Data Mining

Lecture Notes for Chapter 4

Artificial Neural Networks

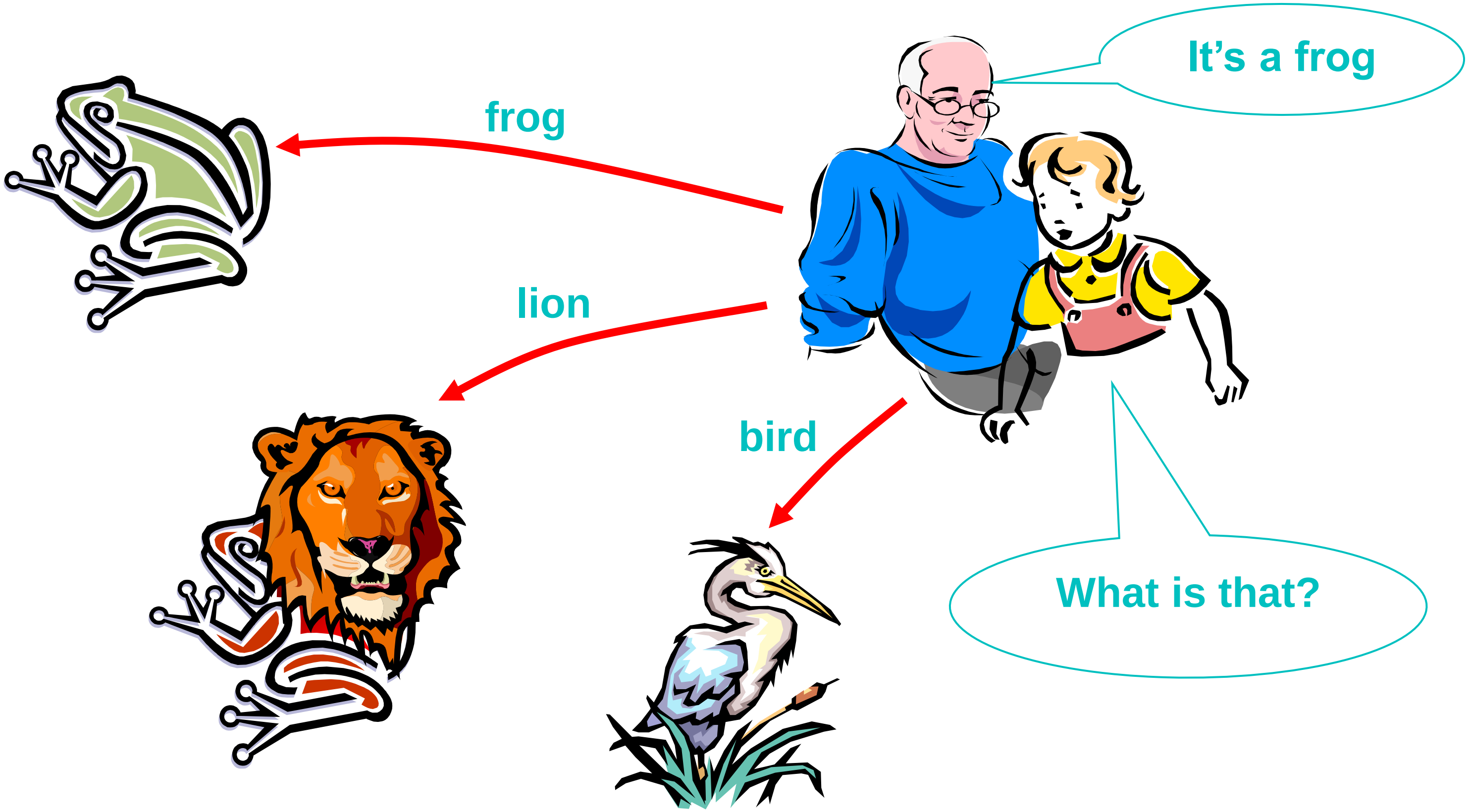
Introduction to Data Mining , 2nd Edition

by

Tan, Steinbach, Karpatne, Kumar

The idea of ANNs..?

- NNs learn relationship between cause and effect or organize large volumes of data into orderly and informative patterns.



Neural networks to the rescue…

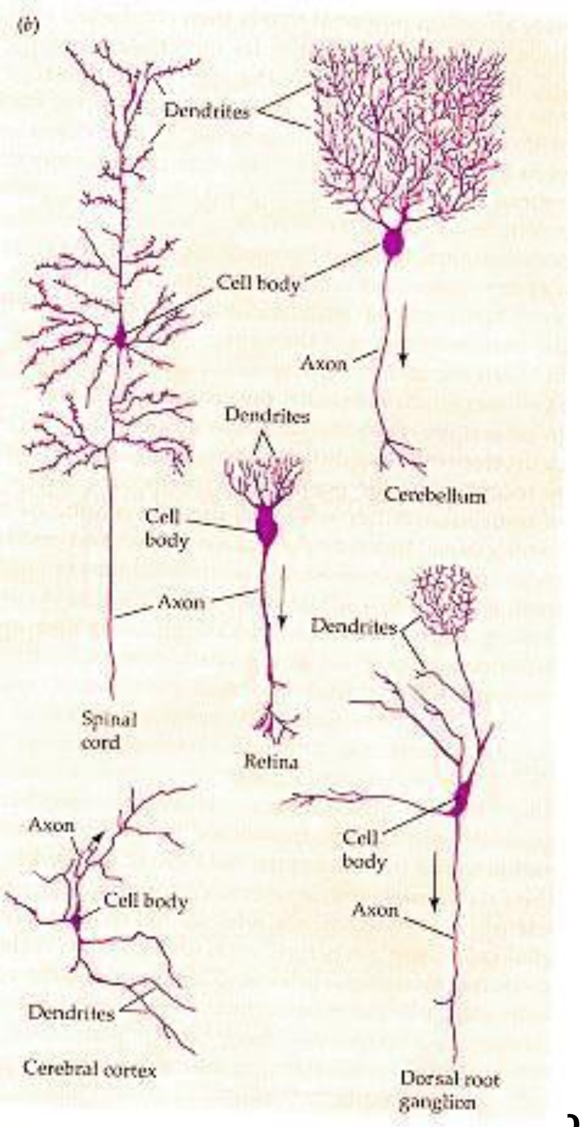
- **Neural network:** *information processing paradigm inspired by biological nervous systems, such as our brain*
- Structure: large number of highly interconnected processing elements (*neurons*) working together
- Like people, they learn *from experience* (by example)

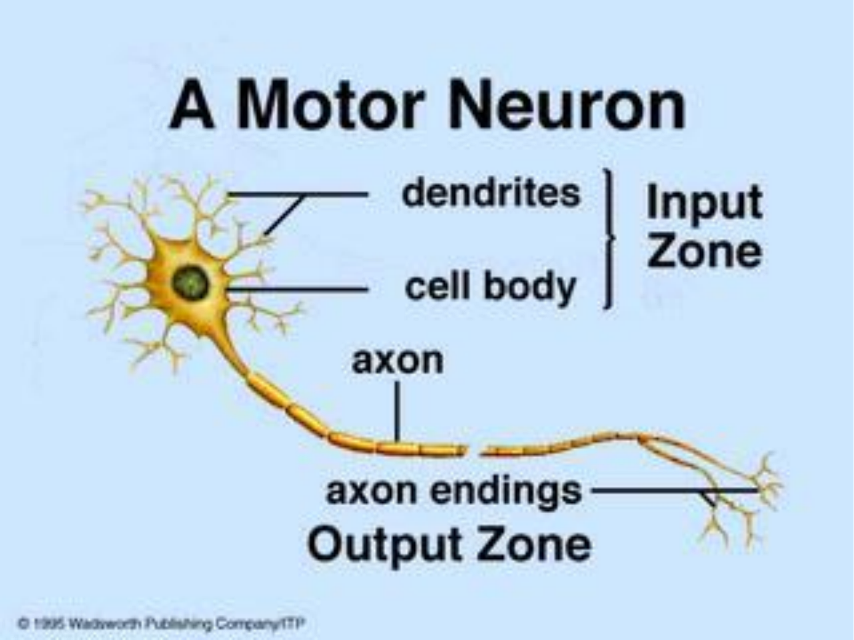
Definition of ANN

“Data processing system consisting of a large number of simple, highly interconnected processing elements (artificial neurons) in an architecture inspired by the structure of the cerebral cortex of the brain”

(Tsoukalas & Uhrig, 1997).

Biological Neural Networks

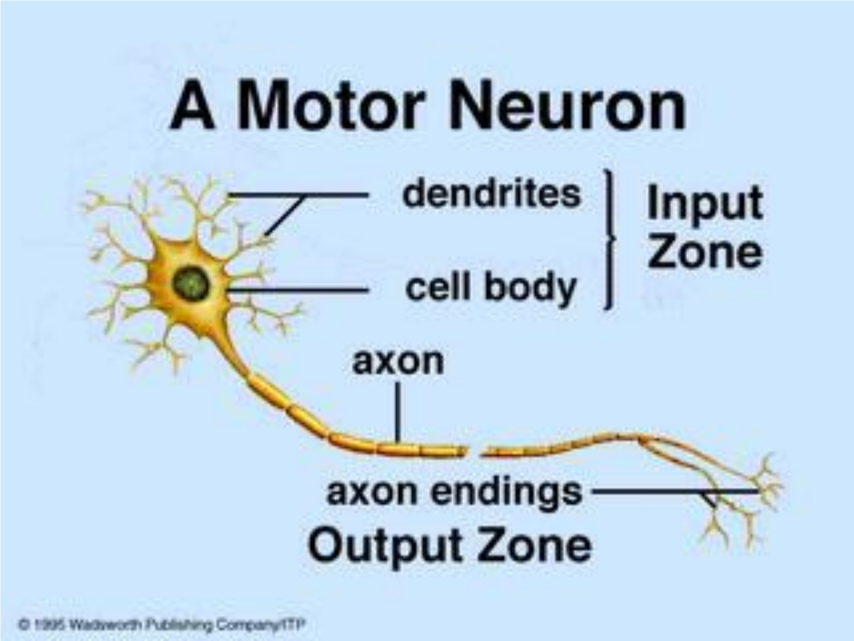




Biological neuron

Biological Neural Networks

- A biological neuron has three types of main components; dendrites, soma (or cell body) and axon.
- Dendrites receives signals from other neurons.



© 1995 Wadsworth Publishing Company/ITP

- The soma, sums the incoming signals. When sufficient input is received, the cell fires; that is it transmit a signal over its axon to other cells.

Artificial Neurons

- ANN is an information processing system that has certain performance characteristics in common with biological nets.
- Several key features of the processing elements of ANN are suggested by the properties of biological neurons:

1. The processing element receives many signals.
2. Signals may be modified by a weight at the receiving synapse.
3. The processing element sums the weighted inputs.
4. Under appropriate circumstances (sufficient input), the neuron transmits a single output.
5. The output from a particular neuron may go to many other neurons.

Artificial Neurons

- ANN is an information processing system that has certain performance characteristics in common with biological nets.
- Several key features of the processing elements of ANN are suggested by the properties of biological neurons:

1. The processing element receives many signals.
2. Signals may be modified by a weight at the receiving synapse.
3. The processing element sums the weighted inputs.
4. Under appropriate circumstances (sufficient input), the neuron transmits a single output.
5. The output from a particular neuron may go to many other neurons.

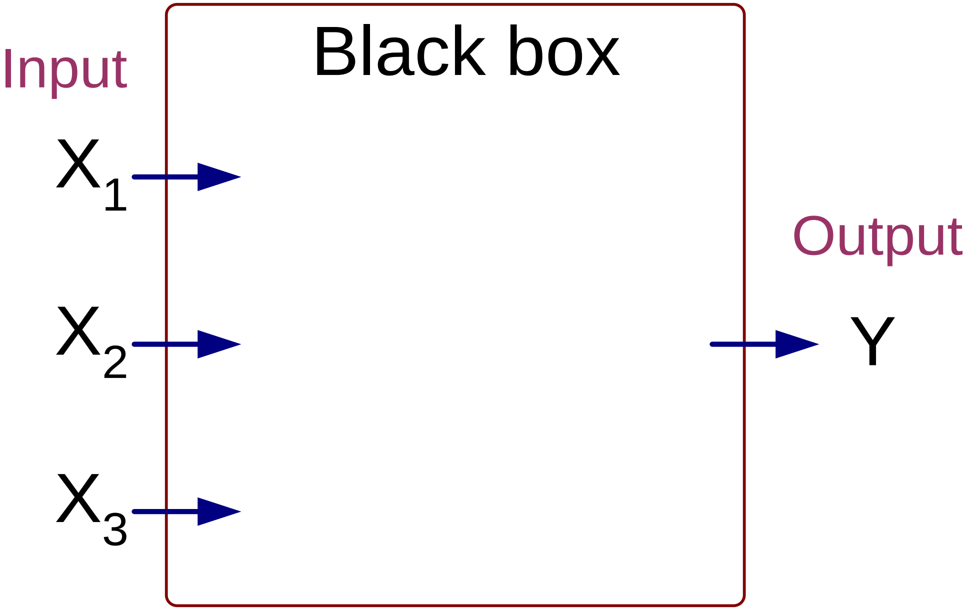
Artificial Neurons

- ANNs have been developed as generalizations of mathematical models of neural biology, based on the assumptions that:

1. Information processing occurs at many simple elements called neurons.
2. Signals are passed between neurons over connection links.
3. Each connection link has an associated weight, which, in typical neural net, multiplies the signal transmitted.
4. Each neuron applies an activation function to its net input to determine its output signal.

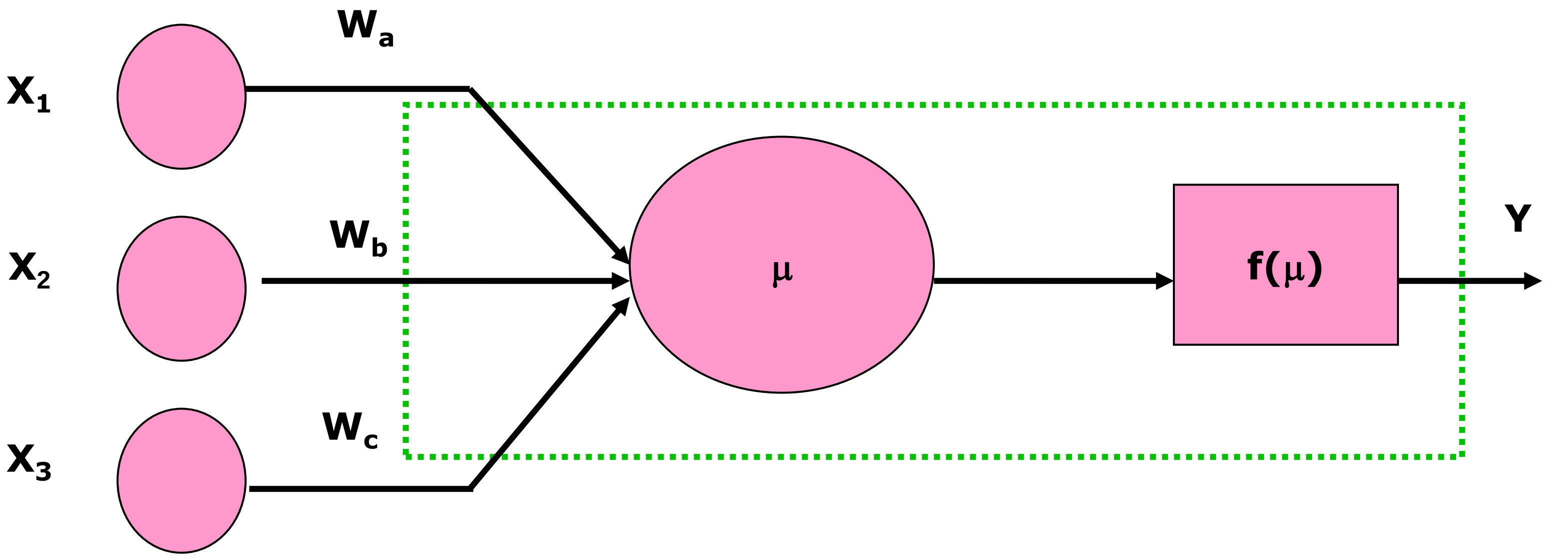
Artificial Neural Networks (ANN)

X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1



Output Y is 1 if at least two of the three inputs are equal to 1.

Model Of A Neuron



Input units

Connection weights

Summing function

computation

(dendrite)

(synapse)

(axon)

(soma)

- A neural net consists of a large number of simple processing elements called neurons, units, cells or nodes.
- Each neuron is connected to other neurons by means of directed communication links, each with associated weight.
- The weight represent information being used by the net to solve a problem.

- Each neuron has an internal state, called its activation or activity level, which is a function of the inputs it has received. Typically, a neuron sends its activation as a signal to several other neurons.

- It is important to note that a neuron can send only one signal at a time, although that signal is broadcast to several other neurons.

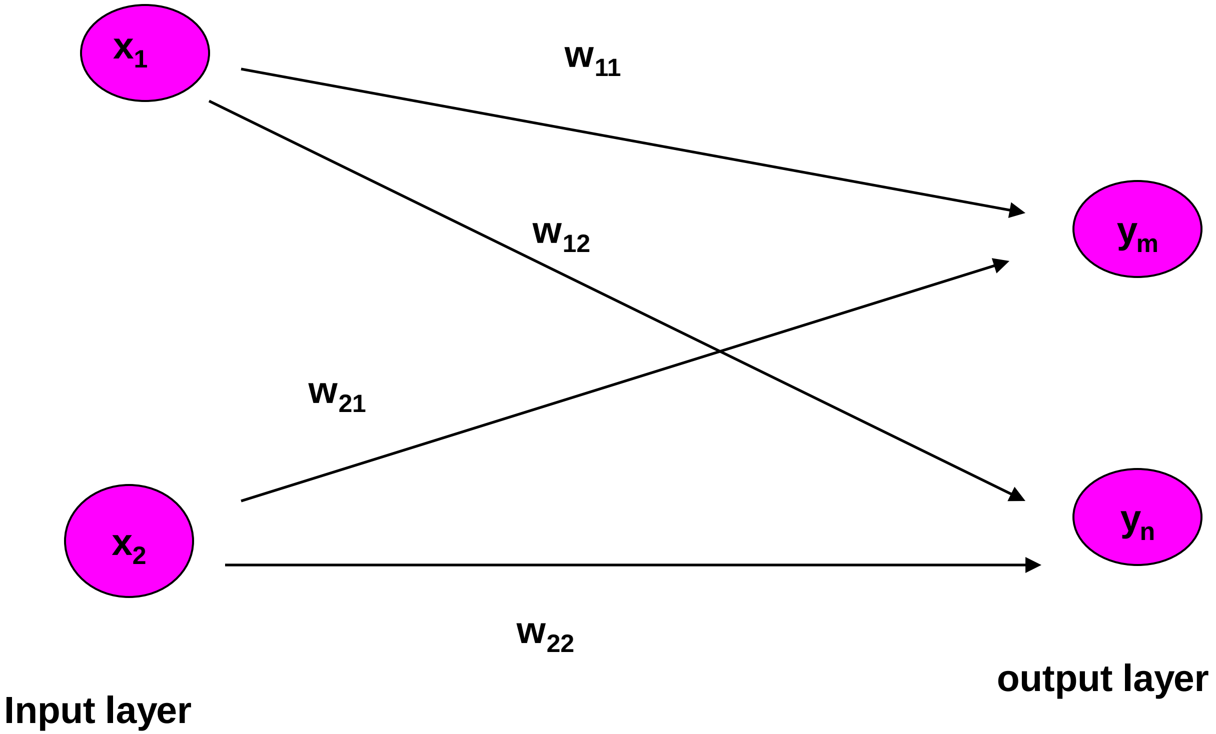
- Neural networks are configured for a specific application, such as pattern recognition or data classification, through a learning process
- In a biological system, learning involves adjustments to the synaptic connections between neurons

➔ same for artificial neural networks (ANNs)

Characterization

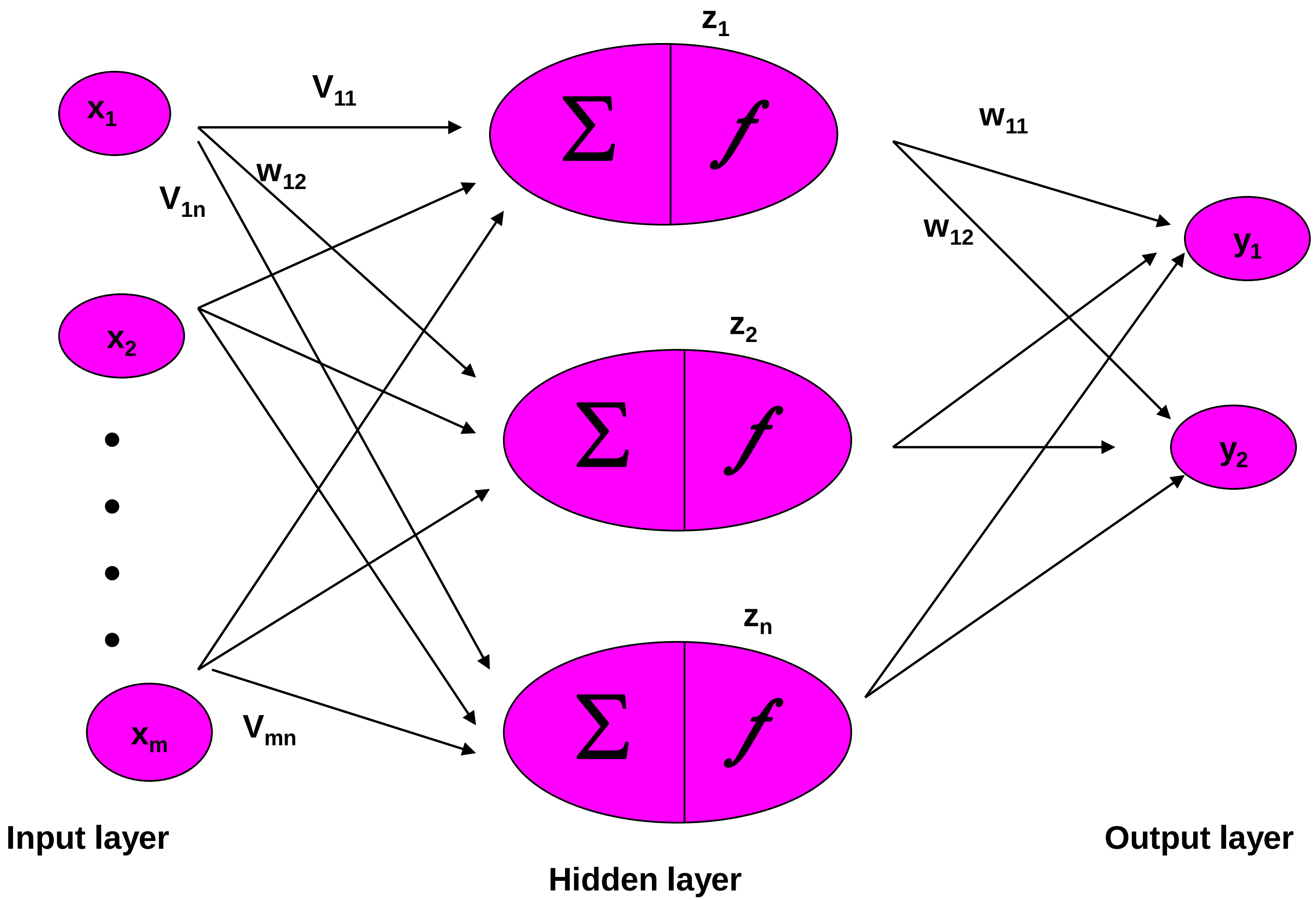
- Architecture
  - a pattern of connections between neurons
    - ◆Single Layer Feedforward
    - ◆Multilayer Feedforward
    - ◆Recurrent
- Strategy / Learning Algorithm
  - a method of determining the connection weights
    - ◆Supervised
    - ◆Unsupervised
    - ◆Reinforcement
- Activation Function
  - Function to compute output signal from input signal

