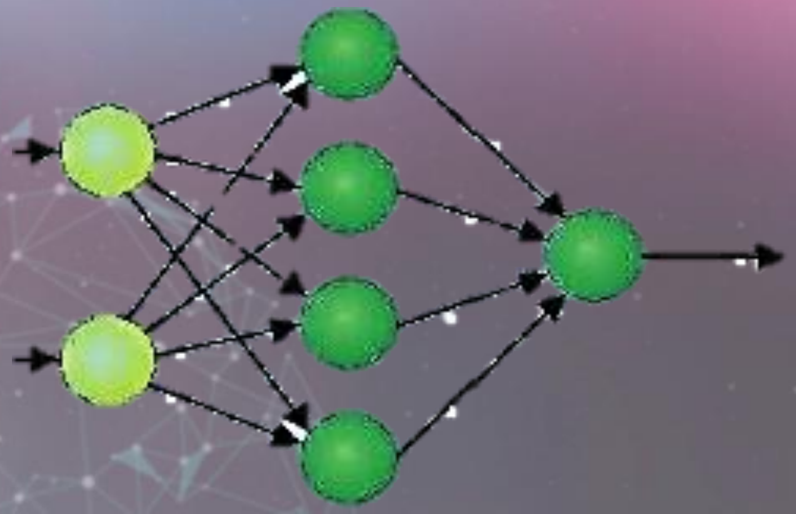
BACK PROPAGATION THEORY IN NEURAL NETWORKS

YAZAN MAALLA

2020

Artificial Neural Networks



- **Artificial intelligence**
- **Biological inspired :**

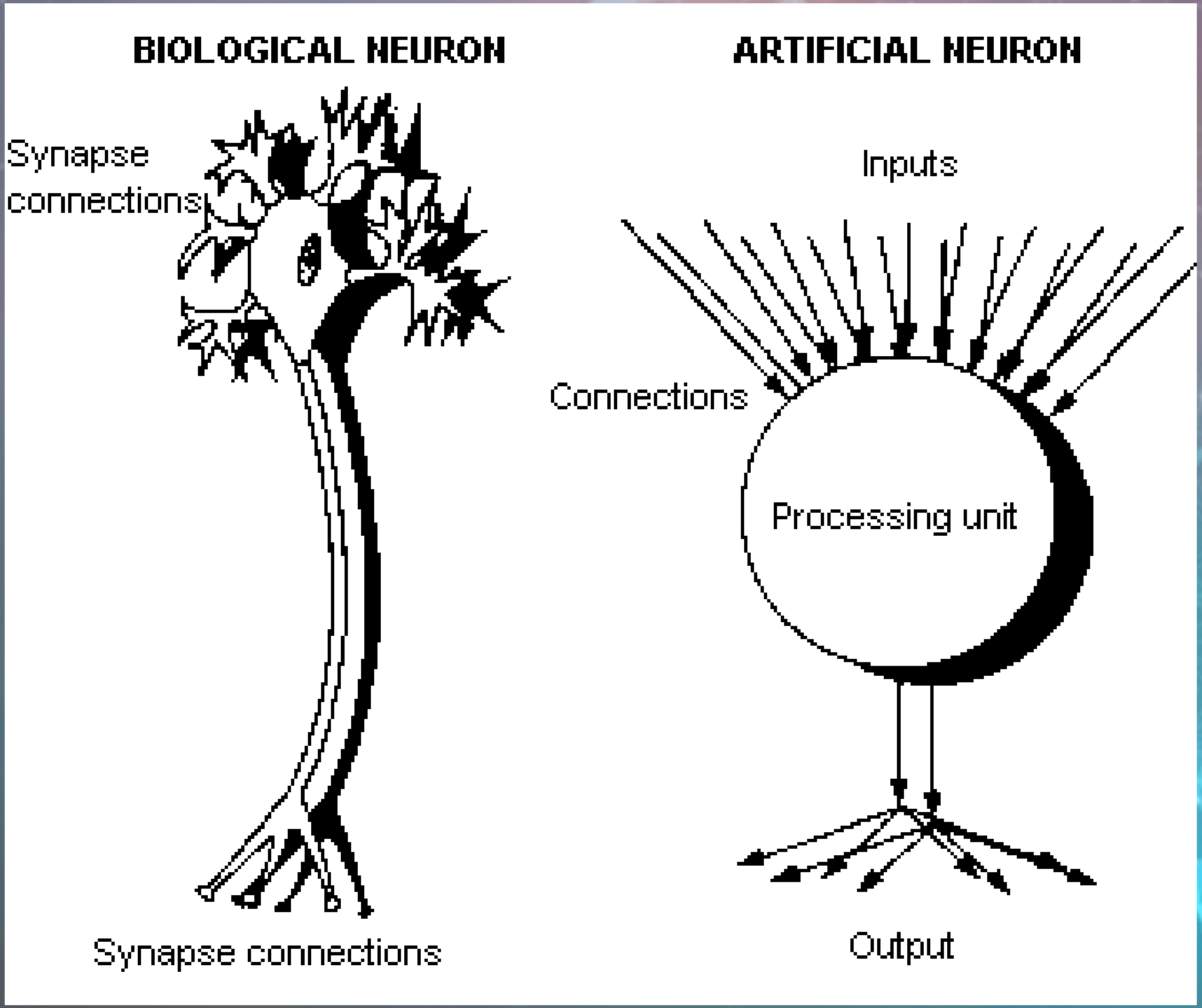
  Computing models tend to mimic the behavior of the biological human brain.

