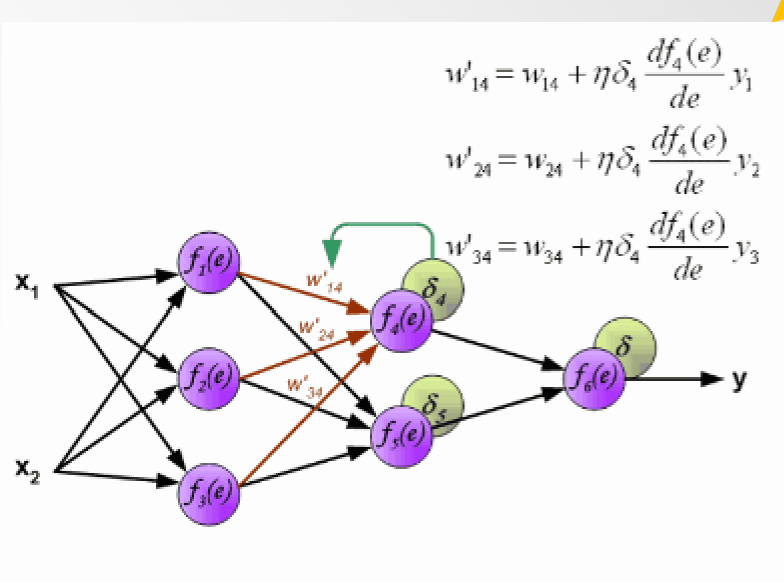
$$w'_{(x1)1} = w_{(x1)1} + \eta \delta_1 \frac{df_1(e)}{de} x_1$$

$$w'_{(x2)1} = w_{(x2)1} + \eta \delta_1 \frac{df_1(e)}{de} x_2$$



$$w'_{(x1)2} = w_{(x1)2} + \eta \delta_2 \frac{df_2(e)}{de} x_1$$

$$w'_{(x2)2} = w_{(x2)2} + \eta \delta_2 \frac{df_2(e)}{de} x_2$$

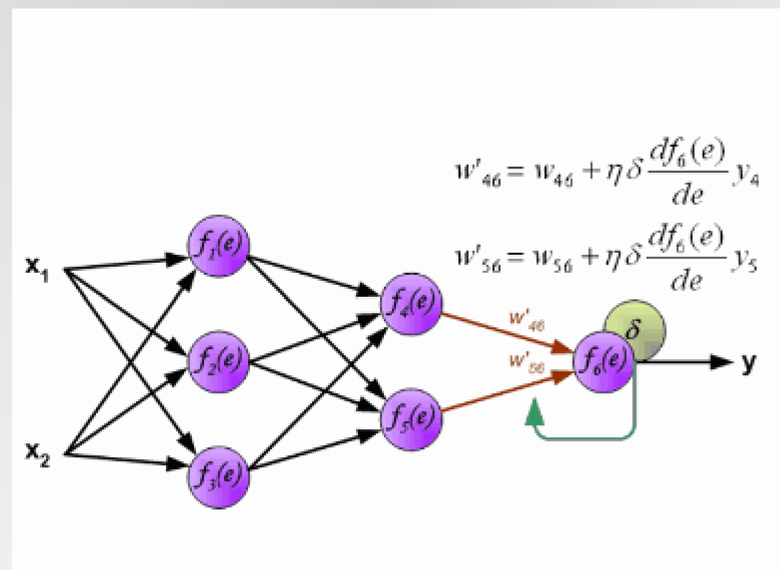


$$w'_{14} = w_{14} + \eta \delta_4 \frac{df_4(e)}{de} y_1$$

$$w'_{24} = w_{24} + \eta \delta_4 \frac{df_4(e)}{de} y_2$$

$$w'_{34} = w_{34} + \eta \delta_4 \frac{df_4(e)}{de} y_3$$

....



$$w'_{46} = w_{46} + \eta \delta \frac{df_6(e)}{de} y_4$$

$$w'_{56} = w_{56} + \eta \delta \frac{df_6(e)}{de} y_5$$

Summary

- Neural Network
- Forward and Backward Propagation