Single Layer Feedforward NN



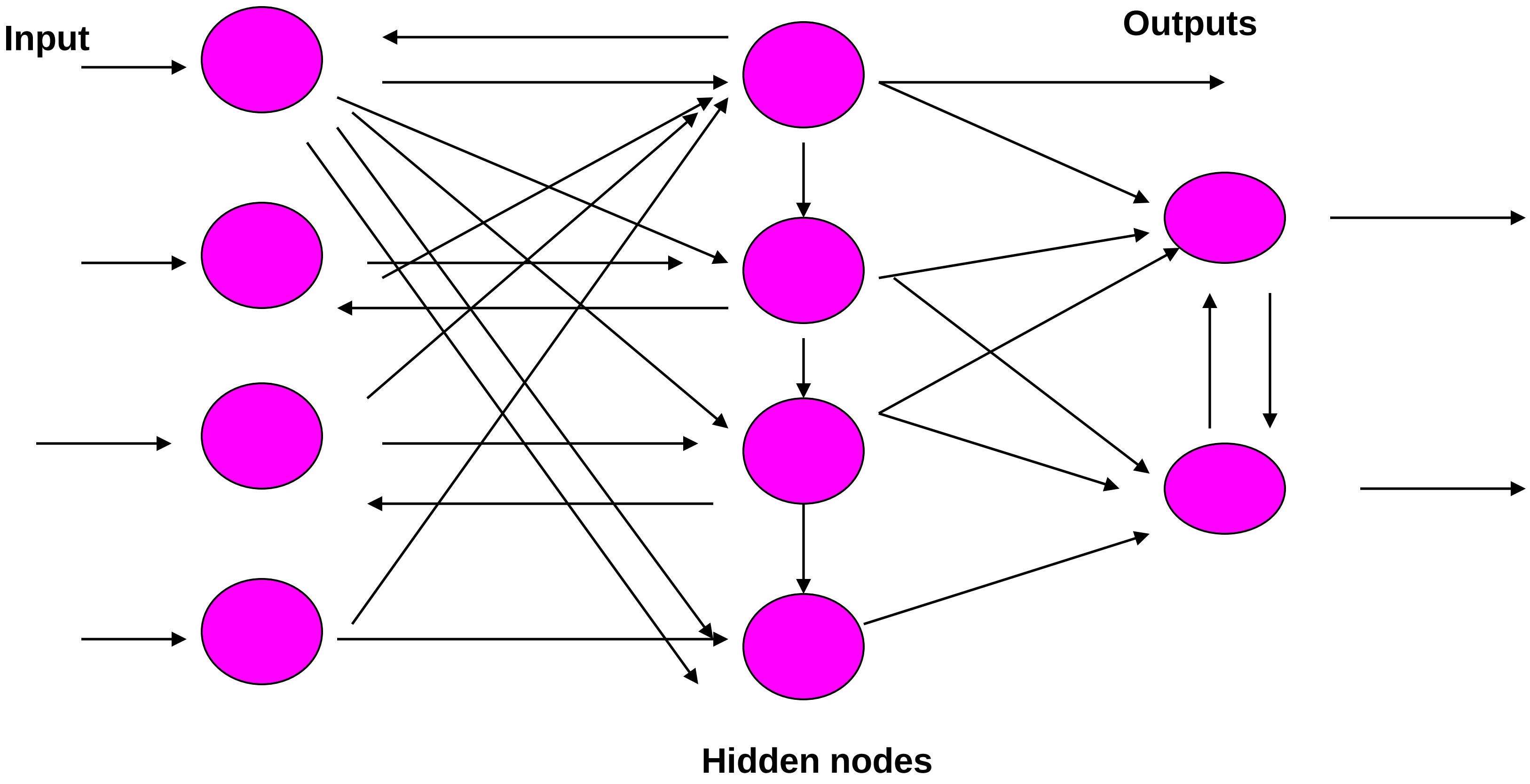
Contoh: **ADALINE, AM, Hopfield, LVQ, Perceptron, SOFM**

Multilayer Neural Network



Contoh: **CCN, GRNN, MADALINE, MLFF with BP, Neocognitron, RBF, RCE**

Recurrent NN



Contoh: **ART, BAM, BSB, Boltzman Machine, Cauchy Machine, Hopfield, RNN**

Strategy / Learning Algorithm

Supervised Learning

□ Learning is performed by presenting pattern with target

□ During learning, produced output is compared with the desired output

  – The difference between both output is used to modify learning weights according to the learning algorithm

□ Recognizing hand-written digits, pattern recognition and etc.

□ Neural Network models: perceptron, feed-forward, radial basis function, support vector machine.

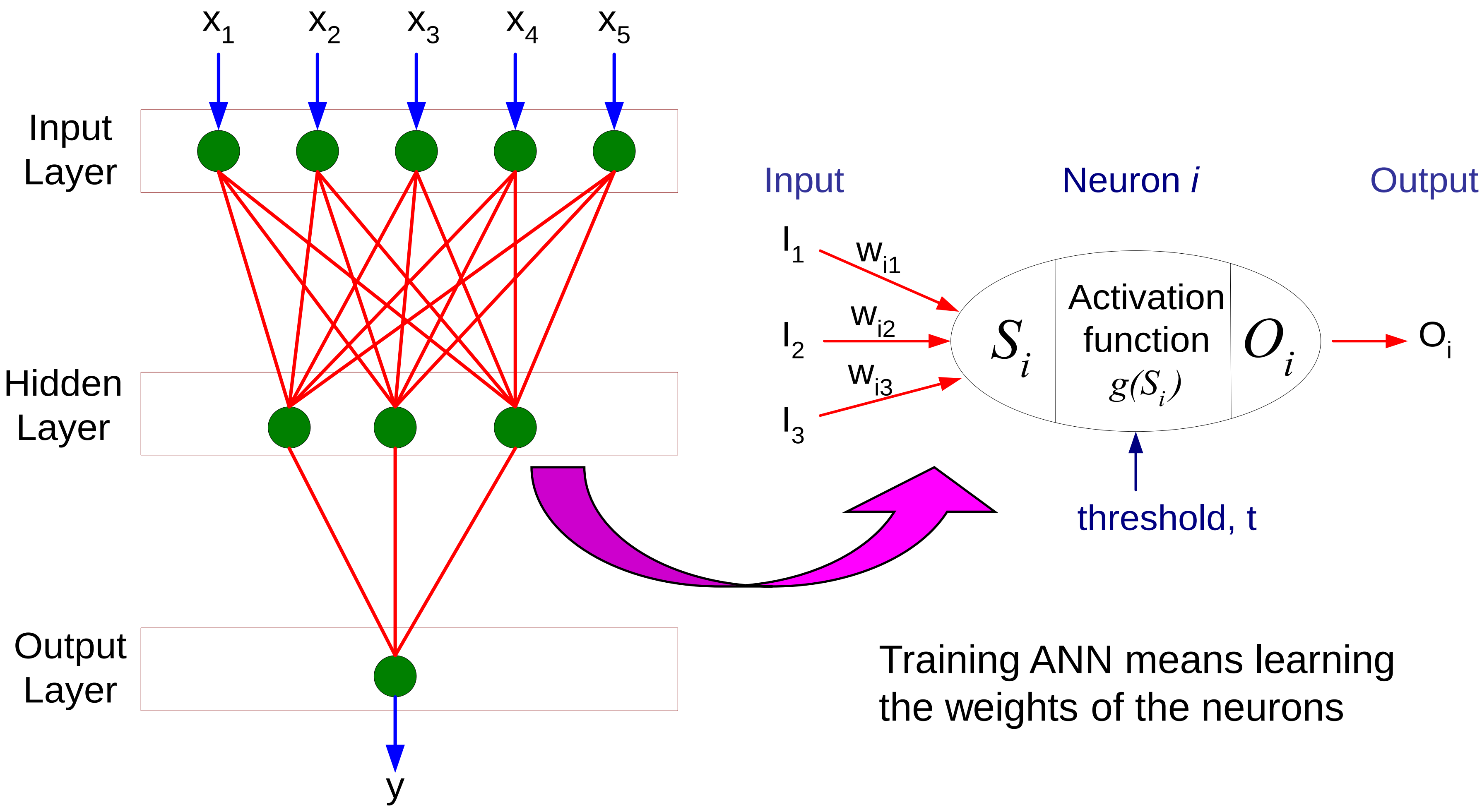
Unsupervised Learning

- Targets are not provided
- Appropriate for clustering task
    - Find similar groups of documents in the web, content addressable memory, clustering.
- Neural Network models: Kohonen, self organizing maps, Hopfield networks.

Reinforcement Learning

- Target is provided, but the desired output is absent.
- The net is only provided with guidance to determine the produced output is correct or vise versa.
- Weights are modified in the units that have errors

General Structure of ANN



Training ANN means learning the weights of the neurons

Artificial Neural Networks (ANN)

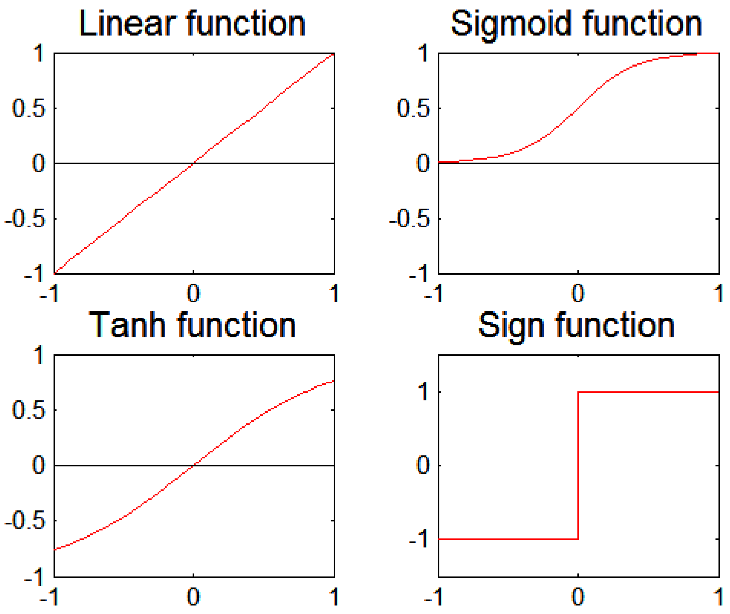
- Various types of neural network topology
  - single-layered network (perceptron) versus multi-layered network
  - Feed-forward versus recurrent network

Artificial Neural Networks (ANN)

- Various types of neural network topology
  - single-layered network (perceptron) versus multi-layered network
  - Feed-forward versus recurrent network

- Various types of activation functions (f)

$$Y = f\left(\sum_i w_i X_i\right)$$



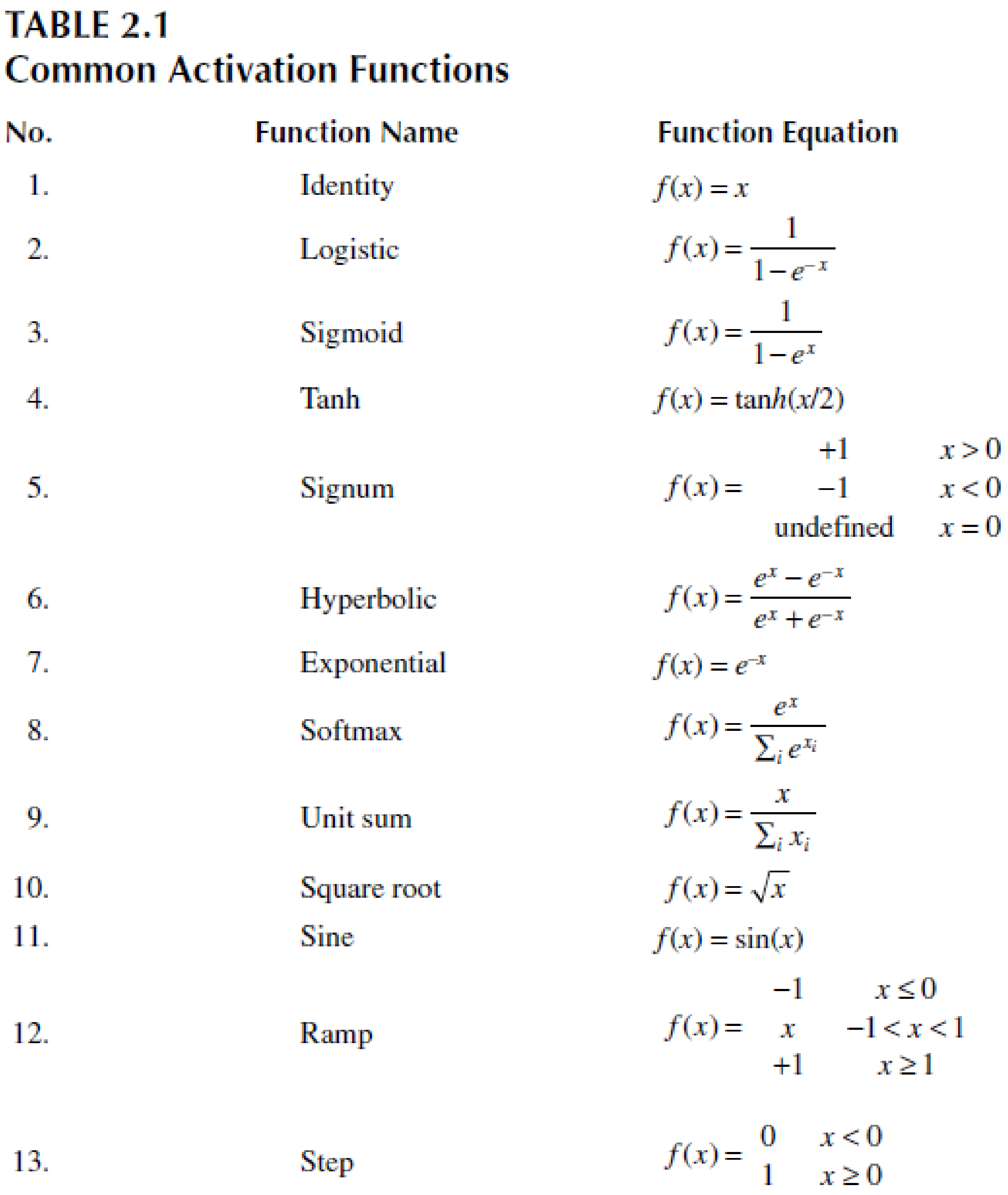
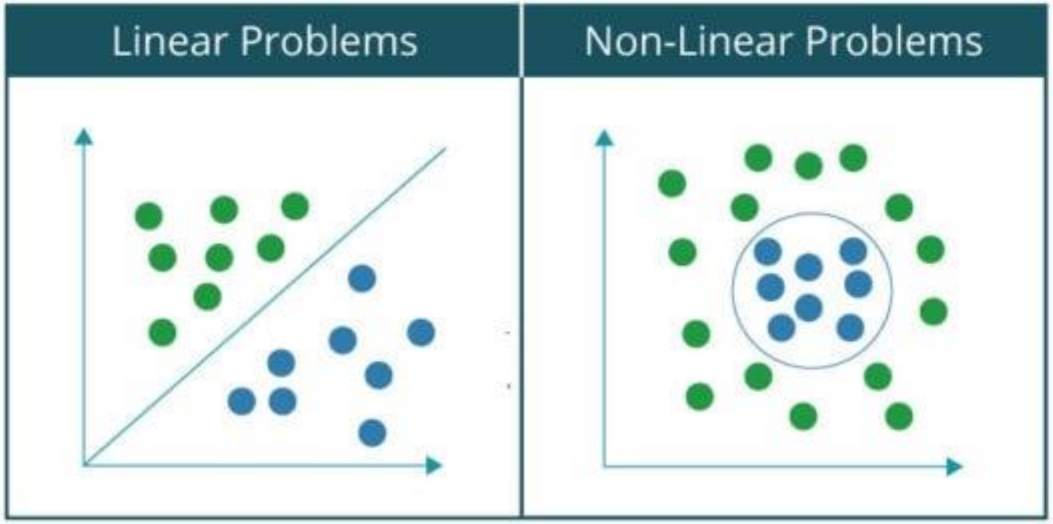


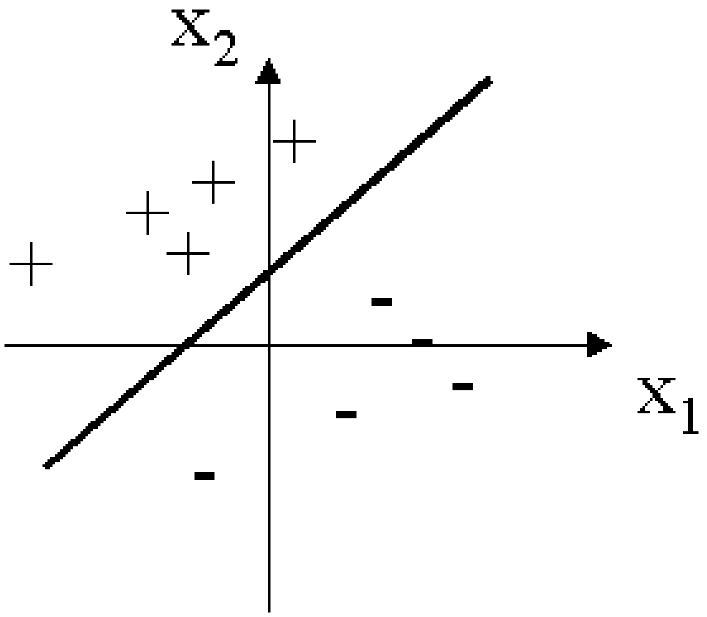
TABLE 2.1
Common Activation Functions

No.	Function Name	Function Equation
1.	Identity	$f(x) = x$
2.	Logistic	$f(x) = \dfrac{1}{1 - e^{-x}}$
3.	Sigmoid	$f(x) = \dfrac{1}{1 - e^{x}}$
4.	Tanh	$f(x) = \mathrm{tan}h(x/2)$
5.	Signum	$f(x) = \begin{matrix} +1 & x > 0 \\ -1 & x < 0 \\ \text{undefined} & x = 0 \end{matrix}$
6.	Hyperbolic	$f(x) = \dfrac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$
7.	Exponential	$f(x) = e^{-x}$
8.	Softmax	$f(x) = \dfrac{e^{x}}{\sum_i e^{x_i}}$
9.	Unit sum	$f(x) = \dfrac{x}{\sum_i x_i}$
10.	Square root	$f(x) = \sqrt{x}$
11.	Sine	$f(x) = \sin(x)$
12.	Ramp	$f(x) = \begin{matrix} -1 & x \leq 0 \\ x & -1 < x < 1 \\ +1 & x \geq 1 \end{matrix}$
13.	Step	$f(x) = \begin{matrix} 0 & x < 0 \\ 1 & x \geq 0 \end{matrix}$

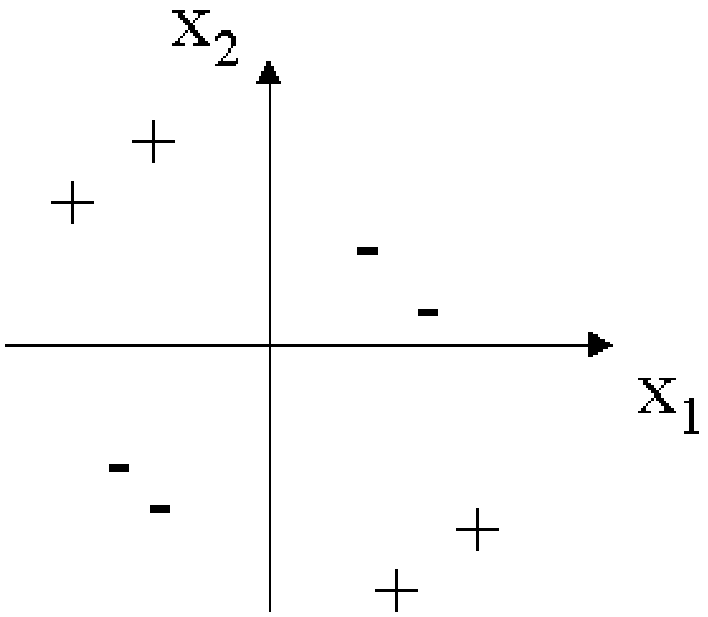
Types of Classification Problems

- One can categorize all kinds of classification problems that can be solved using neural networks into two broad categories:
  - **Linearly Separable Problems**
  - **Non-Linearly Separable Problems**