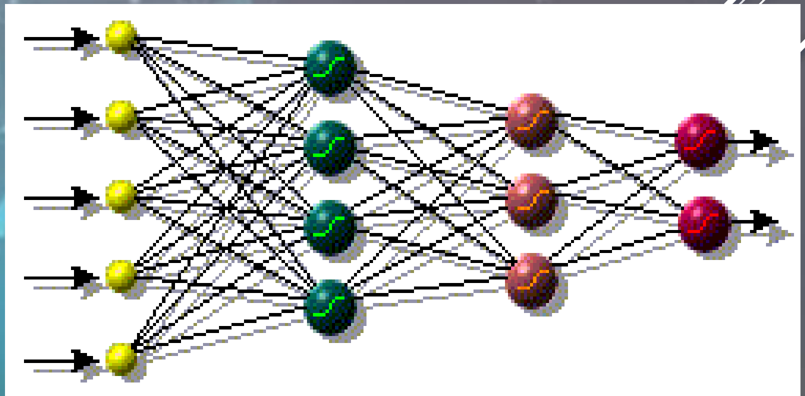
- Consist of many simple connected processing units,

  Neurons.

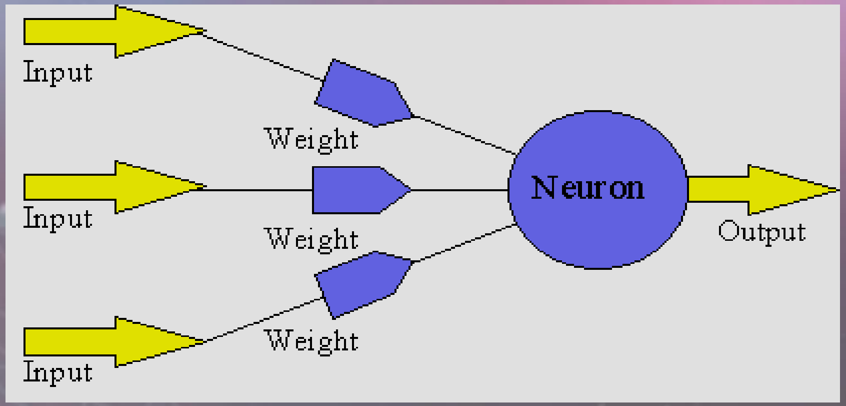
- Each Neuron performs a simple calculating operation.
- The net's total behavior is determined by the connections among all the neurons.

Human Brain V.s ANN





Single neuron model

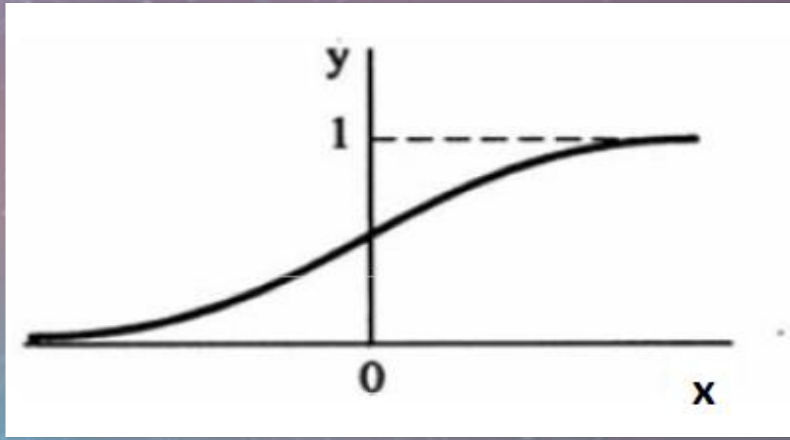


- **Inputs x_i** arrive from other neurons or from outside the network.
- real **weights w_i**
- The adder, $S = \sum_{i=1}^{i=n} w_i * x_i$
- The response of the neuron is a **transform function *f*** of its weighted inputs ***S***

The response function

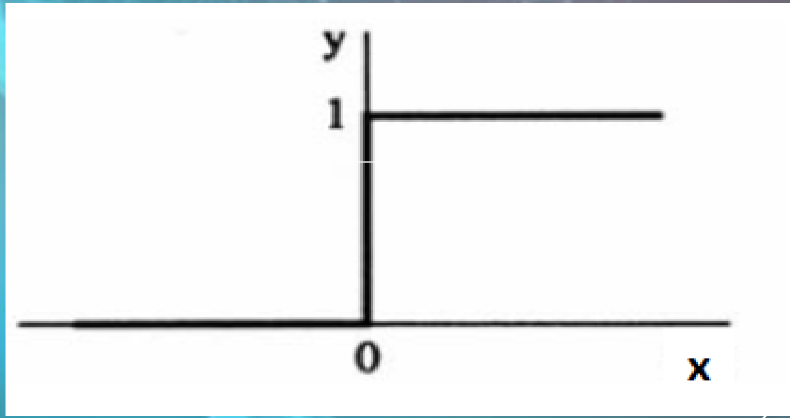
- **Sigmoid**

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



- **Piecewise linear**

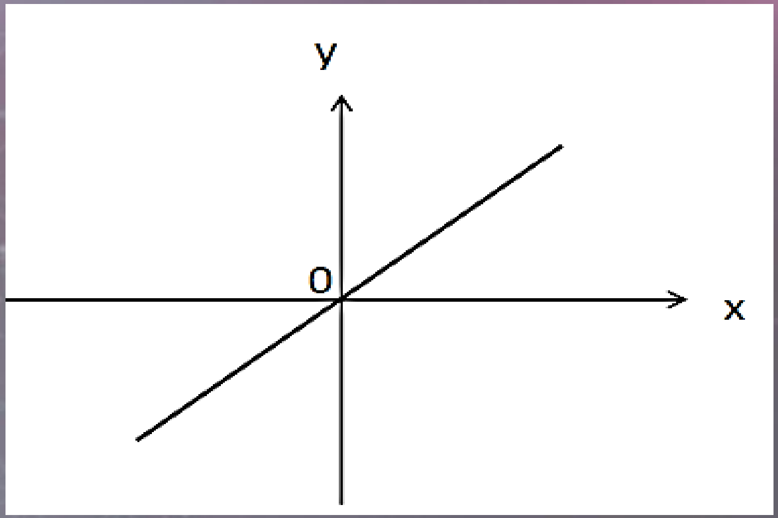
$$f(x) = \begin{cases} x, \text{ if } x \geq \theta \\ 0, \text{ if } x < \theta \end{cases}$$