Linearly Separable

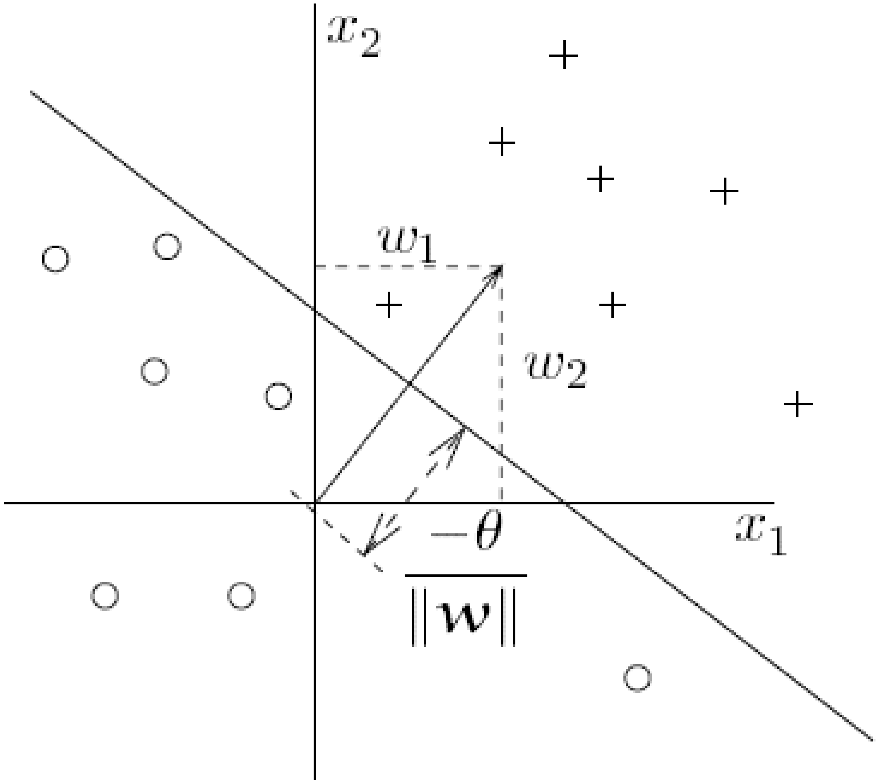


Not Linearly Separable

$$w_1 x_1 + w_2 x_2 + \theta = 0$$

- The bias is proportional to the offset of the plane from the origin
- The weights determine the slope of the line
- The weight vector is perpendicular to the plane

$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{\theta}{w_2}$$



Perceptron

- Single layer network
  - Contains only input and output nodes

- Activation function: f = sign(w•x)

- Applying model is straightforward

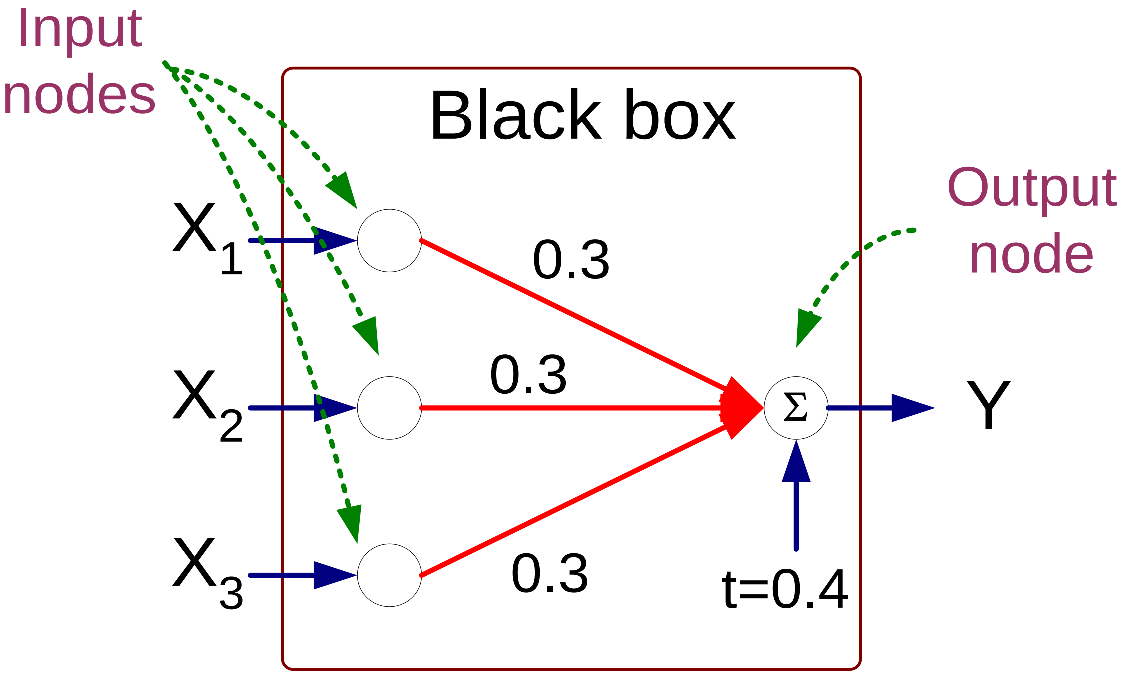
$$Y = sign(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)$$

$$\text{where } sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

  - $X_1 = 1$, $X_2 = 0$, $X_3 = 1$ => y = sign(0.2) = 1

Artificial Neural Networks (ANN)

X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1



$$Y = sign(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)$$

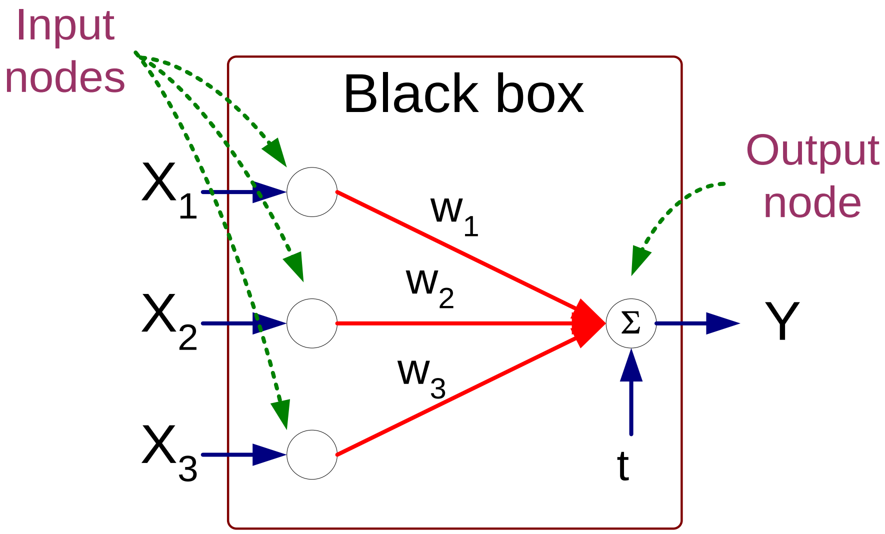
$$\text{where } sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Artificial Neural Networks (ANN)

- Model is an assembly of inter-connected nodes and weighted links

- Output node sums up each of its input value according to the weights of its links

- Compare output node against some threshold t



Perceptron Model

$$Y = sign\left(\sum_{i=1}^{d} w_i X_i - t\right)$$

$$= sign\left(\sum_{i=0}^{d} w_i X_i\right)$$

Perceptron Learning Algorithm

- We want to train the perceptron to classify inputs correctly
- Accomplished by adjusting the connecting weights and the bias
- Can only properly handle linearly separable sets
- We have a “training set” which is a set of input vectors used to train the perceptron.

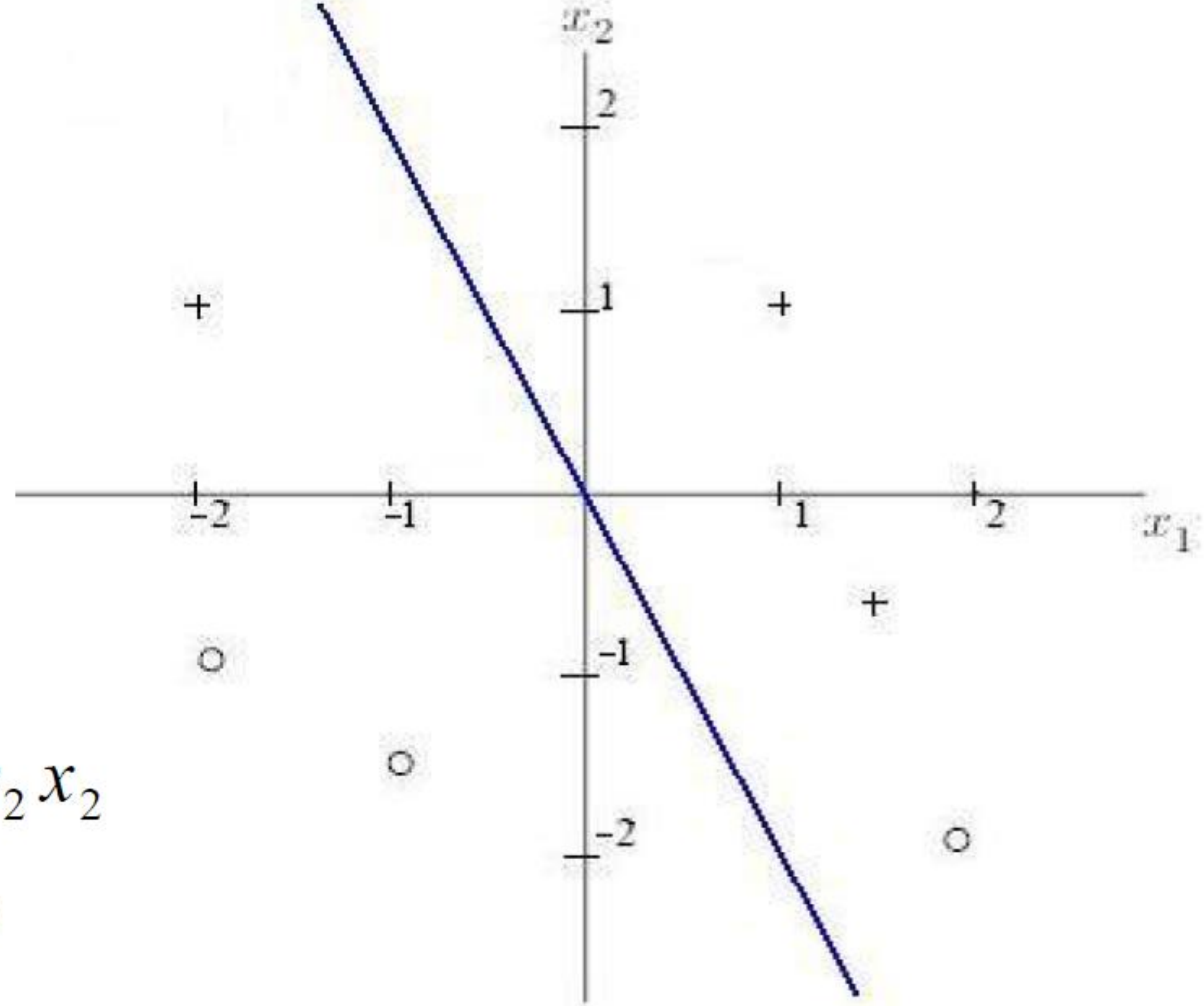
Learning Example

Initial Values:

$\eta = 0.2$

$$w = \begin{pmatrix} 0 \\ 1 \\ 0.5 \end{pmatrix}$$

$$\begin{aligned} 0 &= w_0 + w_1 x_1 + w_2 x_2 \\ &= 0 + x_1 + 0.5\mathrm{x}_2 \\ \Rightarrow \quad x_2 &= -2\mathrm{x}_1 \end{aligned}$$



Learning Algorithm

$\eta = 0.2$

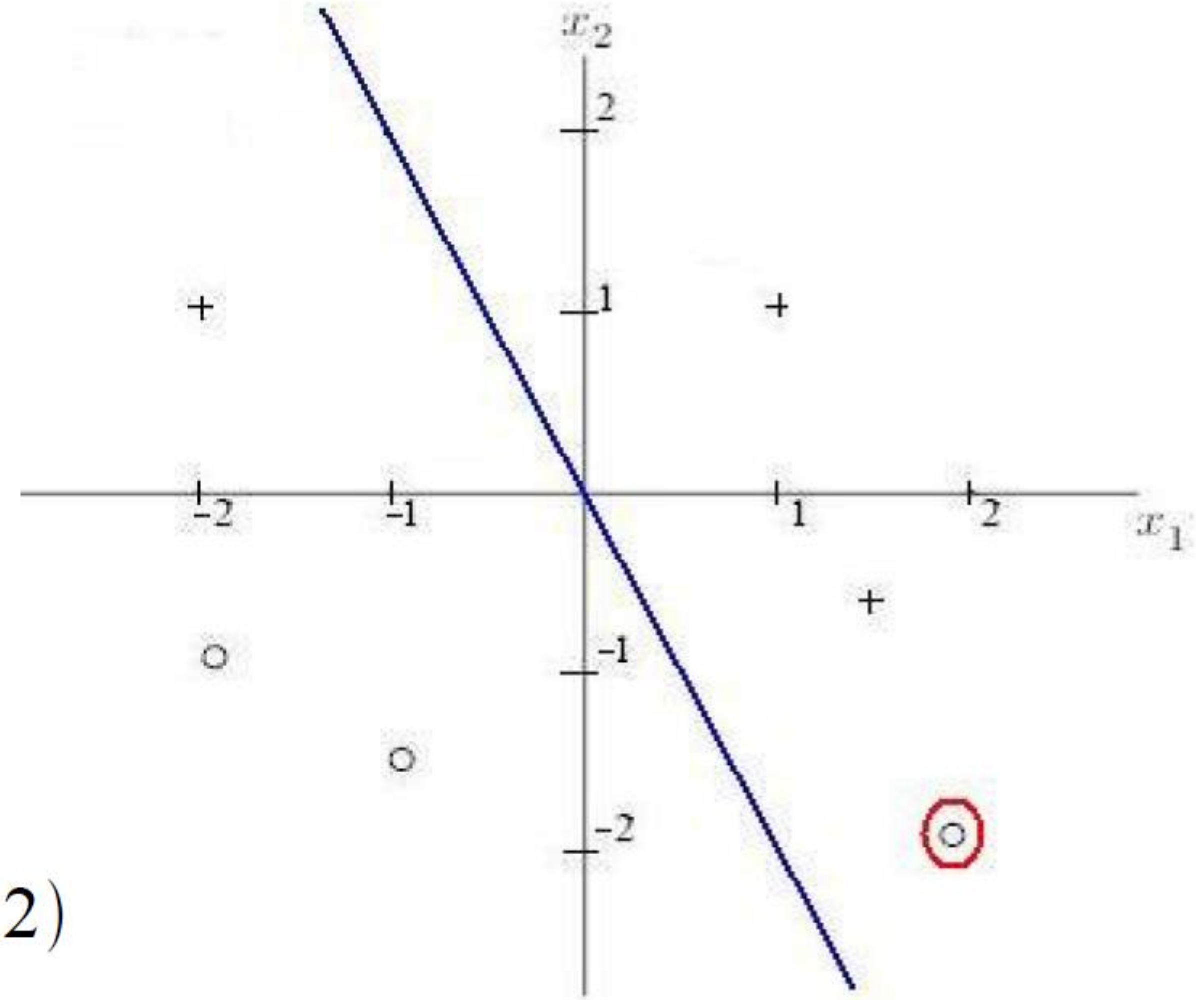
$$w = \begin{pmatrix} 0 \\ 1 \\ 0.5 \end{pmatrix}$$