The response function

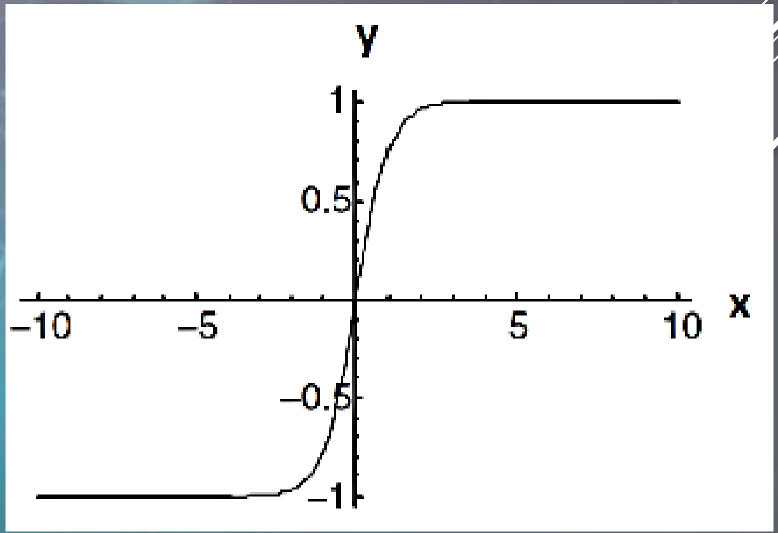
- linear

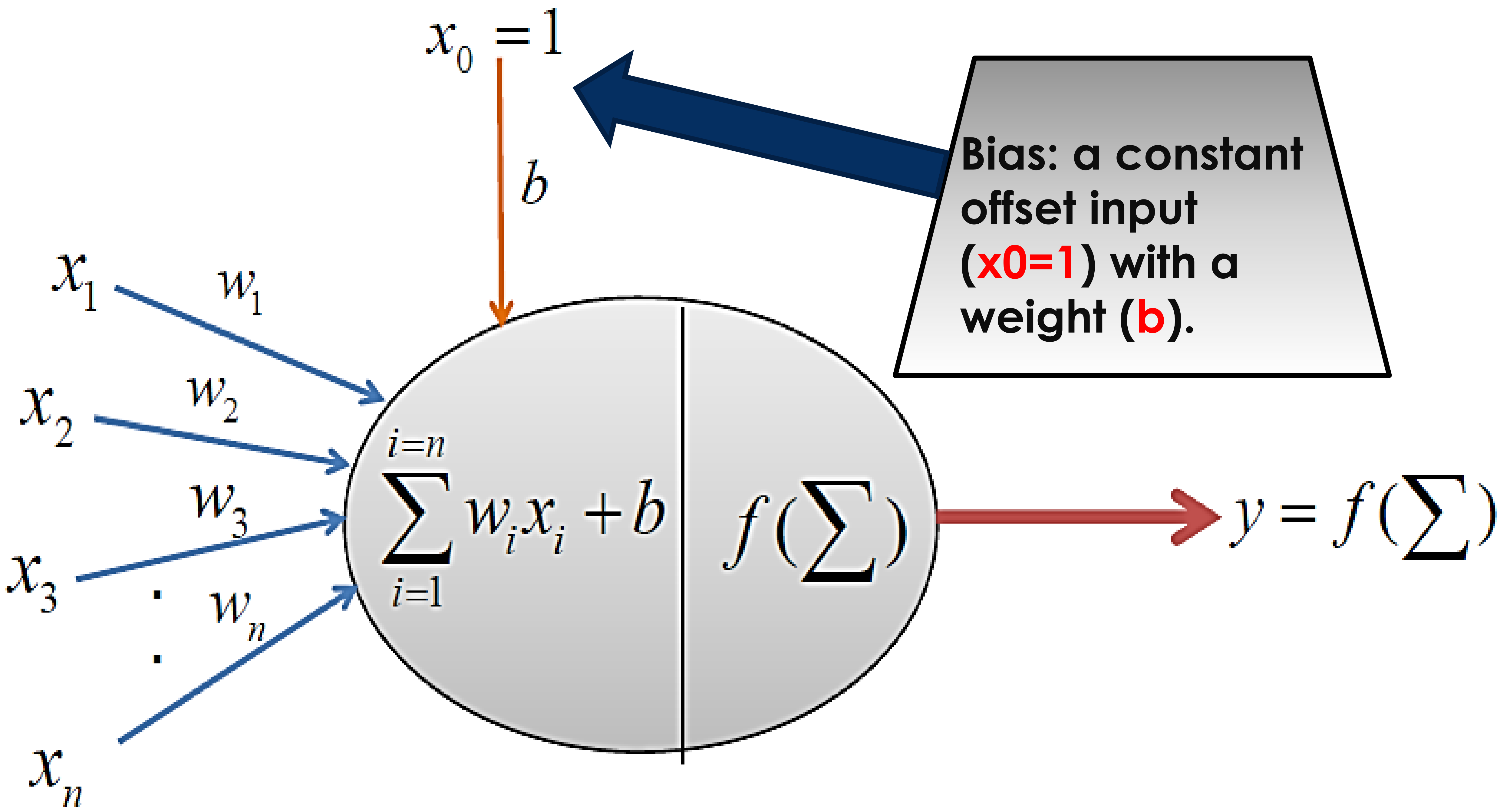
$$f(x) = x$$



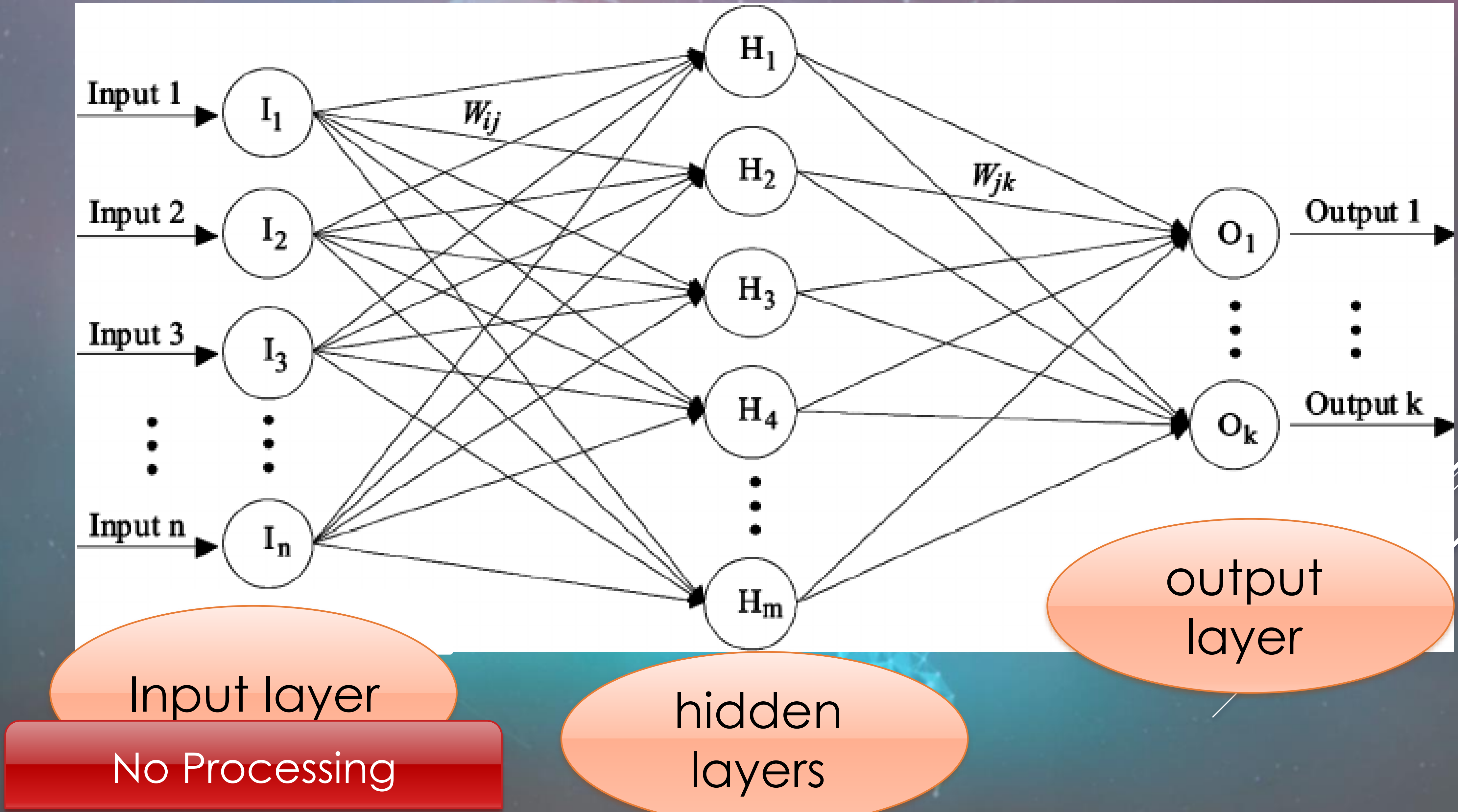
- Hyperbolic tangent function

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




$x_0 = 1$
b
Bias: a constant offset input (x0=1) with a weight (b).
x_1
w_1
x_2
w_2
x_3
w_3
w_n
x_n
$\sum_{i=1}^{i=n} w_i x_i + b$
$f(\sum)$
$y = f(\sum)$

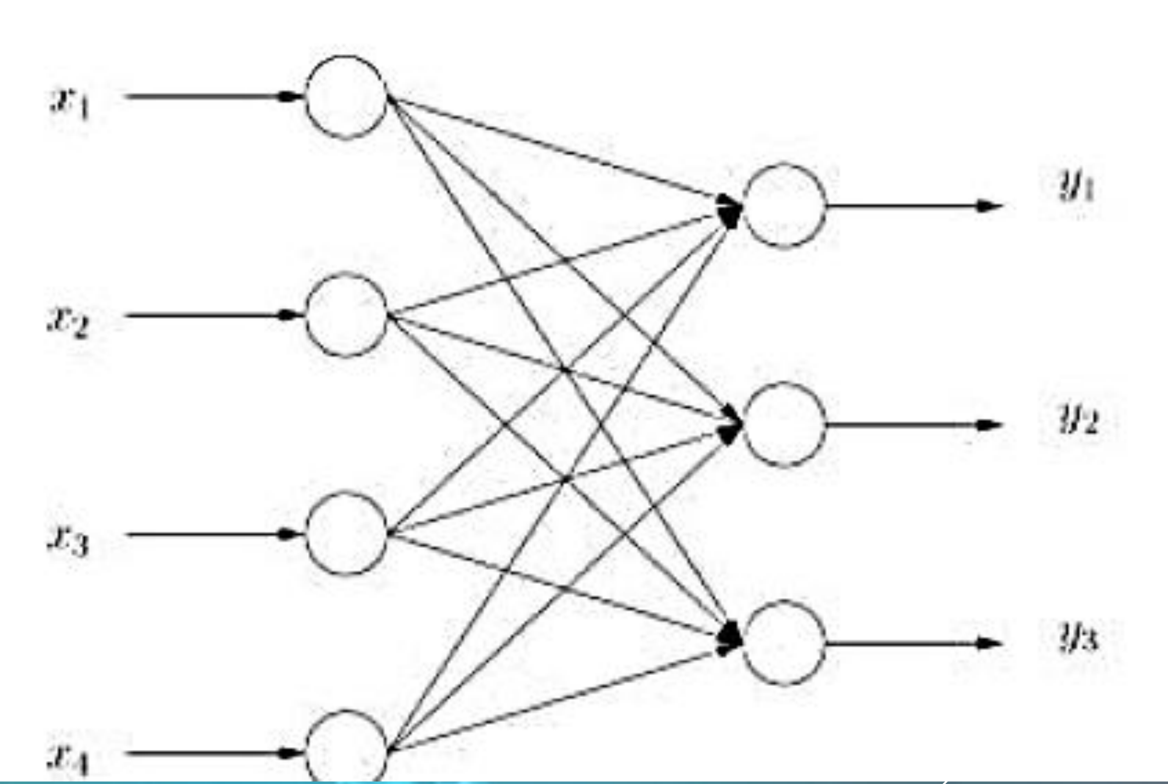
Neural Network Topology



Perceptron :