$x_1 = 2, x_2 = -2$

$$w_0 = w_0 - 0.2 * 1$$

$$w_1 = w_1 - 0.2 * 2$$

$$w_2 = w_2 - 0.2 * (-2)$$

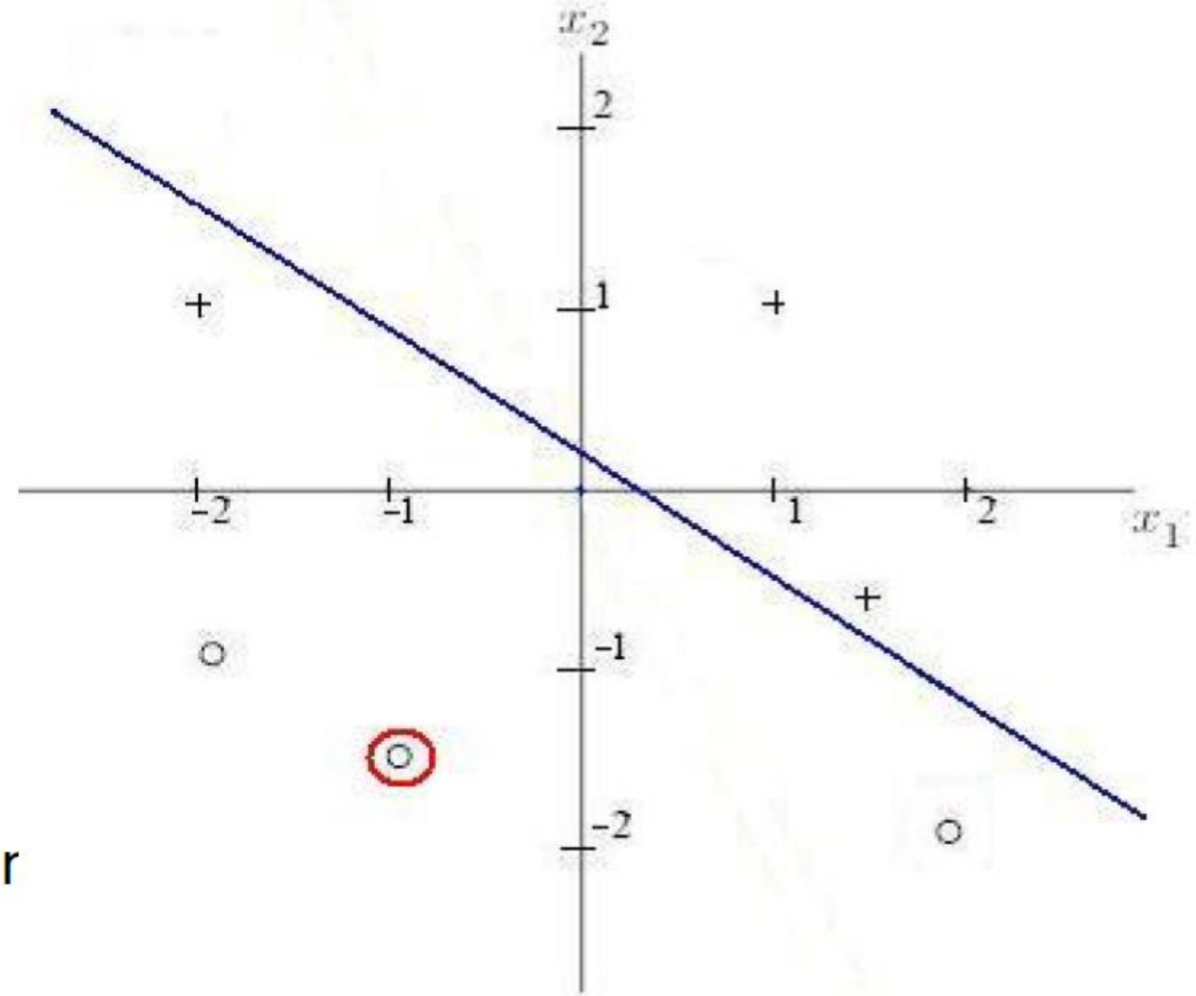


$\eta = 0.2$

$$w = \begin{pmatrix} -0.2 \\ 0.6 \\ 0.9 \end{pmatrix}$$

$x_1 = -1,\ x_2 = -1.5$
$w^T x < 0$

Correct classificatior
no action

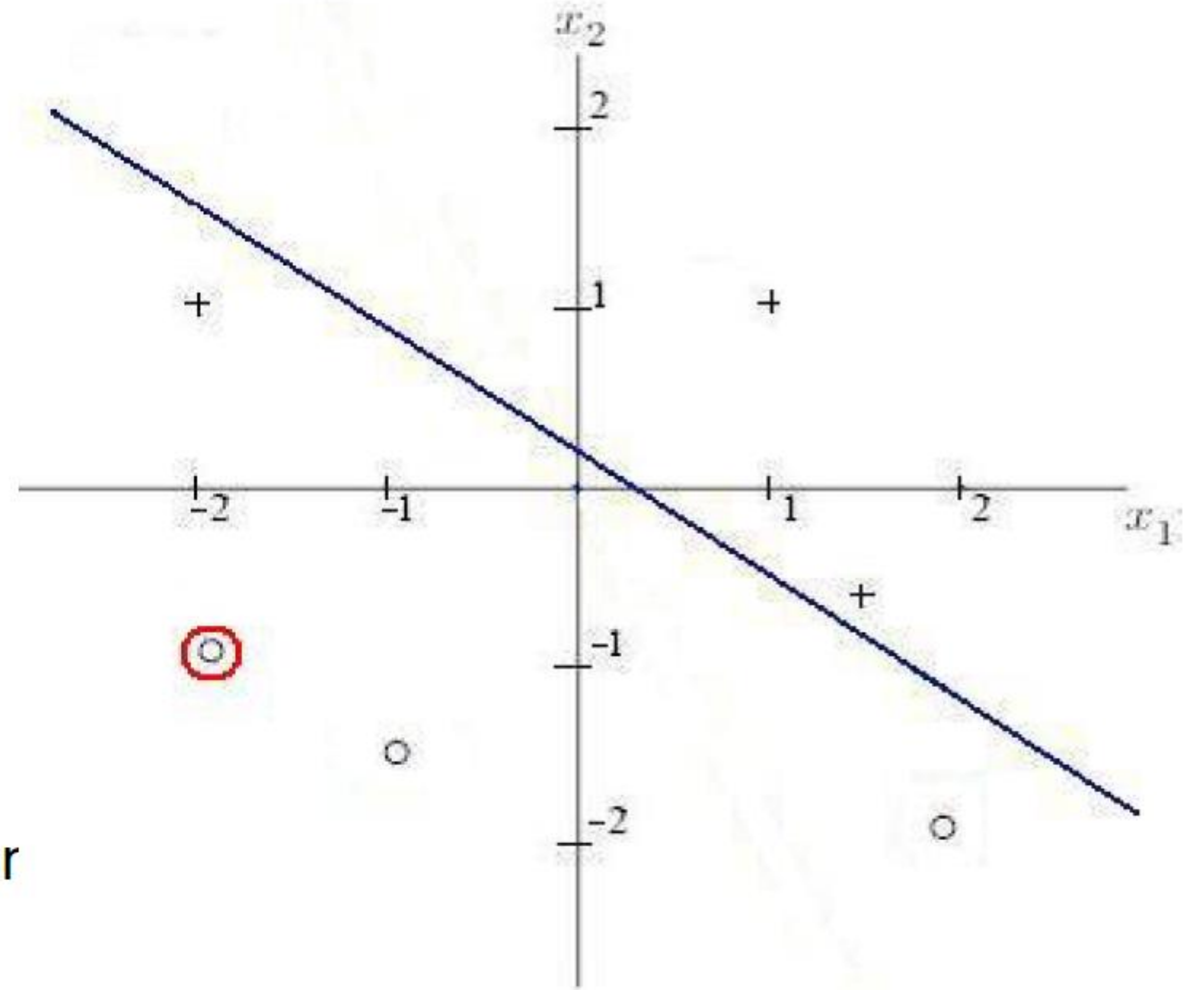


$\eta = 0.2$

$$w = \begin{pmatrix} -0.2 \\ 0.6 \\ 0.9 \end{pmatrix}$$

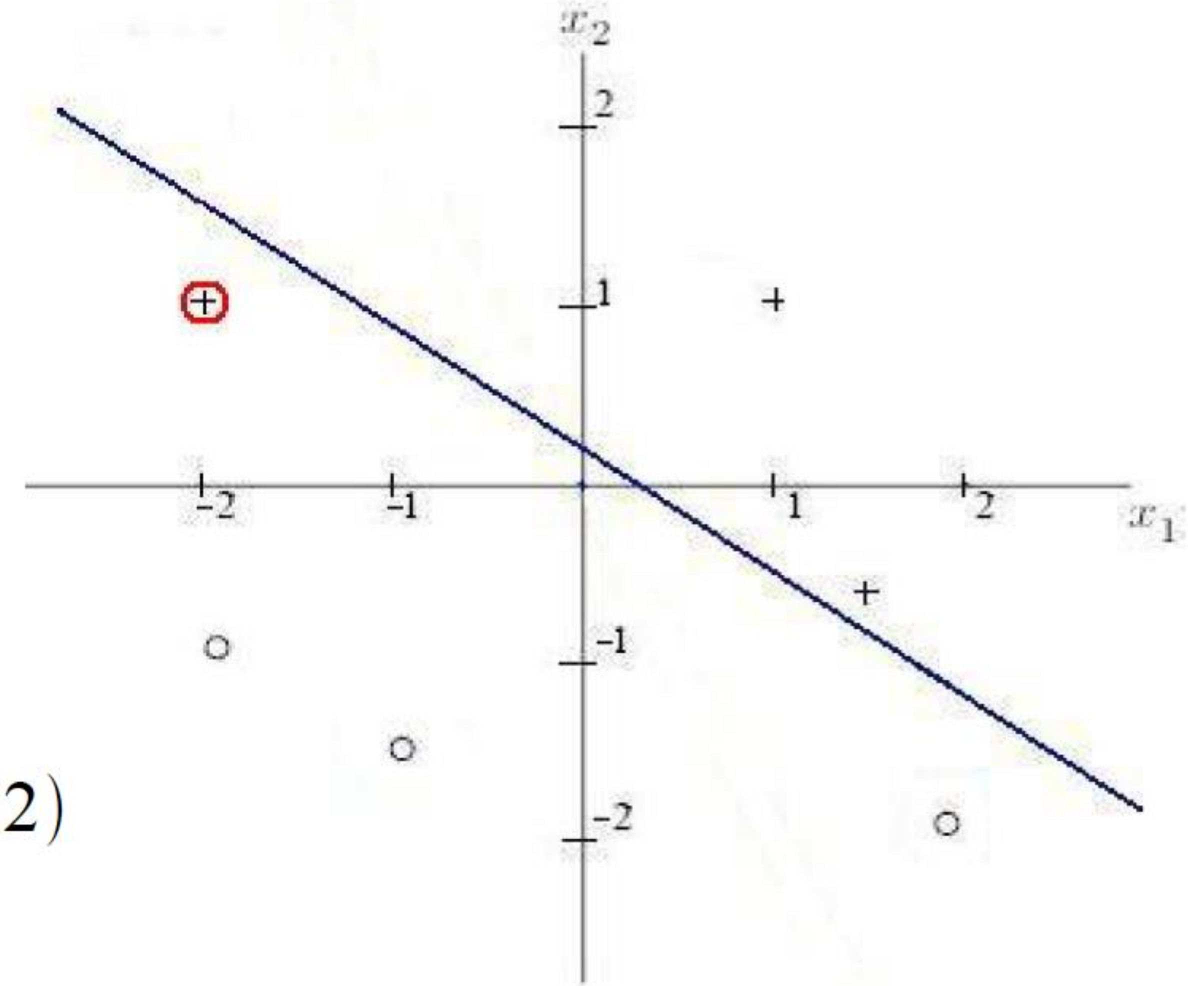
$x_1 = -2,\ x_2 = -1$
$w^T x < 0$

Correct classificatior
no action



$\eta = 0.2$

$$w = \begin{pmatrix} -0.2 \\ 0.6 \\ 0.9 \end{pmatrix}$$



x_1 = -2, x_2 = 1

$$w_0 = w_0 + 0.2 * 1$$

$$w_1 = w_1 + 0.2 * (-2)$$

$$w_2 = w_2 + 0.2 * 1$$

$$\eta = 0.2$$

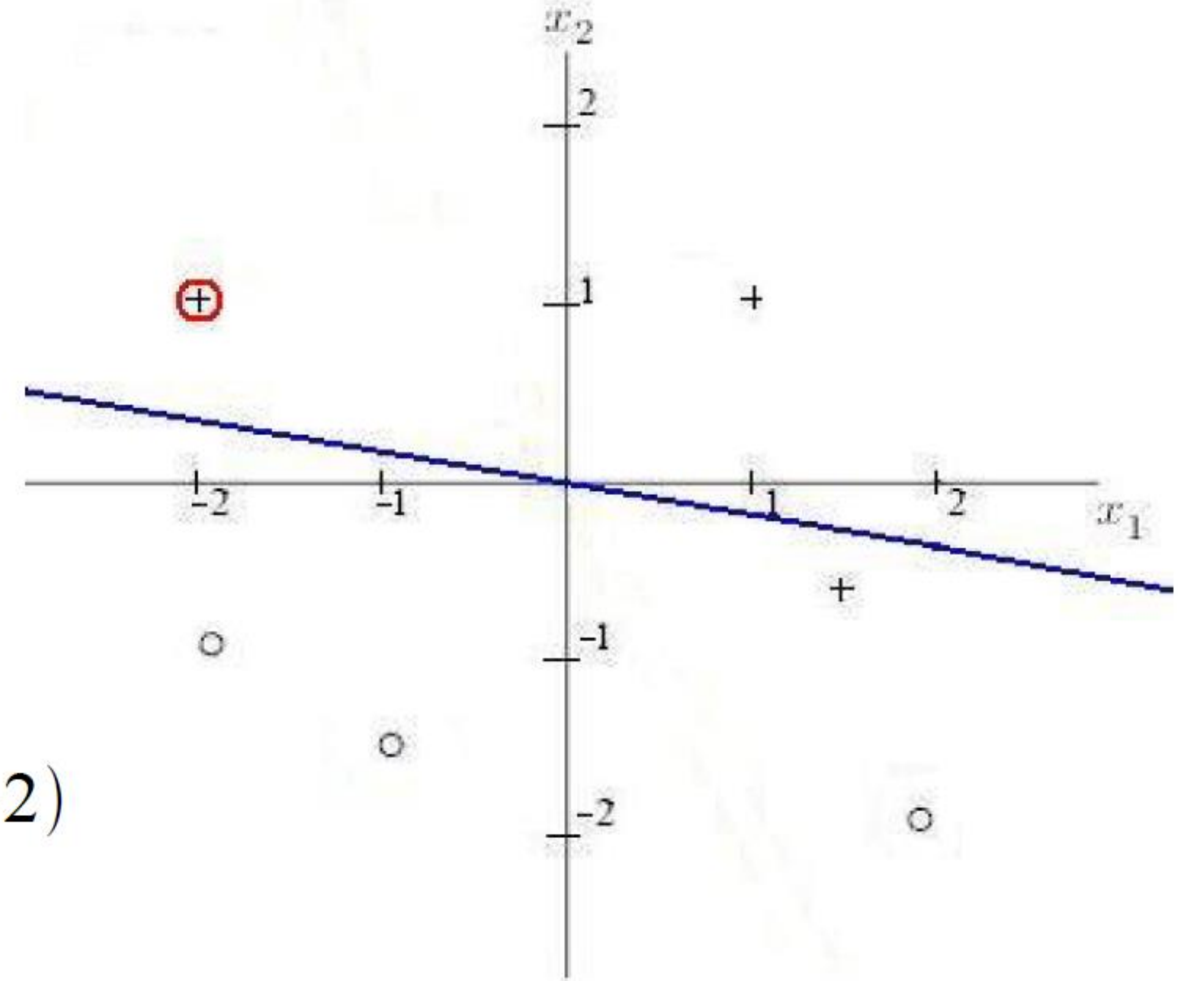
$$w = \begin{pmatrix} 0 \\ 0.2 \\ 1.1 \end{pmatrix}$$

$x_1 = -2,\ x_2 = 1$

$$w_0 = w_0 + 0.2 * 1$$

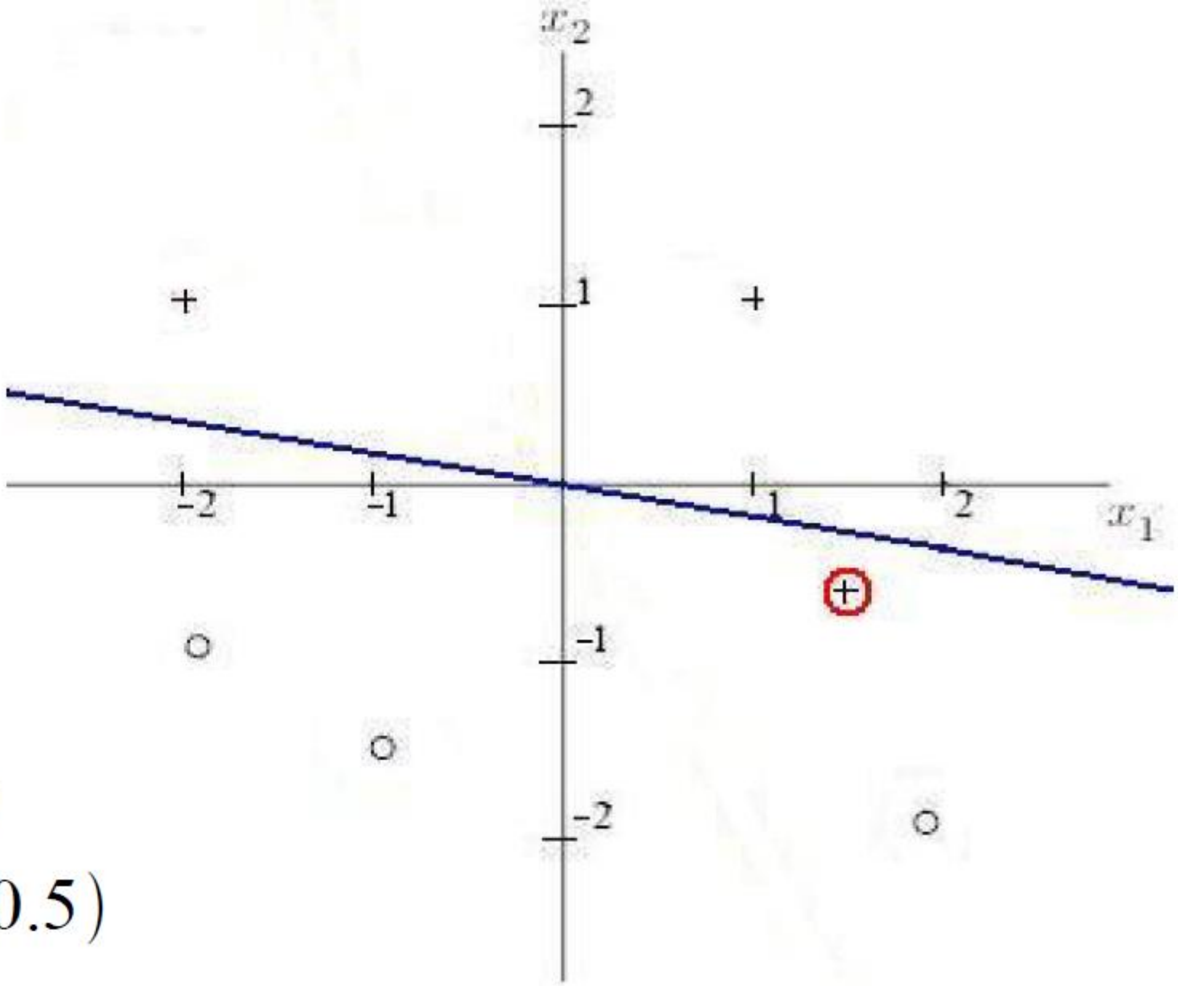
$$w_1 = w_1 + 0.2 * (-2)$$

$$w_2 = w_2 + 0.2 * 1$$



$\eta = 0.2$

$$w = \begin{pmatrix} 0 \\ 0.2 \\ 1.1 \end{pmatrix}$$



$x_1 = 1.5,\ x_2 = -0.5$

$$w_0 = w_0 + 0.2 * 1$$

$$w_1 = w_1 + 0.2 * 1.5$$

$$w_2 = w_2 + 0.2 * (-0.5)$$

$\eta = 0.2$

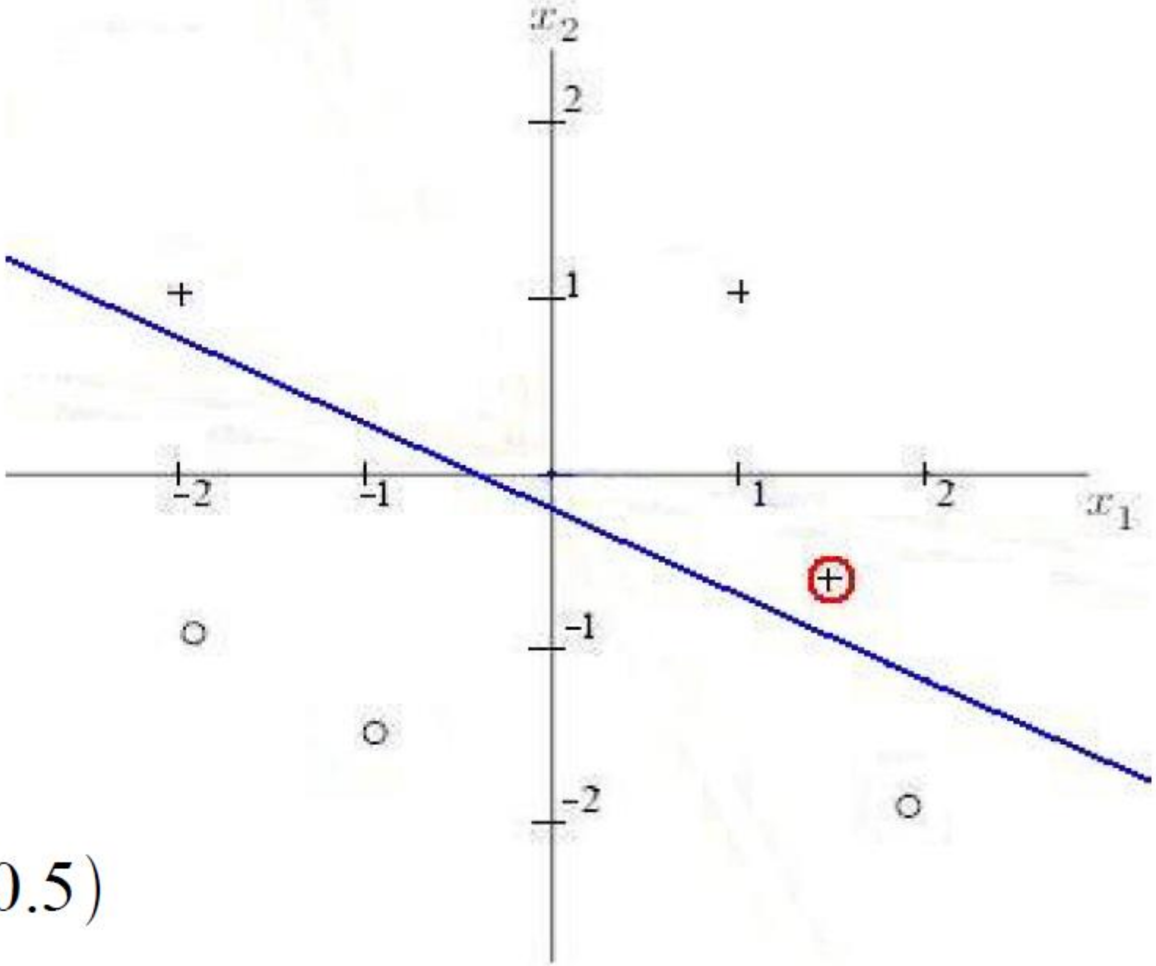
$$w = \begin{pmatrix} 0.2 \\ 0.5 \\ 1 \end{pmatrix}$$

$x_1 = 1.5$, $x_2 = -0.5$

$$w_0 = w_0 + 0.2 * 1$$

$$w_1 = w_1 + 0.2 * 1.5$$

$$w_2 = w_2 + 0.2 * (-0.5)$$



Perceptron Learning Rule

- Initialize the weights (w_0, w_1, …, w_d)
- Repeat
  - For each training example (x_i, y_i)
    - Compute f(w, x_i)
    - Update the weights:

$$w^{(k+1)} = w^{(k)} + \lambda\left[y_i - f(w^{(k)}, x_i)\right]x_i$$

- Until stopping condition is met

Perceptron Learning Rule

- Weight update formula:

$$w^{(k+1)} = w^{(k)} + \lambda\left[y_i - f(w^{(k)}, x_i)\right]x_i \;\; ; \;\; \lambda : \text{learning rate}$$

- Intuition:
  - Update weight based on error: $e = \left[y_i - f(w^{(k)}, x_i)\right]$
  - If y=f(x,w), e=0: no update needed
  - If y>f(x,w), e=1: weight must be increased so that f(x,w) will increase
  - If y<f(x,w), e=-1: weight must be decreased so that f(x,w) will decrease

Example of Perceptron Learning

$$w^{(k+1)} = w^{(k)} + \lambda\left[y_i - f(w^{(k)}, x_i)\right]x_i$$

$$Y = sign(\sum_{i=0}^{d} w_i X_i)$$

$$\boldsymbol{\lambda = 0.2}$$

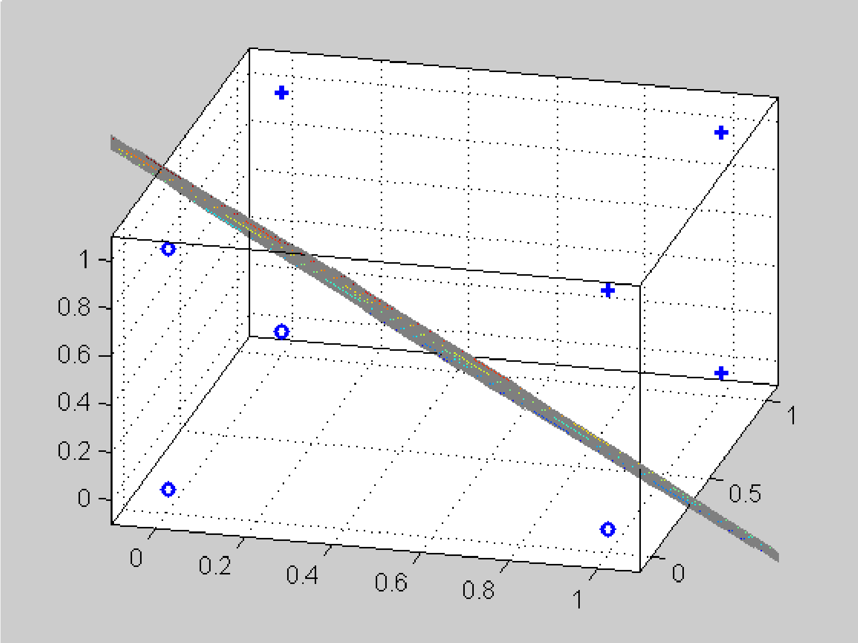
X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1

	w_0	w_1	w_2	w_3
0	0	0	0	0
1	-0.2	-0.2	0	0
2	0	0	0	0.2
3	0	0	0	0.2
4	0	0	0	0.2
5	-0.2	0	0	0
6	-0.2	0	0	0
7	0	0	0.2	0.2
8	-0.2	0	0.2	0.2

Epoch	w_0	w_1	w_2	w_3
0	0	0	0	0
1	-0.2	0	0.2	0.2
2	-0.2	0	0.4	0.2
3	-0.4	0	0.4	0.2
4	-0.4	0.2	0.4	0.4
5	-0.6	0.2	0.4	0.2
6	-0.6	0.4	0.4	0.2

Perceptron Learning Rule

- Since f(w,x) is a linear combination of input variables, decision boundary is linear



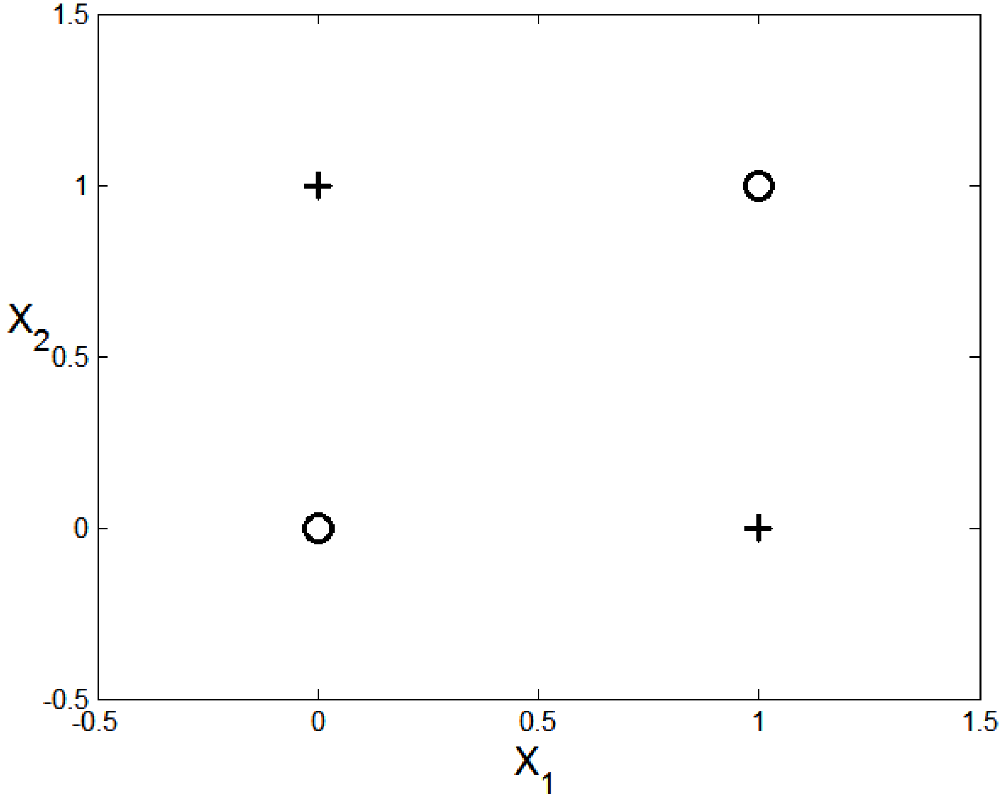
- For nonlinearly separable problems, perceptron learning algorithm will fail because no linear hyperplane can separate the data perfectly