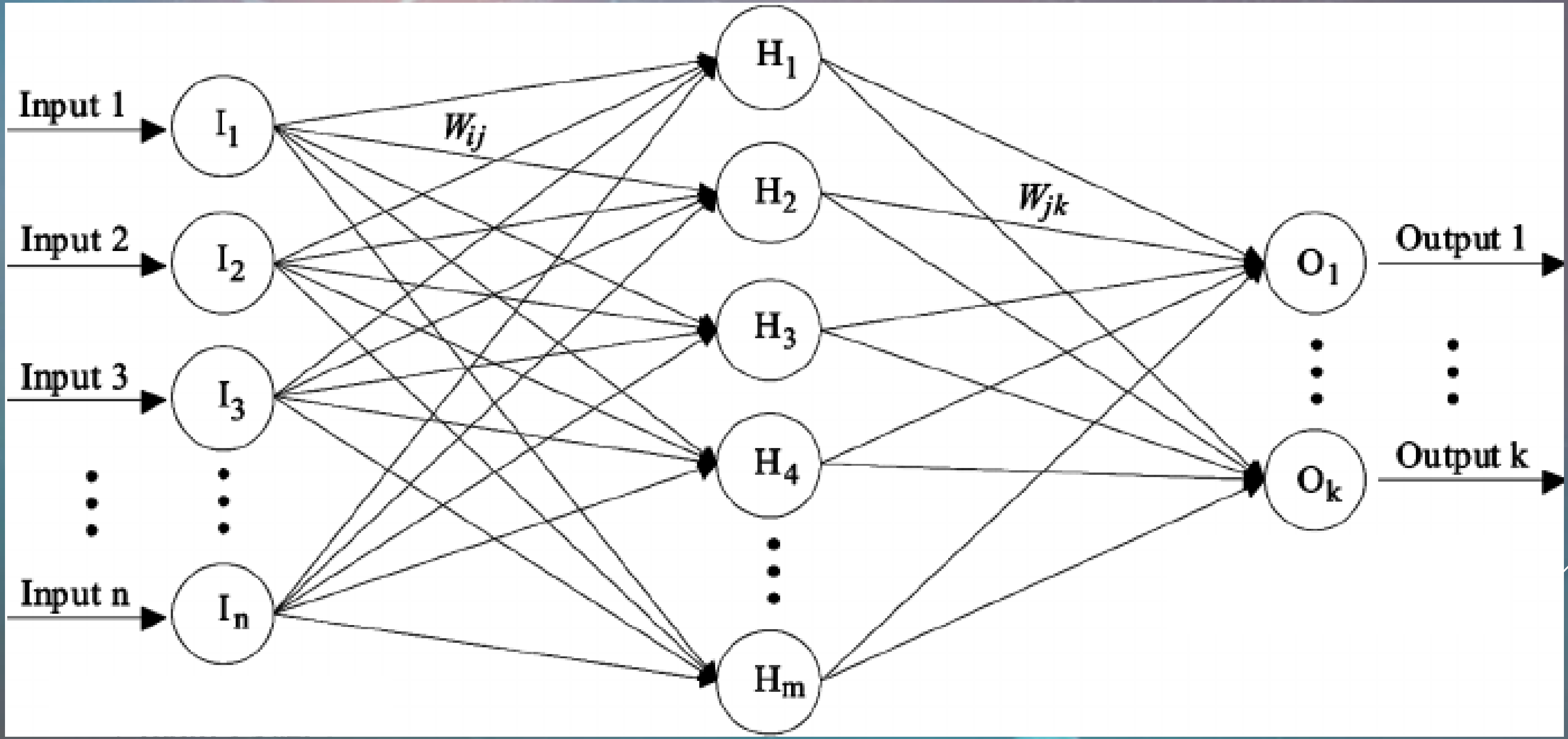
The simplest

- Input layer
- Output layer
- No Hidden layers



MLP : Multi Layer Perceptron

- The most common
- Input layer/ one or more hidden layers/ Output layer

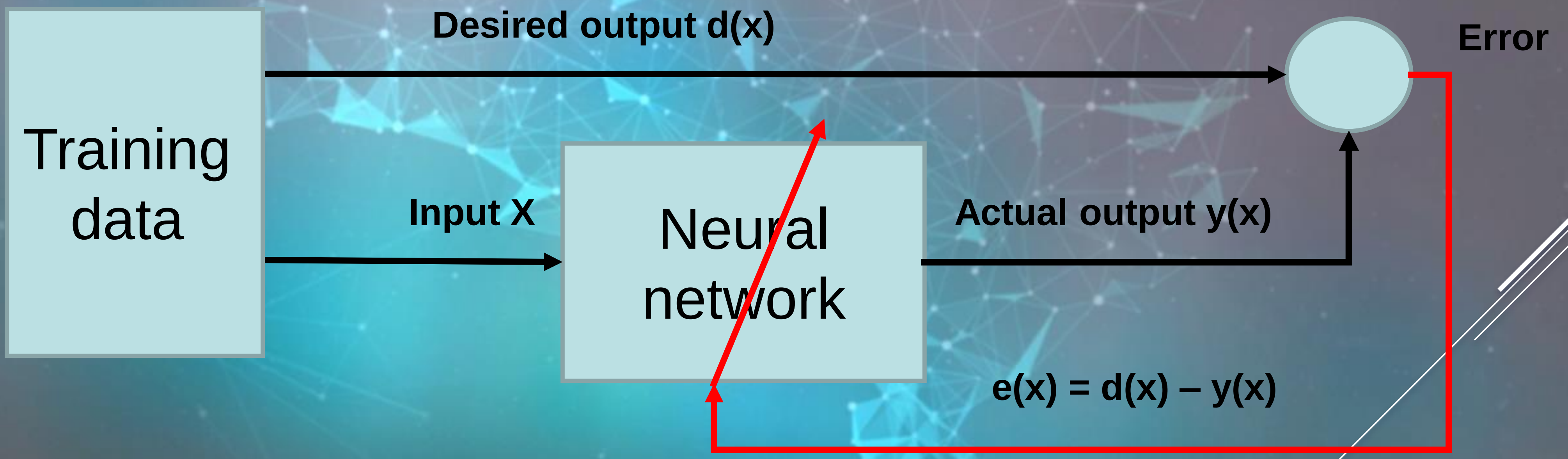


- **Learning procedure:** adjusting connection weights, until network gets desired behavior.
- Learning depends on a set of data (Training data)

- ***Supervised Learning***
- ***Unsupervised Learning***

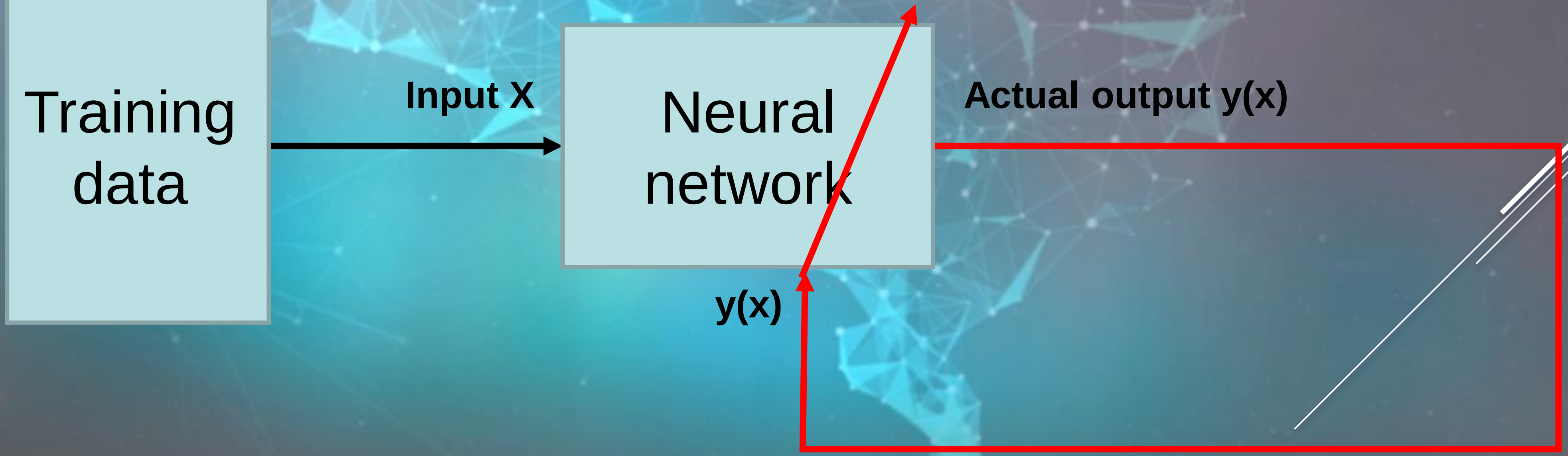
SUPERVISED LEARING

- Training data is a set of (input-desired output) patterns
- Basic principle: iterative error minimization



UNSUPERVISED LEARING

- Training data is only the input value vector.
- The network build up a learning method depending on the features of the input items.