Nonlinearly Separable Data

$$y = x_1 \oplus x_2$$

x_1	x_2	y
0	0	-1
1	0	1
0	1	1
1	1	-1

XOR Data



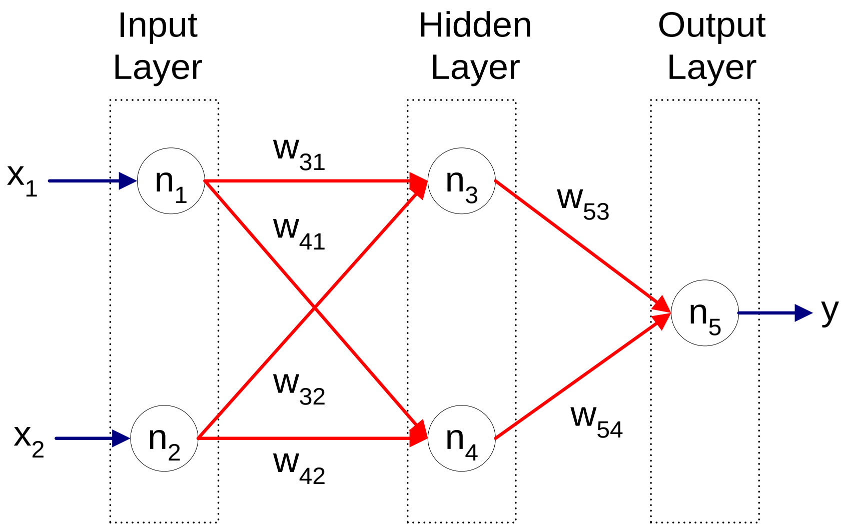
Multilayer Neural Network

- Hidden layers
  - intermediary layers between input & output layers

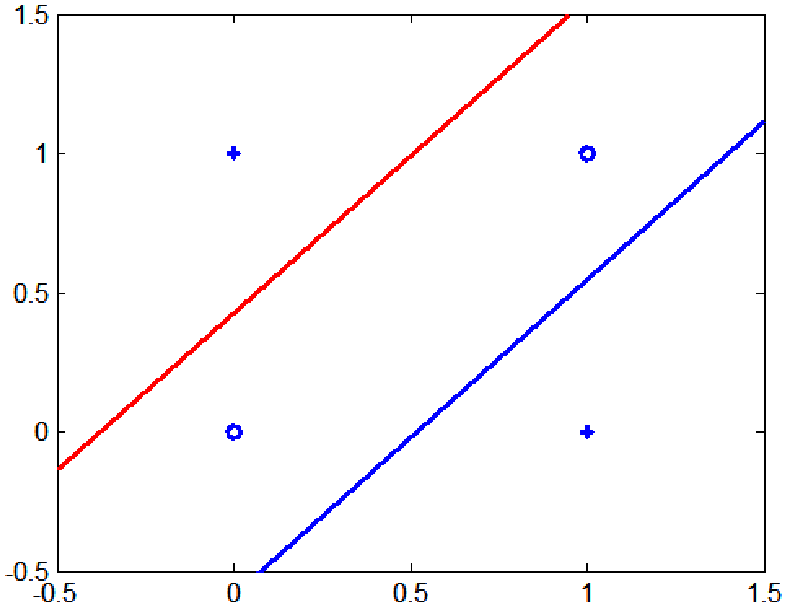
- More general activation functions (sigmoid, linear, etc)

Multi-layer Neural Network

- Multi-layer neural network can solve any type of classification task involving nonlinear decision surfaces



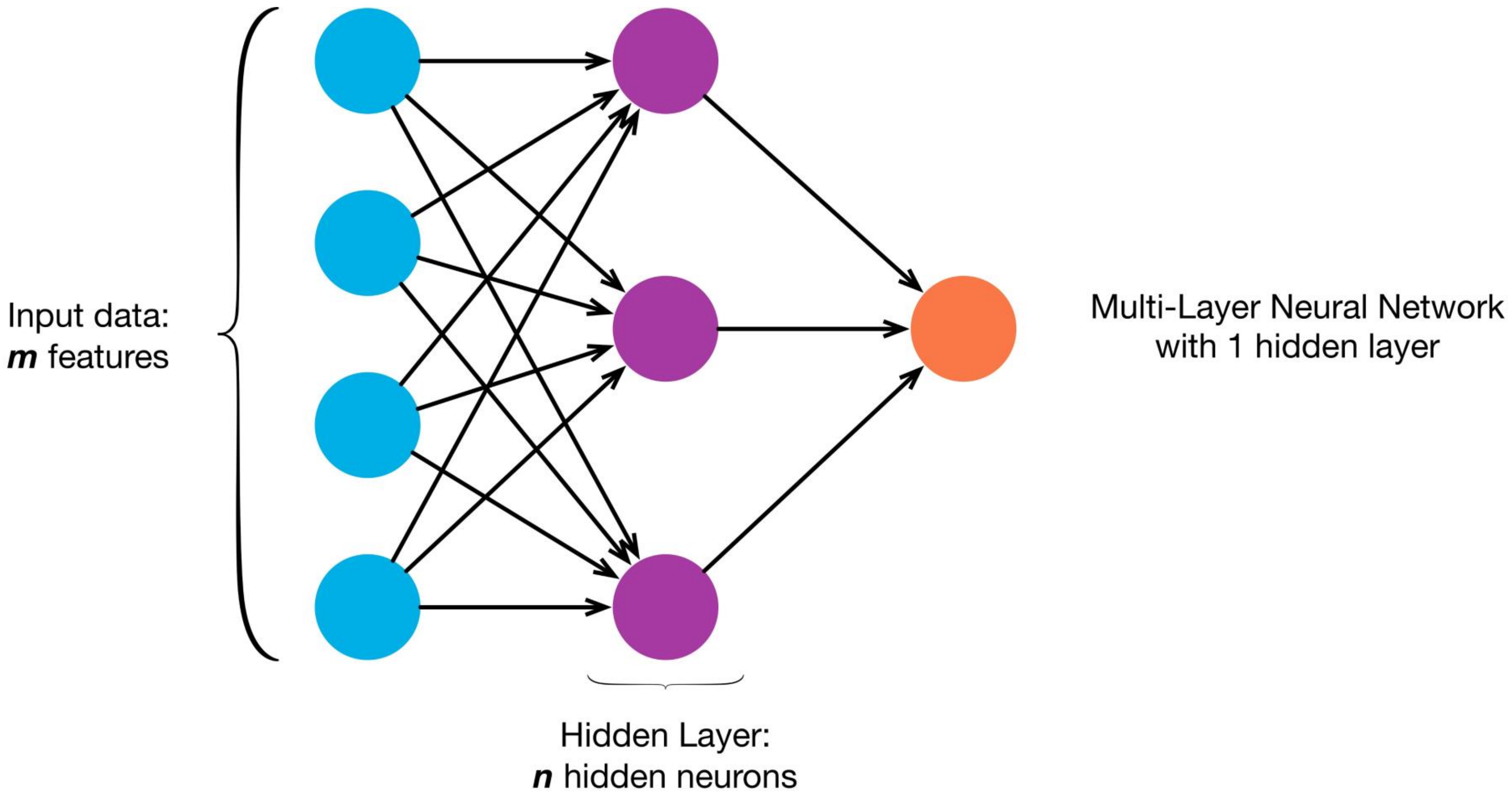
XOR Data



Learning Multi-layer Neural Network

- Can we apply perceptron learning rule to each node, including hidden nodes?
  - Perceptron learning rule computes error term e = y-f(w,x) and updates weights accordingly
    - Problem: how to determine the true value of y for hidden nodes?
  - Approximate error in hidden nodes by error in the output nodes
    - Problem:
      - Not clear how adjustment in the hidden nodes affect overall error
      - No guarantee of convergence to optimal solution

Multi-layer Network

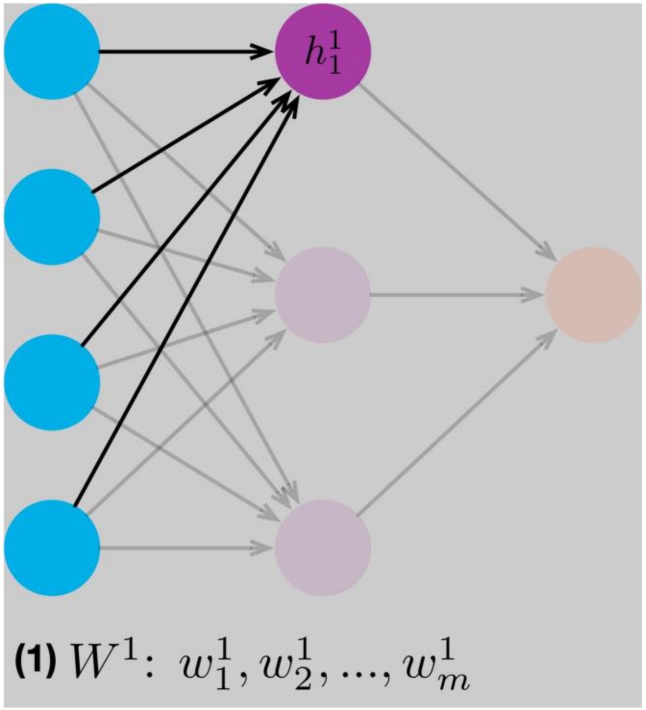


Perform dot product for input data set $X : x_1, x_2, ..., x_m$ with weight set W^1: $w_1^1, w_2^1, ..., w_m^1$, (note the weights have superscript 1 since they are weights to hidden layer h^1's first hidden neuron h_1^1), with the dot product summation, add bias so we have a result z:

$$z = \sum_{i=1}^{m} w_i x_i + bias$$

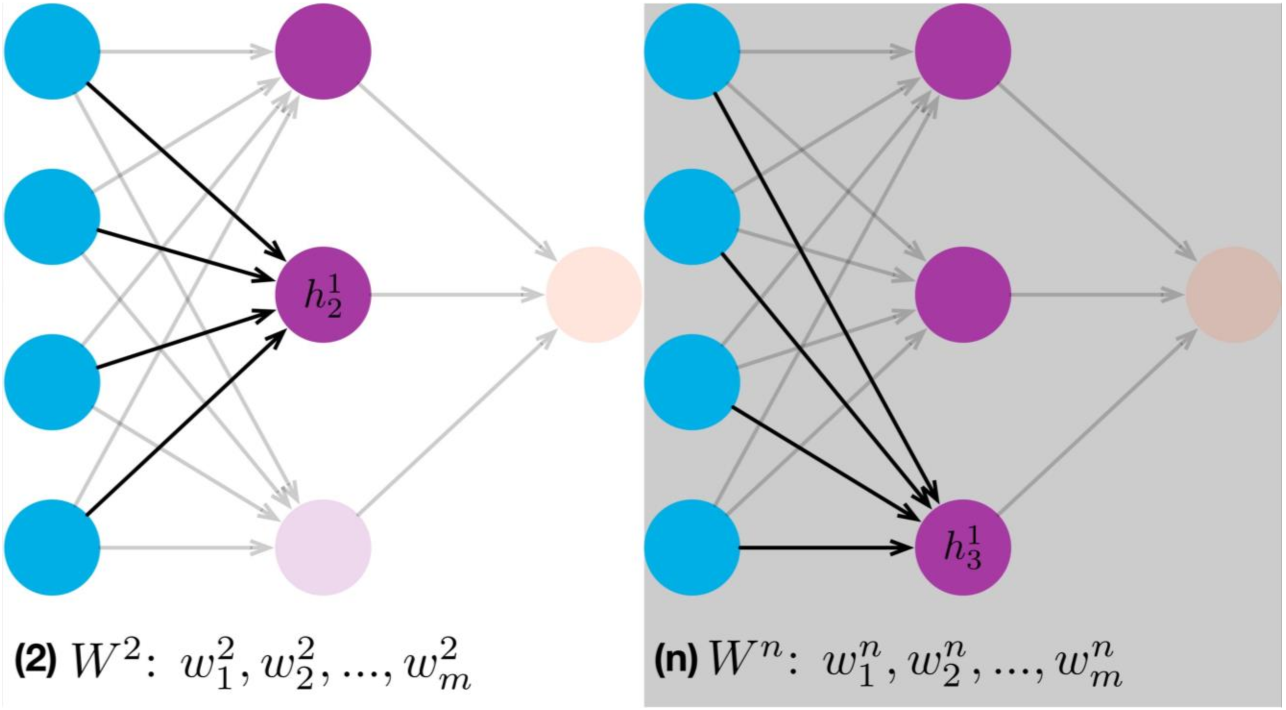
Feed z into an activation function $f(z)$ so that we get an output for the hidden layer h^1's first neuron h_1^1:

$$h_1^1 = f(z), \ (note\ that\ f()\ needs\ not\ be\ step\ function)$$



Repeat the same with input data and weights to the second hidden neuron, which consists of W^2: $w_1^2, w_2^2, ..., w_m^2$, and then we will get the output for the second hidden neuron in hidden layer h^1: h_2^1

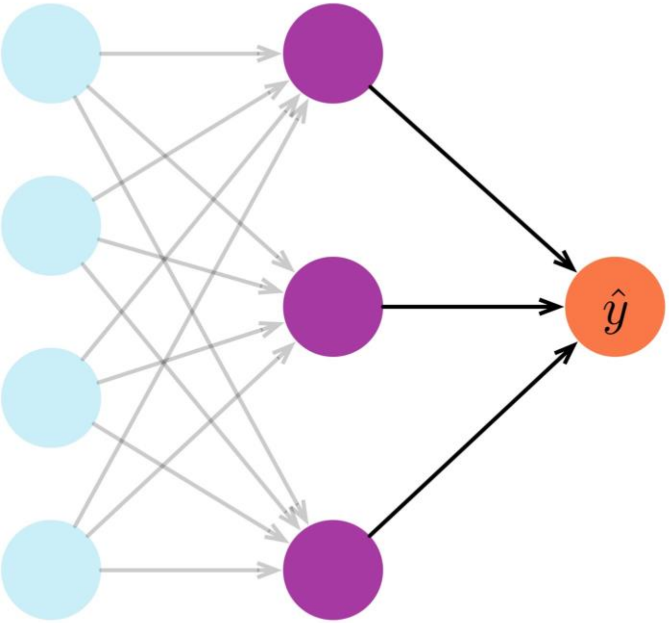
Repeat again and again until the last weight set W^n using input data with the nth set W^n: $w_1^n, w_2^n, ..., w_m^n$ for the third hidden neuron in hidden layer h^1 to get h_n^1, so in total we have n hidden outputs: $h^1 : (h_1^1, h_2^1, ..., h_n^1)$



(2) W^2: $w_1^2, w_2^2, ..., w_m^2$

(n) W^n: $w_1^n, w_2^n, ..., w_m^n$

Now the hidden layer outputs are calculated, we use them as inputs to calculate the final output.



$W_1^h : (w_1^{h_1}, w_2^{h_1}, ..., w_n^{h_1})$

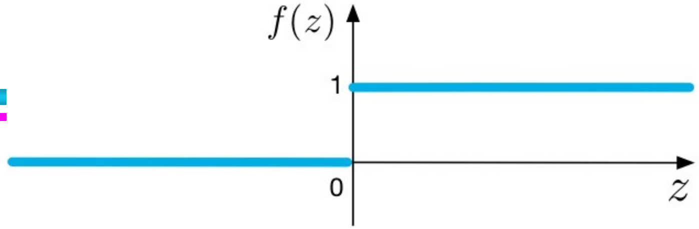
For the final output, perform a dot product for hidden layer outputs (which we now consider as inputs) and hidden layer weights W^h. Remember that the inputs are hidden layer outputs $h^1 : (h_1^1, h_2^1, h_3^1, ..., h_n^1)$. Since we have only 1 output, our set of weights consist of a W_1^h, which is $W_1^h : (w_1^{h_1}, w_2^{h_1}, ..., w_n^{h_1})$ with n weights since we have n hidden layer inputs. Then add the sum of dot product with bias to obtain a result z:

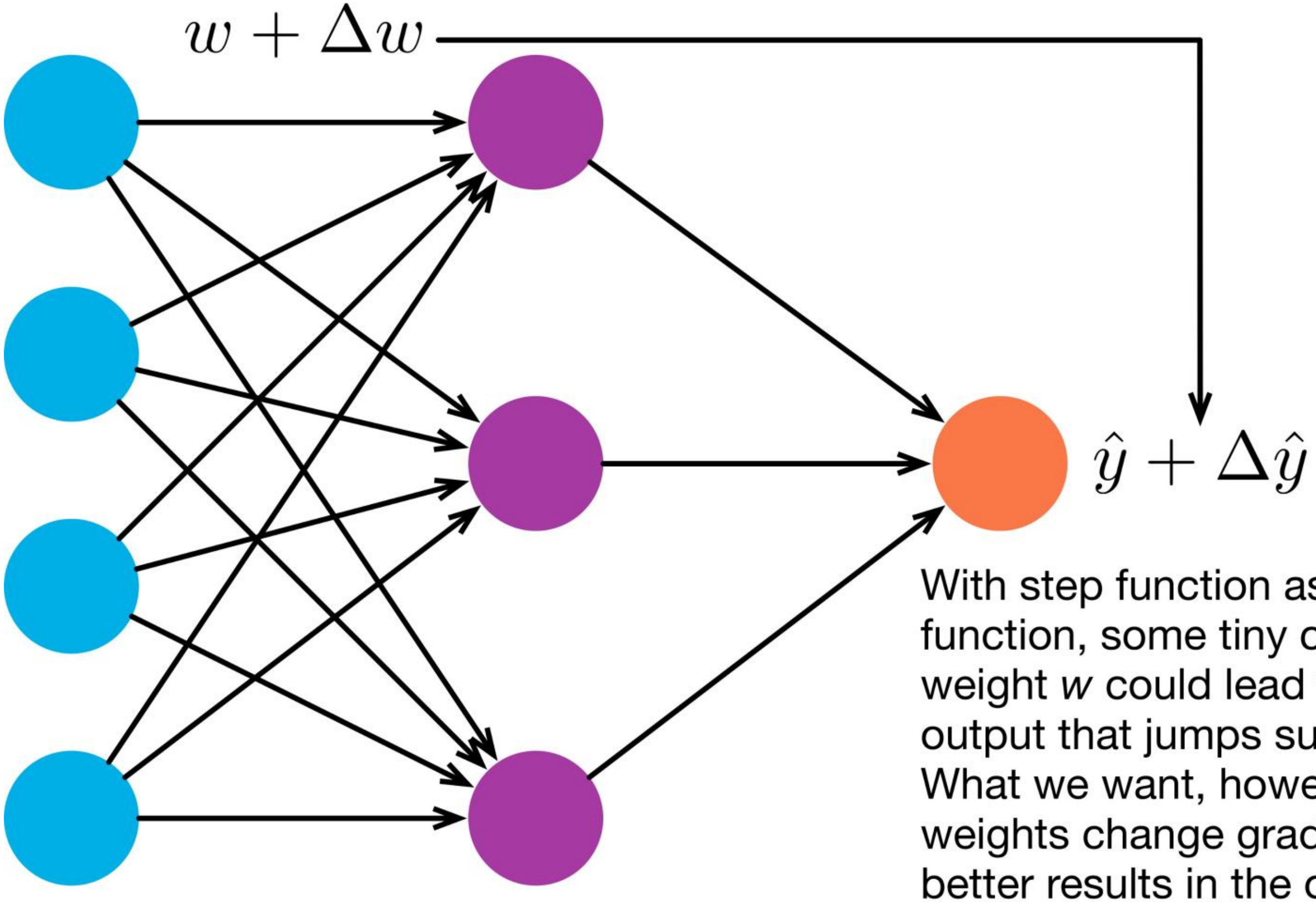
$$z = \sum_{i=1}^{n} w_i^{h_1} h_i^1 + bias$$

Feed z into an activation function $f(z)$ so that we get an output for the output layer:

$$\hat{y} = f(z)$$

Now you understand fully how a perceptron with multiple layers work



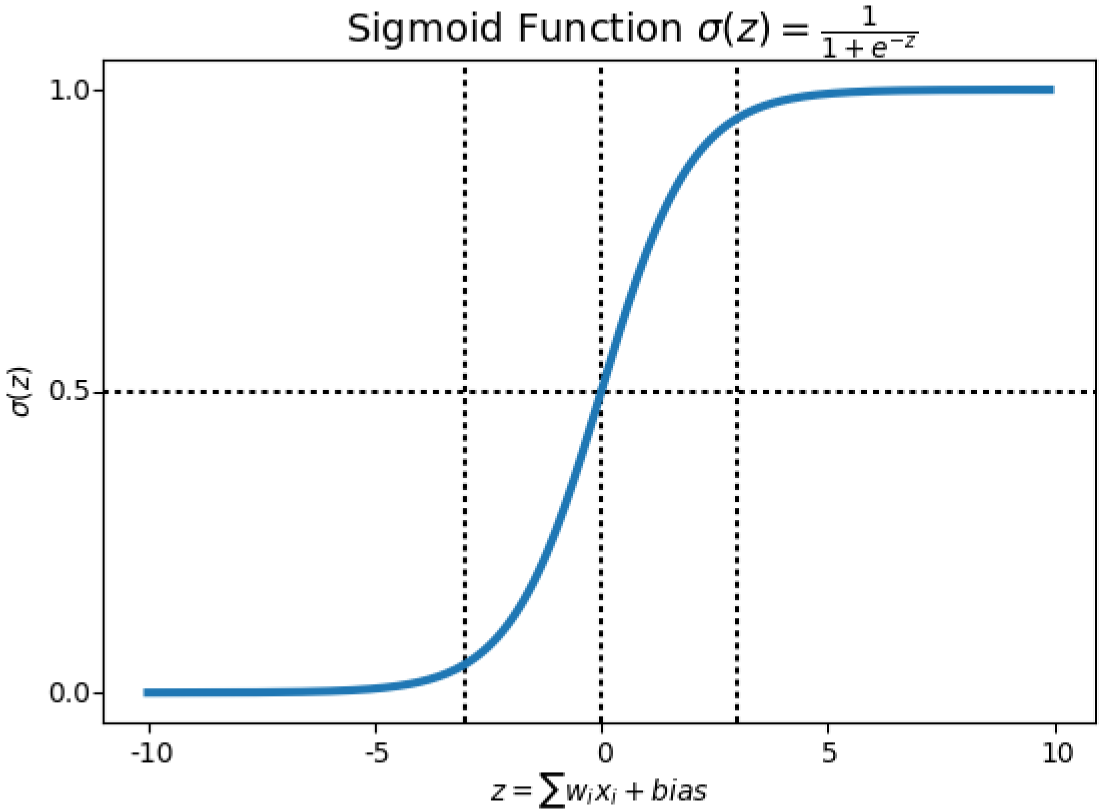


With step function as our activation function, some tiny changes in any weight *w* could lead to the perceptron output that jumps suddenly from 0 to 1. What we want, however, is to have the weights change gradually to produce better results in the output.

Sigmoid Function

$$z = \sum_{i=1}^{m} w_i x_i + bias$$

Sigmoid Function is: $\sigma(z) = \frac{1}{1+e^{-z}}$



Gradient Descent for Multilayer NN

- Weight update: $w_j^{(k+1)} = w_j^{(k)} - \lambda \frac{\partial E}{\partial w_j}$
- Error function: $E = \frac{1}{2}\sum_{i=1}^{N}\left(t_i - f(\sum_j w_j x_{ij})\right)$
- Activation function f must be differentiable
- For sigmoid function:

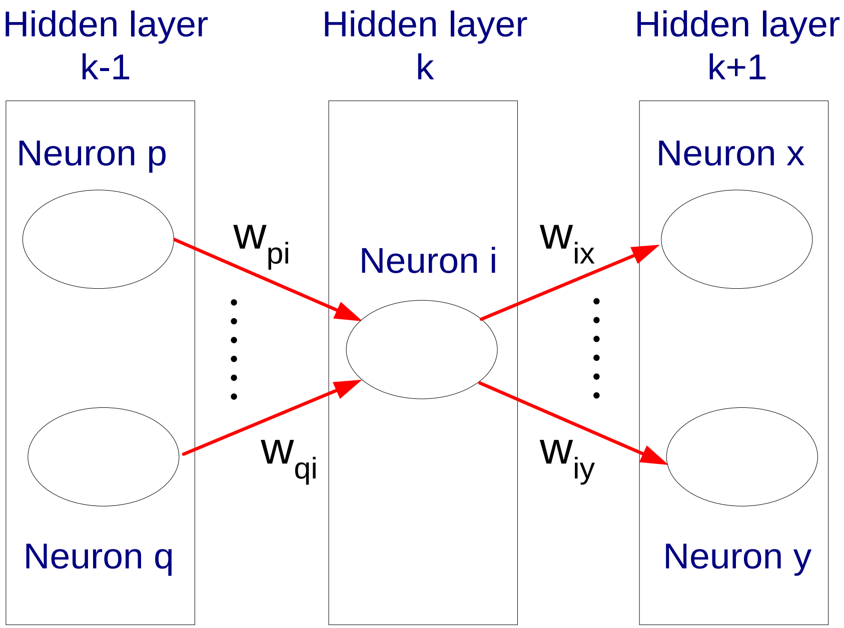
$$w_j^{(k+1)} = w_j^{(k)} + \lambda \sum_i (t_i - o_i) o_i (1 - o_i) x_{ij}$$

- Stochastic gradient descent (update the weight immediately)

Gradient Descent for MultiLayer NN

- For output neurons, weight update formula is the same as before (gradient descent for perceptron)

- For hidden neurons:



$$w_{pi}^{(k+1)} = w_{pi}^{(k)} + \lambda o_i(1-o_i)\sum_{j\in\Phi_i}\delta_j w_{ij} x_{pi}$$

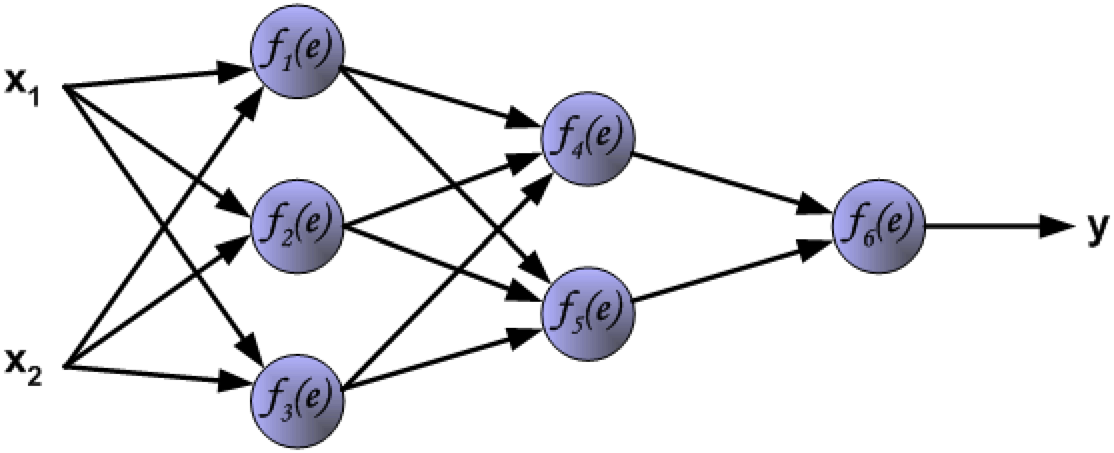
$$\text{Output neurons}: \delta_j = o_j(1-o_j)(t_j - o_j)$$

$$\text{Hidden neurons}: \delta_j = o_j(1-o_j)\sum_{k\in\Phi_j}\delta_k w_{jk}$$

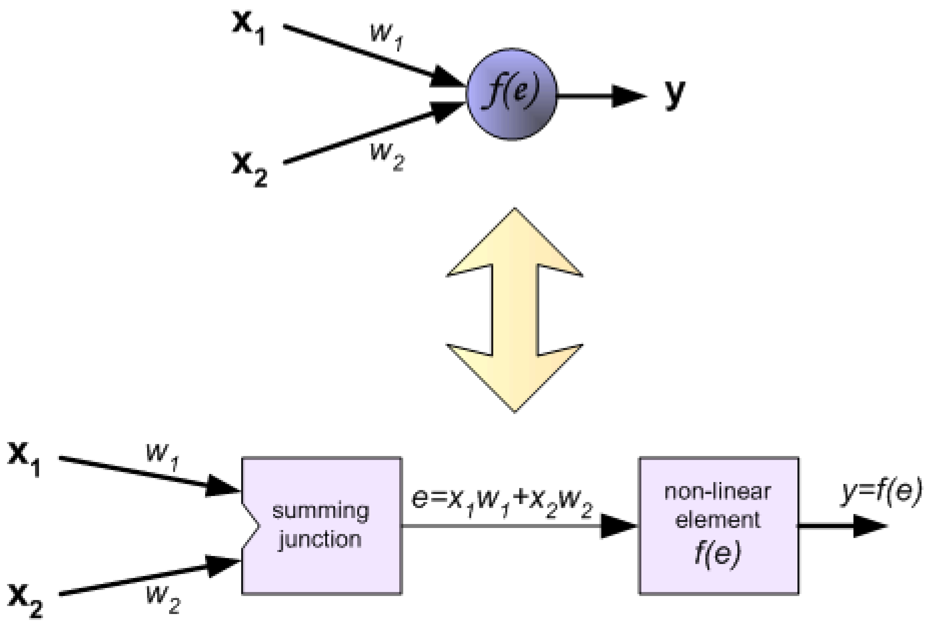
Backpropagation

- Iteratively process a set of training tuples & compare the network's prediction with the actual known target value
- For each training tuple, the weights are modified to **minimize the mean squared error** between the network's prediction and the actual target value
- Modifications are made in the "**backwards**" direction: from the output layer, through each hidden layer down to the first hidden layer, hence "**backpropagation**"
- Steps
  - Initialize weights (to small random #s) and biases in the network
  - Propagate the inputs forward (by applying activation function)
  - Backpropagate the error (by updating weights and biases)
  - Terminating condition (when error is very small, etc.)

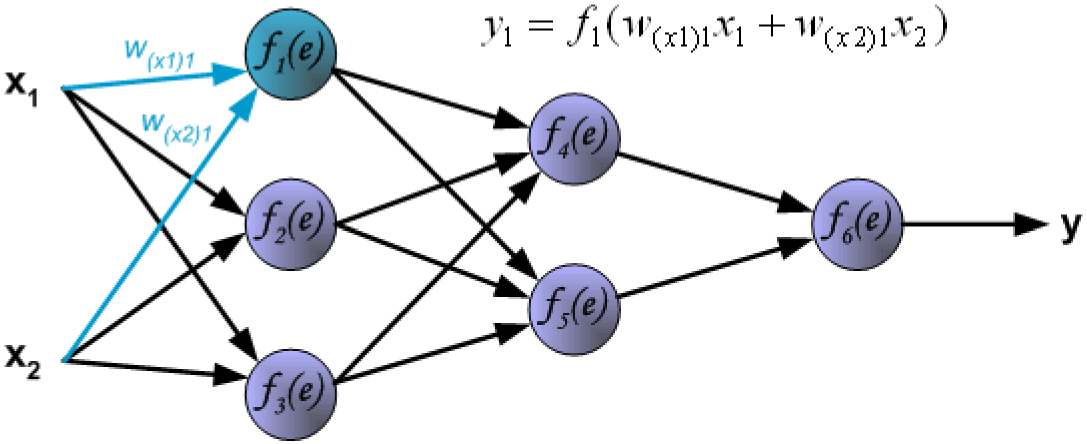
Steps of Backpropagation



Steps of Backpropagation

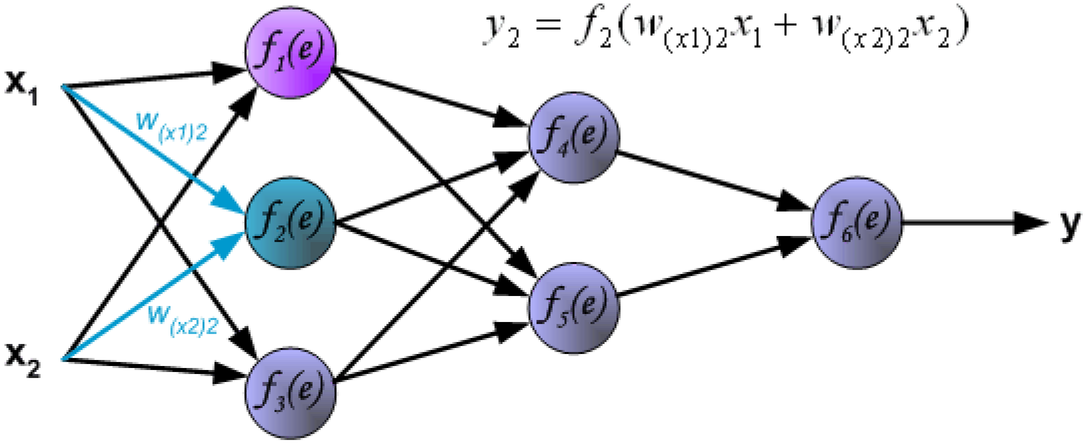


...

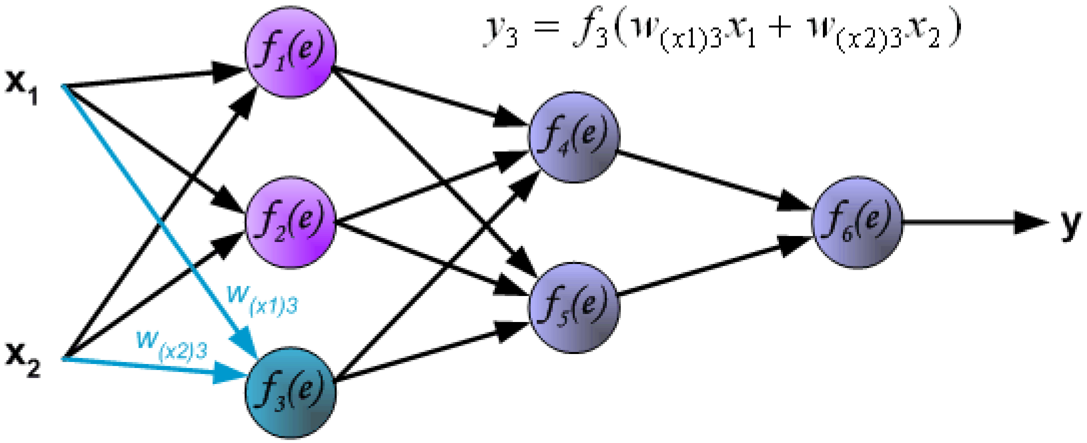


$$y_1 = f_1(w_{(x1)1}x_1 + w_{(x2)1}x_2)$$

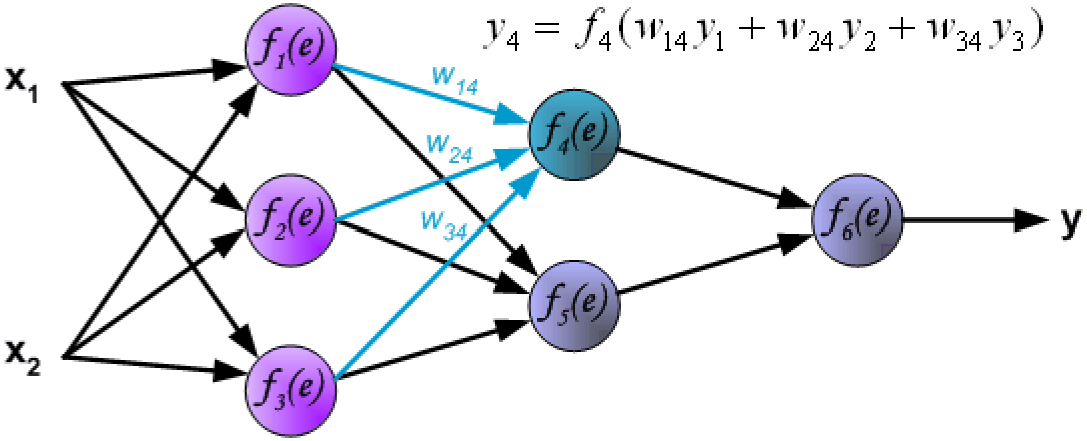
...


$y_2 = f_2(w_{(x1)2}x_1 + w_{(x2)2}x_2)$
x_1
x_2
$w_{(x1)2}$
$w_{(x2)2}$
$f_1(e)$
$f_2(e)$
$f_3(e)$
$f_4(e)$
$f_5(e)$
$f_6(e)$
y

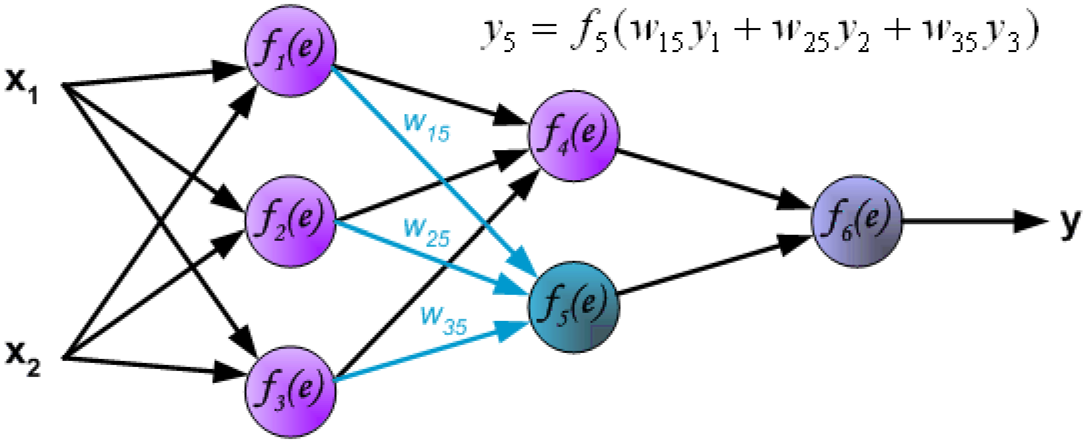
...


$y_3 = f_3(w_{(x1)3}x_1 + w_{(x2)3}x_2)$
x_1
x_2
$f_1(e)$
$f_2(e)$
$f_3(e)$
$f_4(e)$
$f_5(e)$
$f_6(e)$
$w_{(x1)3}$
$w_{(x2)3}$
y

…

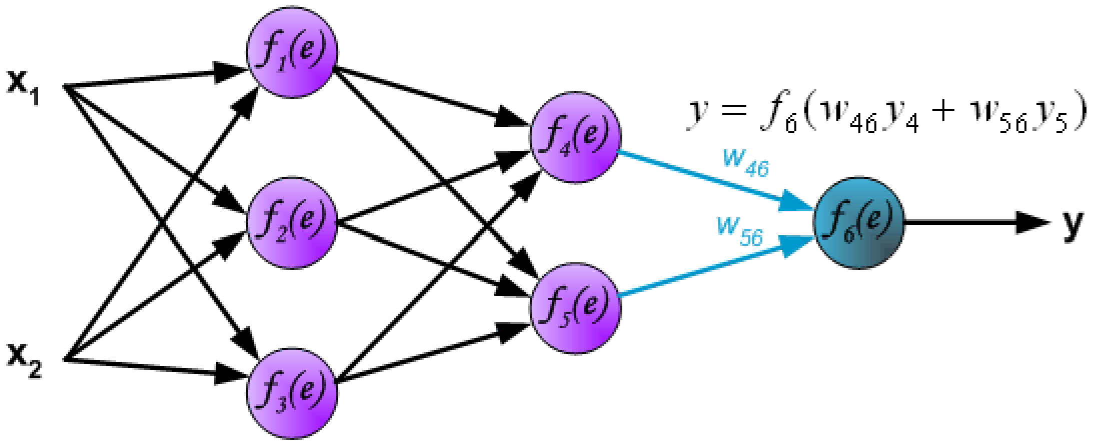

$y_4 = f_4(w_{14}y_1 + w_{24}y_2 + w_{34}y_3)$
x_1
x_2
$f_1(e)$
$f_2(e)$
$f_3(e)$
w_{14}
w_{24}
w_{34}
$f_4(e)$
$f_5(e)$
$f_6(e)$
y

...

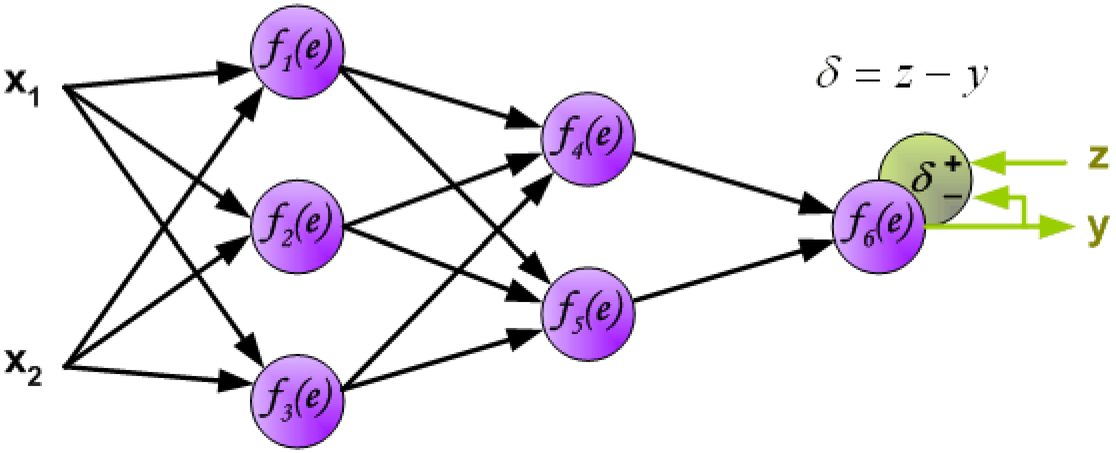


$$y_5 = f_5(w_{15}y_1 + w_{25}y_2 + w_{35}y_3)$$

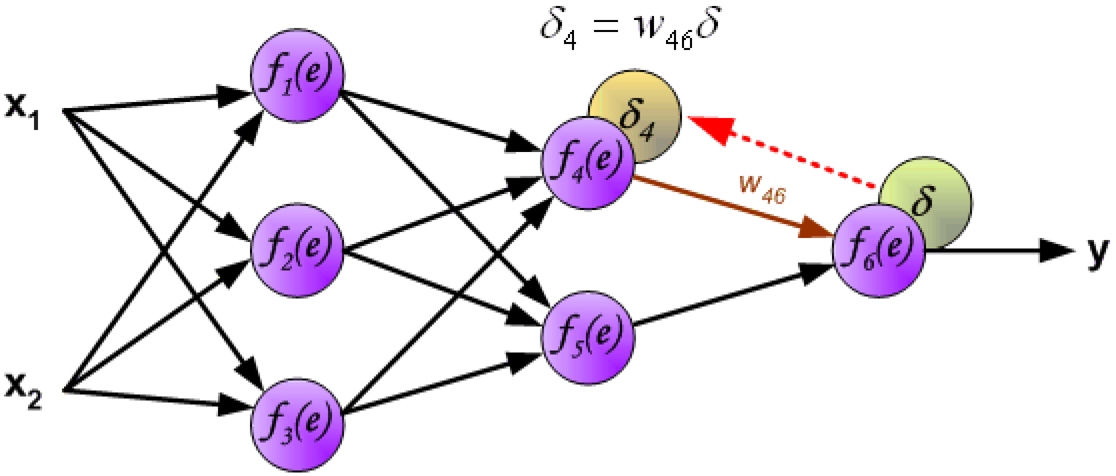
▪▪▪


x_1
x_2
$f_1(e)$
$f_2(e)$
$f_3(e)$
$f_4(e)$
$f_5(e)$
$y = f_6(w_{46}y_4 + w_{56}y_5)$
w_{46}
w_{56}
$f_6(e)$
y

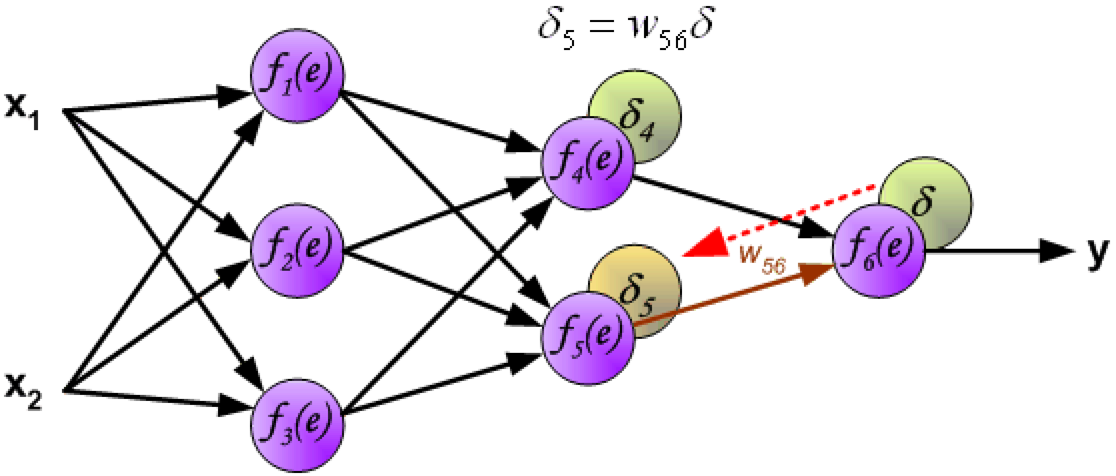
…


$\delta = z - y$
x_1
x_2
$f_1(e)$
$f_2(e)$
$f_3(e)$
$f_4(e)$
$f_5(e)$
$f_6(e)$
δ
+
–
z
y

…

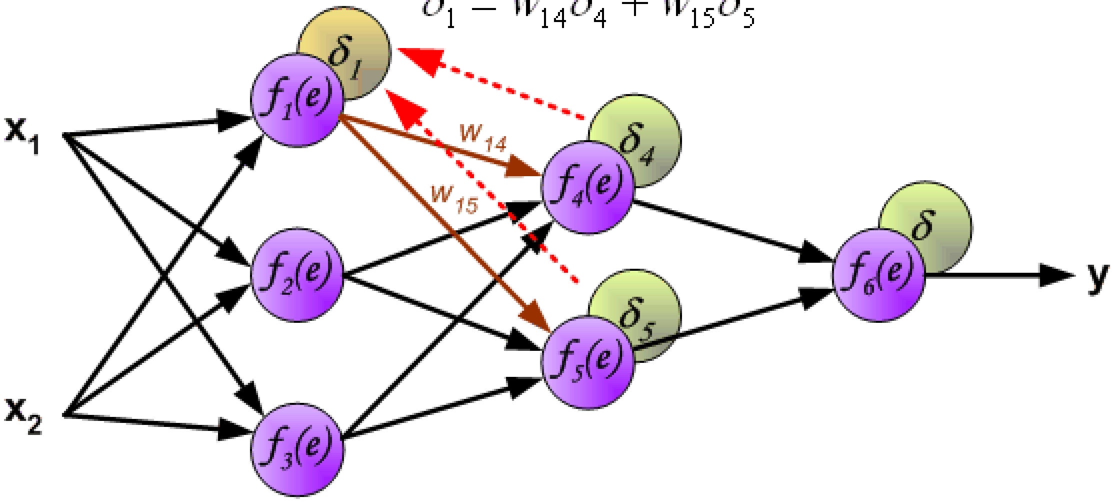

$\delta_4 = w_{46}\delta$
x_1
x_2
$f_1(e)$
$f_2(e)$
$f_3(e)$
$f_4(e)$
$f_5(e)$
$f_6(e)$
δ_4
w_{46}
δ
y

...


$\delta_5 = w_{56}\delta$
x_1
x_2
$f_1(e)$
$f_2(e)$
$f_3(e)$
δ_4
$f_4(e)$
δ_5
$f_5(e)$
w_{56}
δ
$f_6(e)$
y

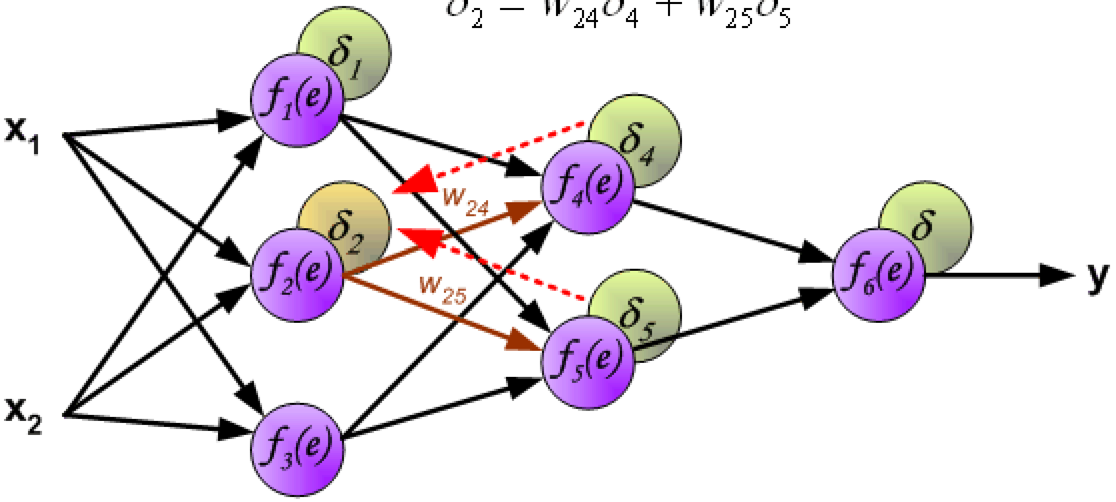
…

$$\delta_1 = w_{14}\delta_4 + w_{15}\delta_5$$



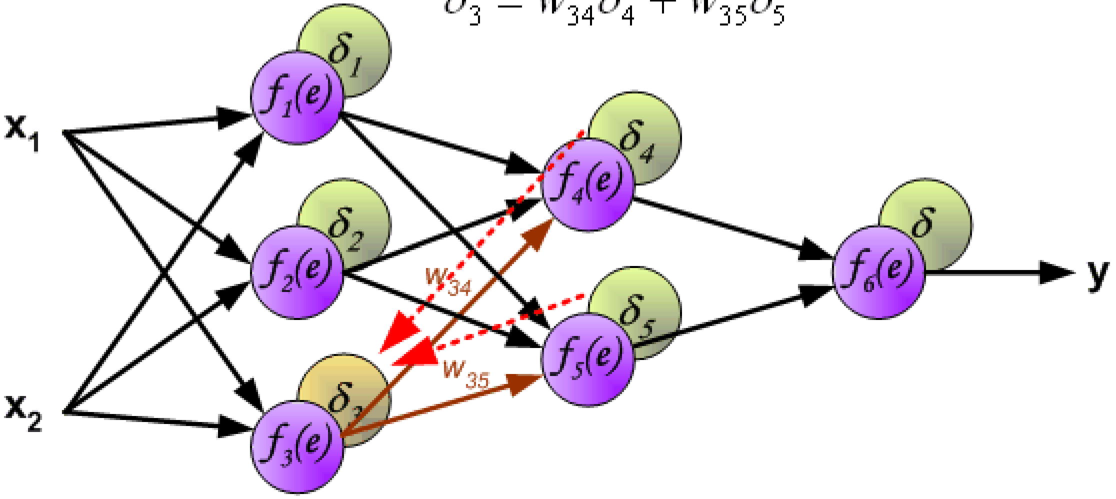
...

$$\delta_2 = w_{24}\delta_4 + w_{25}\delta_5$$

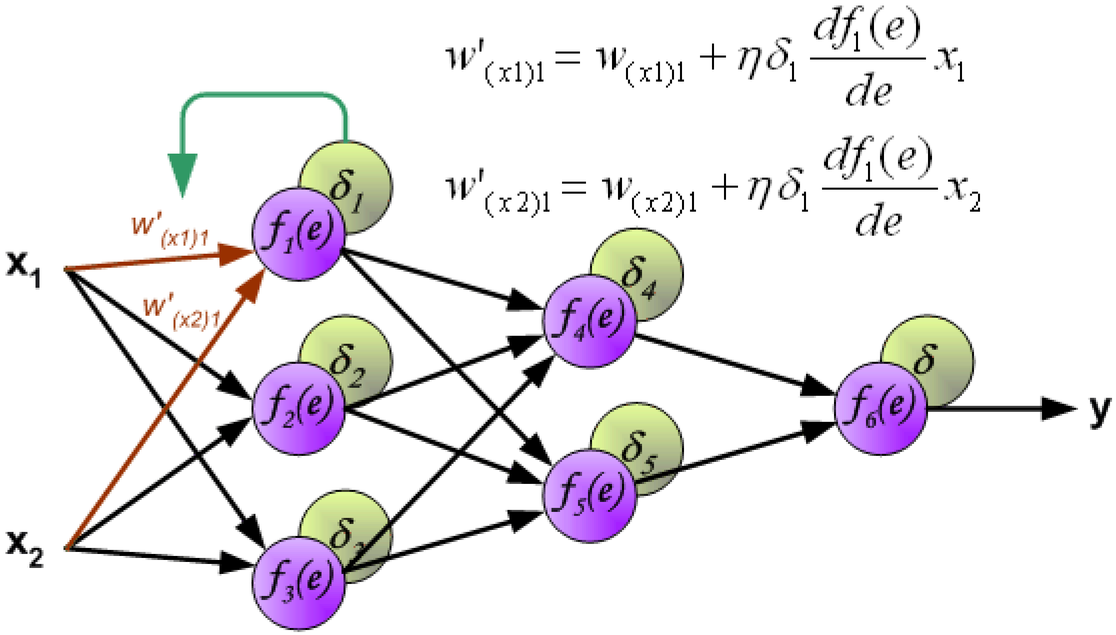


…

$$\delta_3 = w_{34}\delta_4 + w_{35}\delta_5$$



...



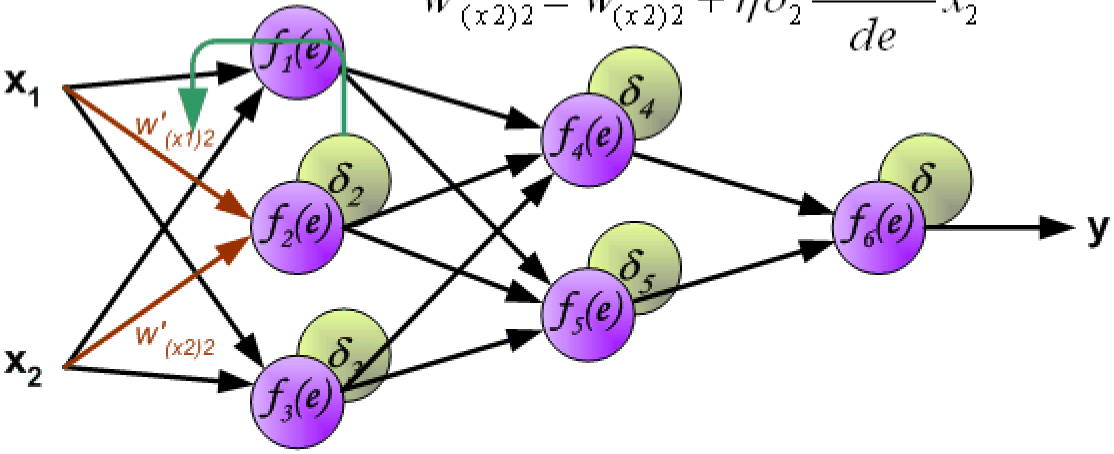
$$w'_{(x1)1} = w_{(x1)1} + \eta \delta_1 \frac{df_1(e)}{de} x_1$$

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

...

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

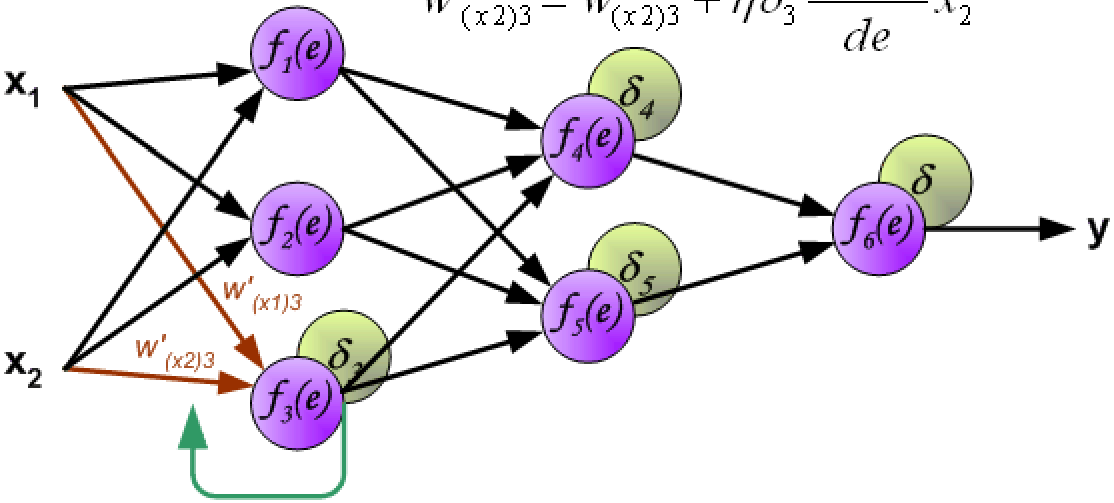
$$w'_{(x2)2} = w_{(x2)2} + \eta \delta_2 \frac{df_2(e)}{de} x_2$$



...

$$w'_{(x1)3} = w_{(x1)3} + \eta \delta_3 \frac{df_3(e)}{de} x_1$$

$$w'_{(x2)3} = w_{(x2)3} + \eta \delta_3 \frac{df_3(e)}{de} x_2$$

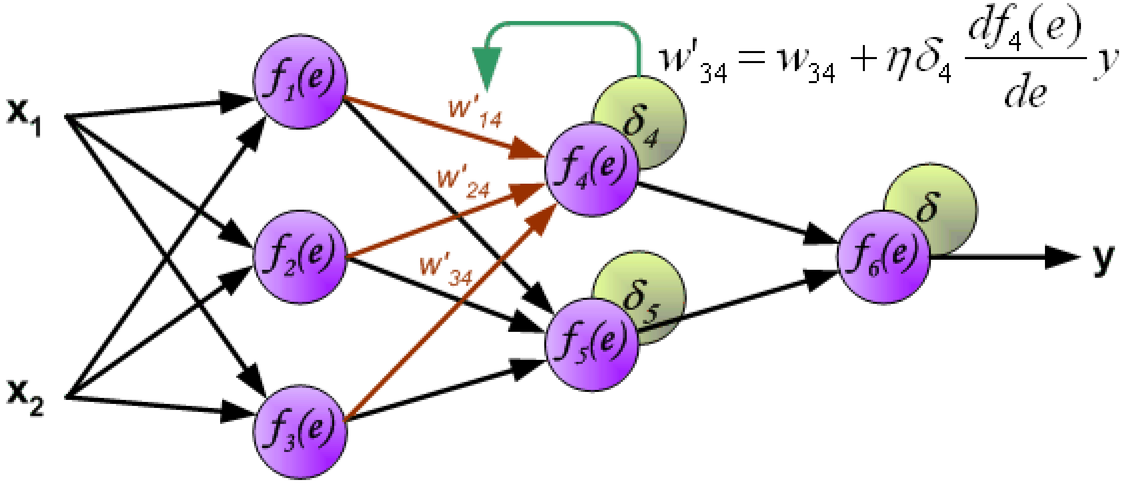


...

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

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

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

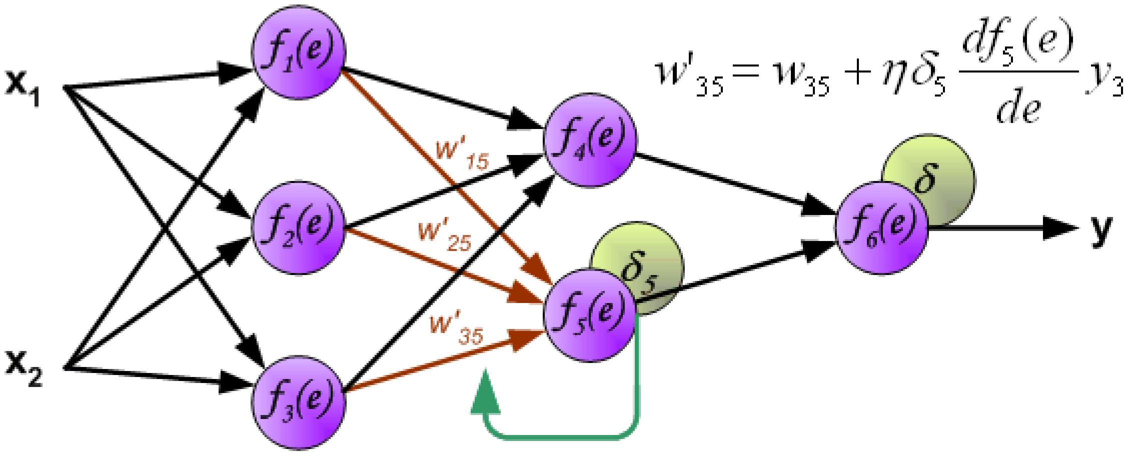


...

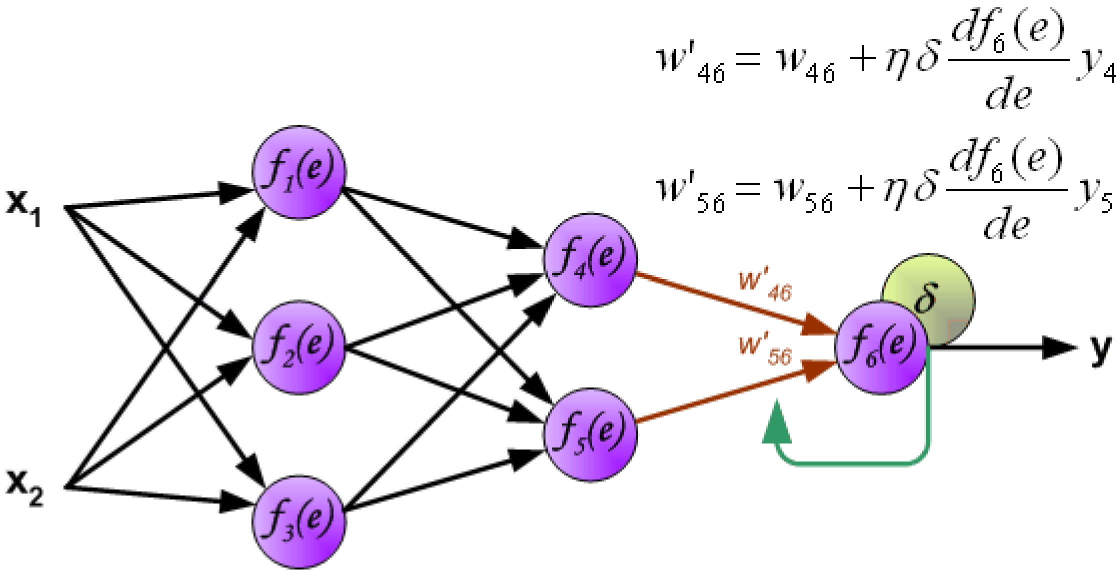
$$w'_{15} = w_{15} + \eta \delta_5 \frac{df_5(e)}{de} y_1$$

$$w'_{25} = w_{25} + \eta \delta_5 \frac{df_5(e)}{de} y_2$$

$$w'_{35} = w_{35} + \eta \delta_5 \frac{df_5(e)}{de} y_3$$



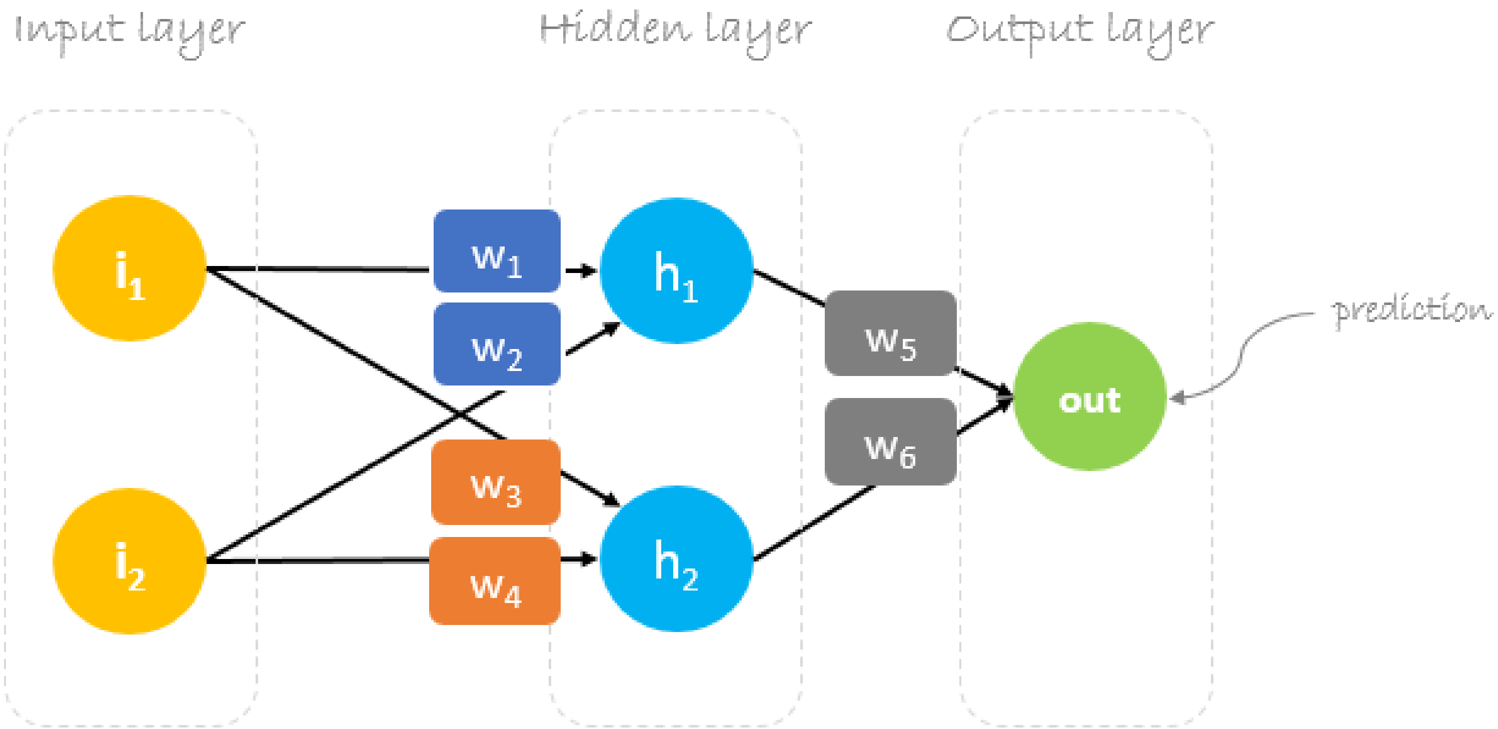
…



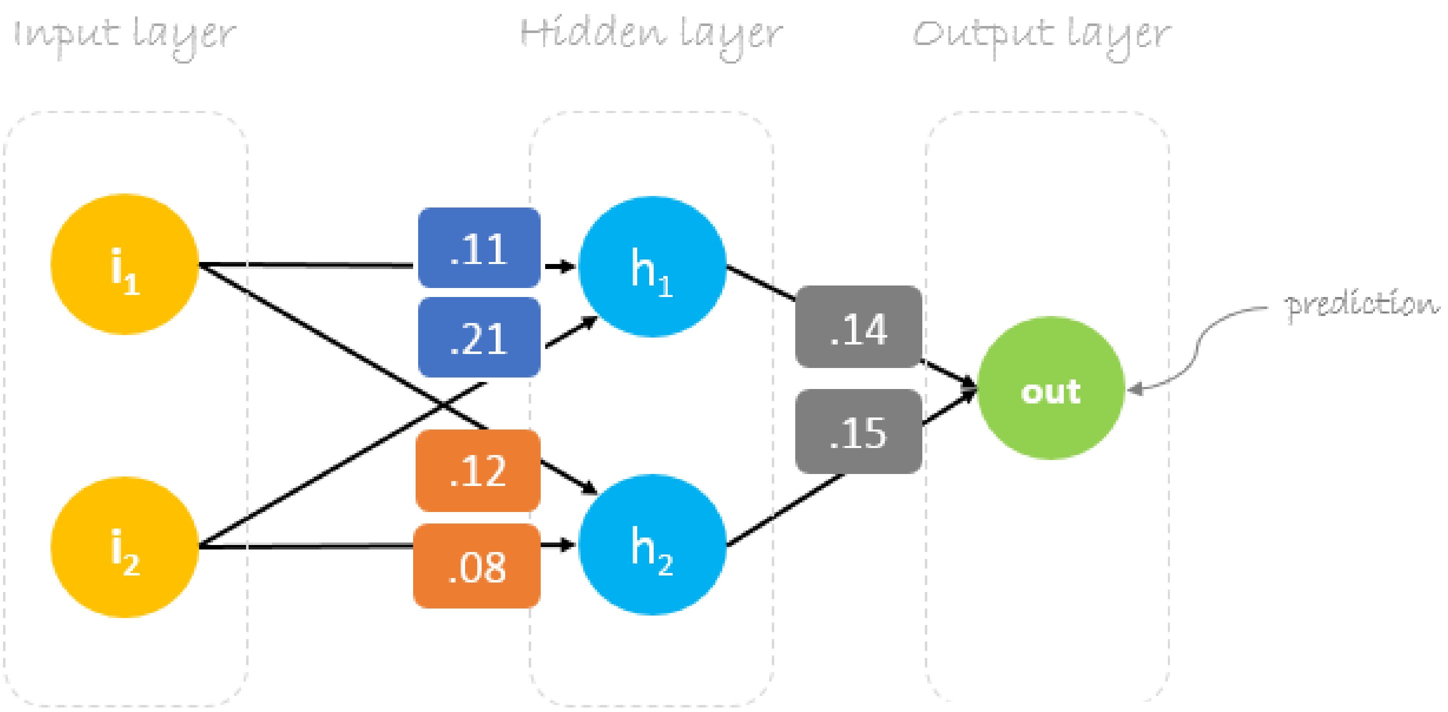
$$w'_{46} = w_{46} + \eta\,\delta\,\frac{df_6(e)}{de}\,y_4$$

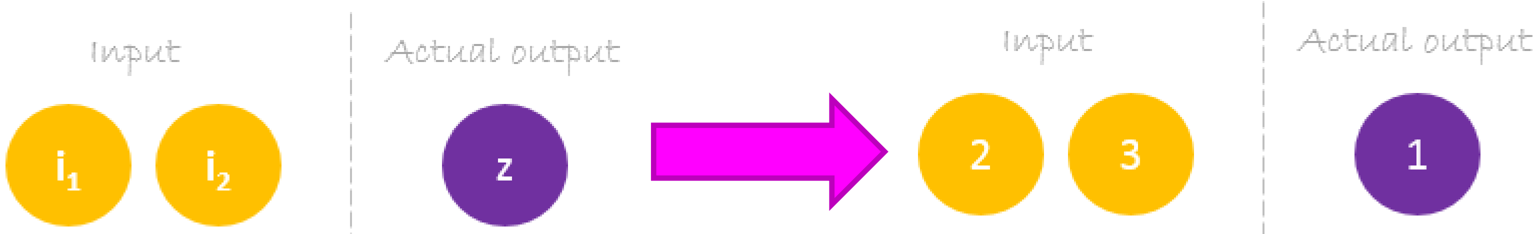
$$w'_{56} = w_{56} + \eta\,\delta\,\frac{df_6(e)}{de}\,y_5$$

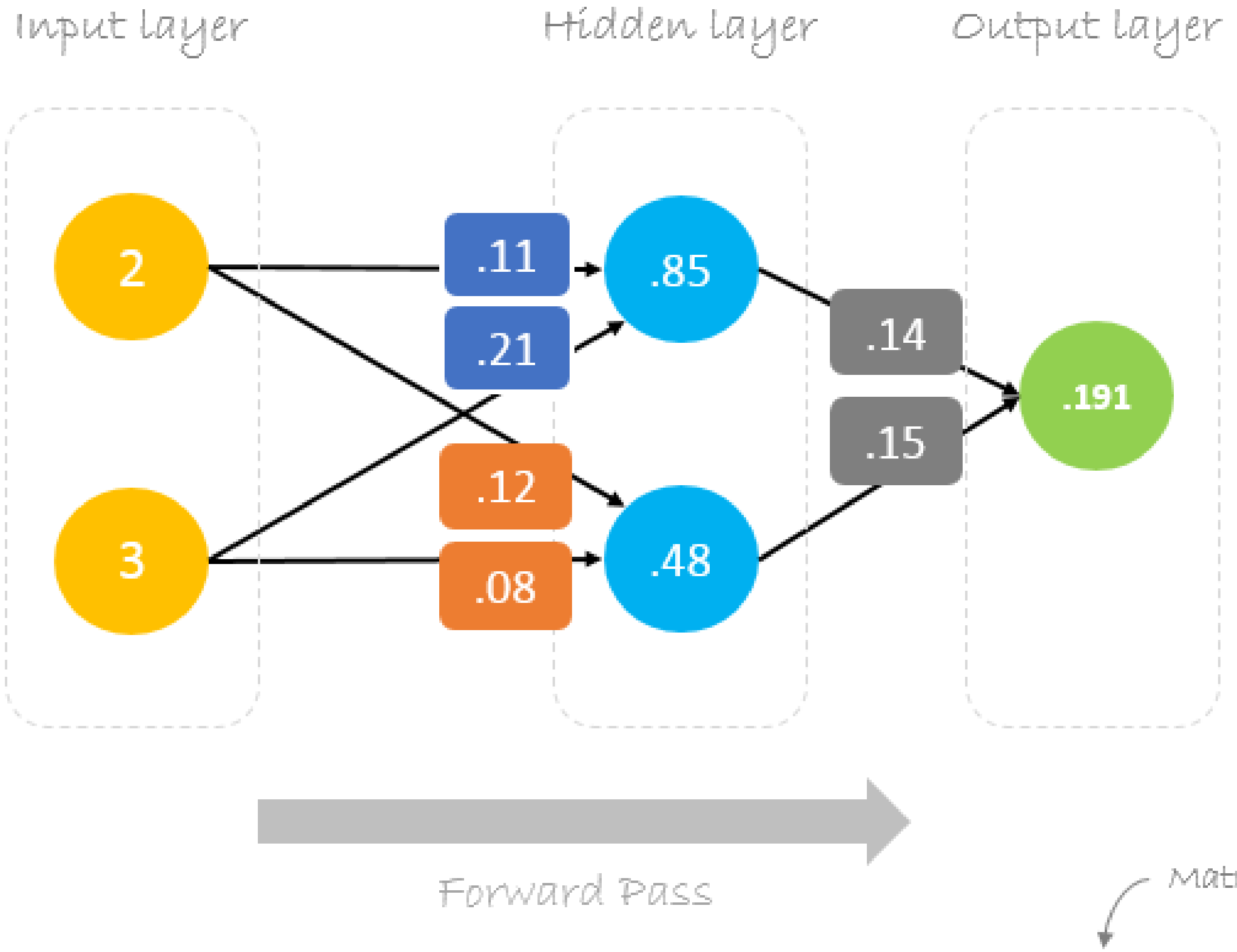
MLP Example


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

Initial Weights







Matrix multiplication

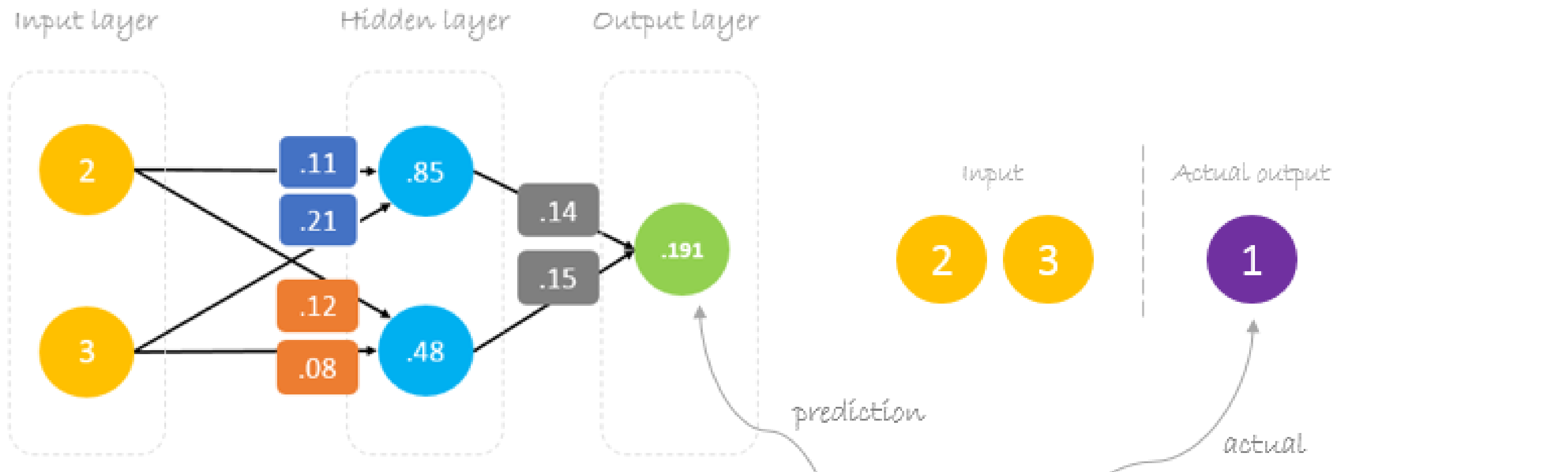
$$[2\quad 3]\,.\begin{bmatrix}0.11 & 0.12\\ 0.21 & 0.08\end{bmatrix} = [0.85\quad 0.48]\,.\begin{bmatrix}0.14\\ 0.15\end{bmatrix} = [0.191]$$

Details

$2 \times .11 + 3 \times .21 = .85$

$.85 \times .14 + .48 \times .15 = .191$

$2 \times .12 + 3 \times .08 = .48$



prediction

actual

Error = 0 , if prediction = actual

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

Error is always positive because of the square

$\frac{1}{2}$ is added to ease the calculation of the derivative

$$\text{Error} = \frac{1}{2}(0.191 - 1.0)^2 = 0.327$$

Decrease the Error

$$\text{prediction} = \underline{\text{out}}$$

$$\downarrow$$

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

$$h_1 = i_1 w_1 + i_2 w_2$$

$$h_2 = i_1 w_3 + i_2 w_4$$

$$\downarrow$$

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

Backpropagation Steps

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

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

$$\frac{\partial Error}{\partial W_6} = \frac{\partial Error}{\partial prediction} * \frac{\partial prediction}{\partial W_6}$$

chain rule

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

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

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

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

$h_2 = i_1 w_3 + i_2 w_4$

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

Δ = prediction – actual

delta

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

Update the weights

$$^*w_6 = w_6 - a\,(h_2\ .\ \Delta)$$

$$^*w_5 = w_5 - a\,(h_1\ .\ \Delta)$$

updated weights

$$^*w_4 = w_4 - a\,(i_2\ .\ \Delta w_6)$$

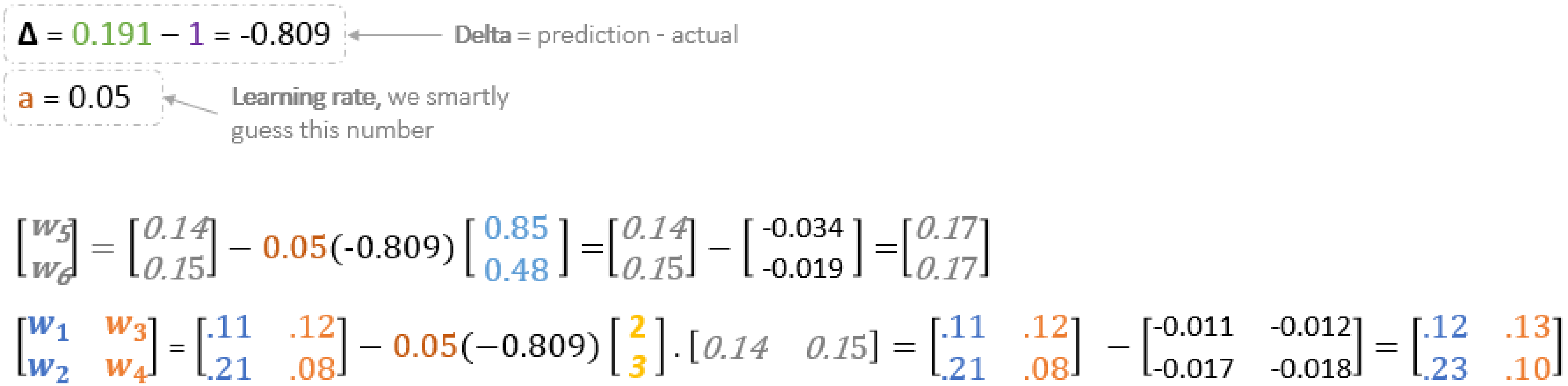
$$^*w_3 = w_3 - a\,(i_1\ .\ \Delta w_6)$$

$$^*w_2 = w_2 - a\,(i_2\ .\ \Delta w_5)$$

$$^*w_1 = w_1 - a\,(i_1\ .\ \Delta w_5)$$

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - a\,\Delta \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - \begin{bmatrix} ah_1\Delta \\ ah_2\Delta \end{bmatrix}$$

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



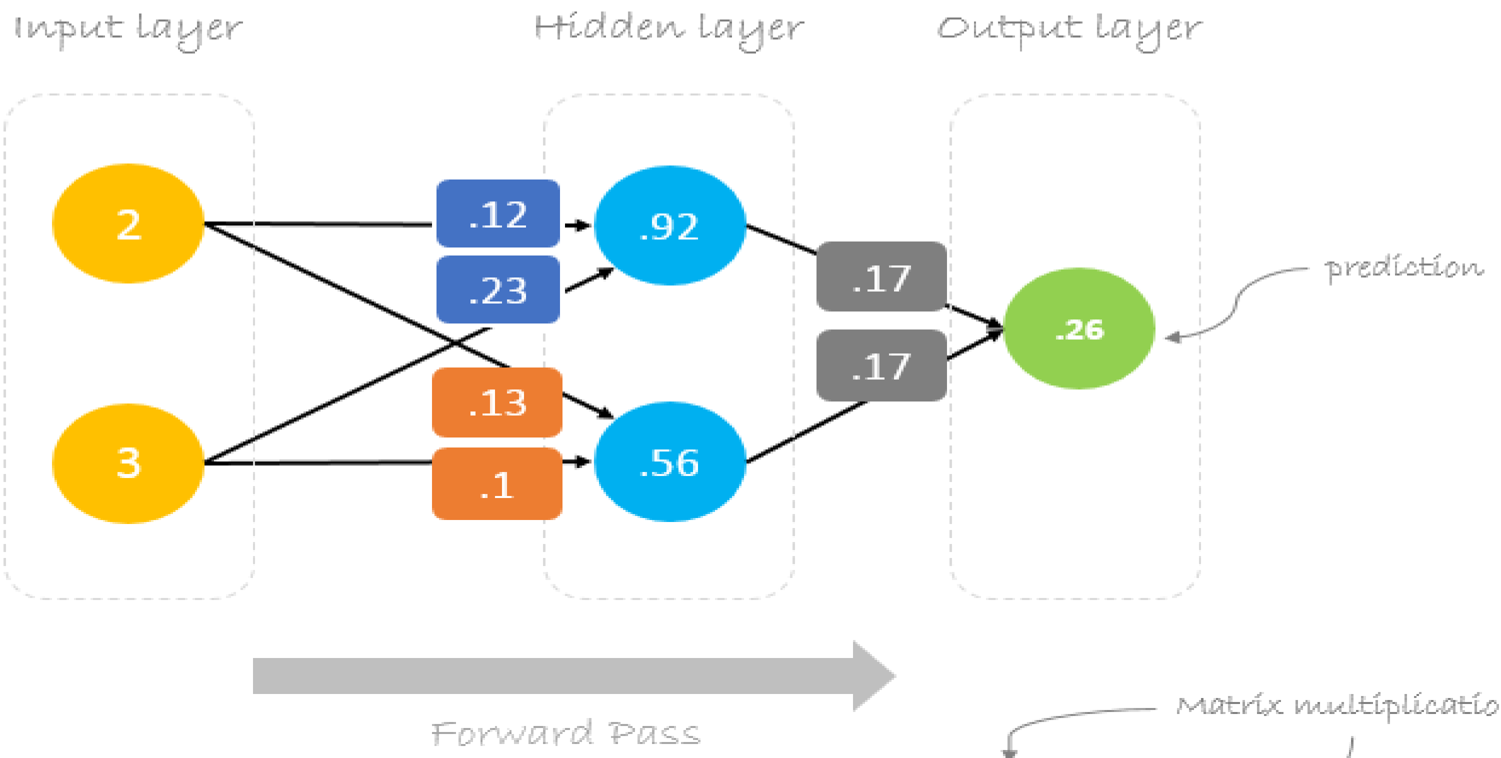
$\Delta = 0.191 - 1 = -0.809$ ← **Delta** = prediction - actual

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

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

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

Weights after the first epoch



Matrix multiplication

$$[2 \quad 3] \cdot \begin{bmatrix} 0.12 & 0.13 \\ 0.23 & 0.10 \end{bmatrix} = [0.92 \quad 0.56] \cdot \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix} = [0.26]$$

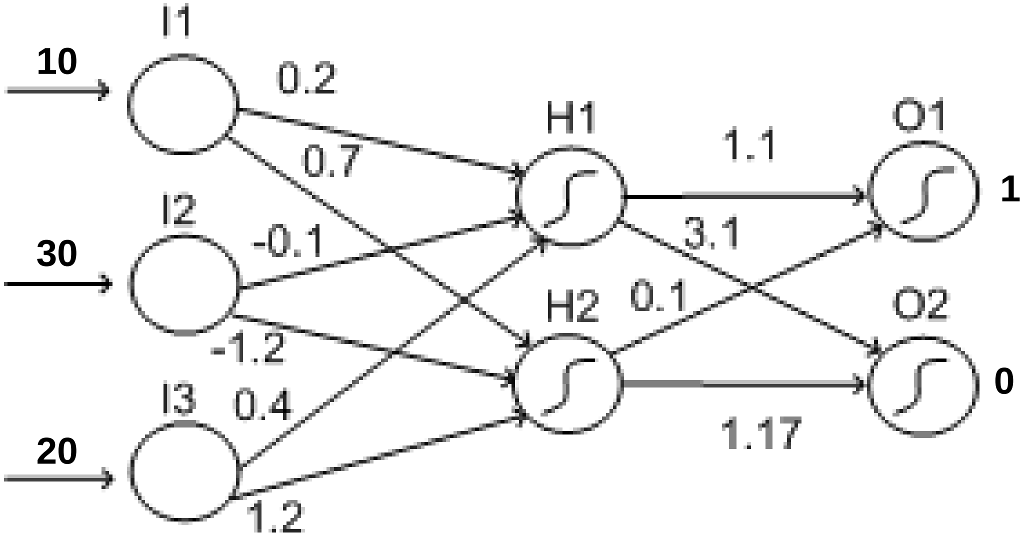
Details

2 x .12 + 3 x .23 = .85

.92 x .17 + .56 x .17 = .26

2 x .13 + 3 x .10 = .48

MLP Exercise



Design Issues in ANN

- Number of nodes in input layer
  - One input node per binary/continuous attribute
  - k or $\log_2$ k nodes for each categorical attribute with k values
- Number of nodes in output layer
  - One output for binary class problem
  - k or $\log_2$ k nodes for k-class problem
- Number of nodes in hidden layer
- Initial weights and biases

Characteristics of ANN

- Multilayer ANN are universal approximators but could suffer from overfitting if the network is too large
- Gradient descent may converge to local minimum
- Model building can be very time consuming, but testing can be very fast
- Can handle redundant attributes because weights are automatically learnt
- Sensitive to noise in training data
- Difficult to handle missing attributes

Recent Noteworthy Developments in ANN

- Use in deep learning and unsupervised feature learning
  - Seek to automatically learn a good representation of the input from unlabeled data
- Google Brain project
  - Learned the concept of a ‘cat’ by looking at unlabeled pictures from YouTube
  - One billion connection network