BACKPROPAGATION ALGORITHM

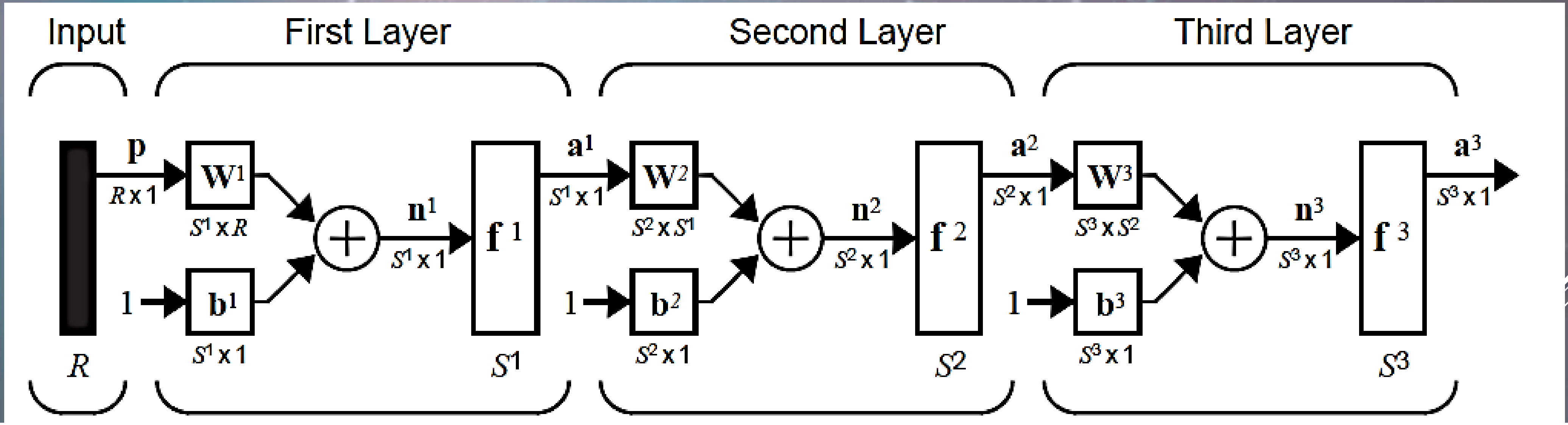
- **Supervised learning**
- **Training Set**
  A collection of input-output patterns that are used to train the network.
- **Testing Set**
  A collection of input-output patterns that are used to assess network performance.
- **Learning Rate-η**
  A scalar parameter, analogous to step size in numerical integration, used to set the rate of adjustments

BACKPROPAGATION ALGORITHM

1. Random weights initialization.
2. Applying data from training set.
3. Compute the actual output using feeding forward
4. Compare actual output to the desired output. /Error computation/
5. Updating weights.
6. Repeating 3 - 5

BACKPROPAGATION ALGORITHM

- Depends on Chain Rule of Calculus, for computing the derivatives of the error function with respect to the weights in MLPs.



$$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1) \qquad \mathbf{a}^2 = \mathbf{f}^2(\mathbf{W}^2\mathbf{a}^1+\mathbf{b}^2) \qquad \mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{a}^2+\mathbf{b}^3)$$

$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3)$$

BACKPROPAGATION ALGORITHM

$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^{m} + \mathbf{b}^{m+1}) \text{ for } m = 0, 1, \ldots, M-1$$

where ***M*** is number of layers

$\mathbf{a}^0 = \mathbf{p}$ — first layer's input is the external input (P)

$\mathbf{a} = \mathbf{a}^M$ — Last layer's output is the external output (a)

Total Mean-Squared-Error (MSE)

$$MSE = \frac{1}{2}\sum_{patterns}\sum_{outputs}(desired - actual)^2$$

BACKPROPAGATION ALGORITHM

- The algorithm is provided with a set of examples of proper network behavior

$$\{\mathbf{p}_1, \mathbf{t}_1\}\ ,\ \{\mathbf{p}_2, \mathbf{t}_2\}\ ,\ \ldots\ ,\ \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

p_i is an input, t_i is the corresponding output

- By applying each input item $\boldsymbol{p_i}$, the network's output $\boldsymbol{a_i}$ is compared to the target $\boldsymbol{t_i}$
- the algorithm should adjust the network parameters in order to minimize the ERROR function:

$$F(\mathbf{x}) = E[e^2] = E[(t-a)^2]$$

$$F(\mathbf{x}) = E[\mathbf{e}^T\mathbf{e}] = E[(\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})]$$

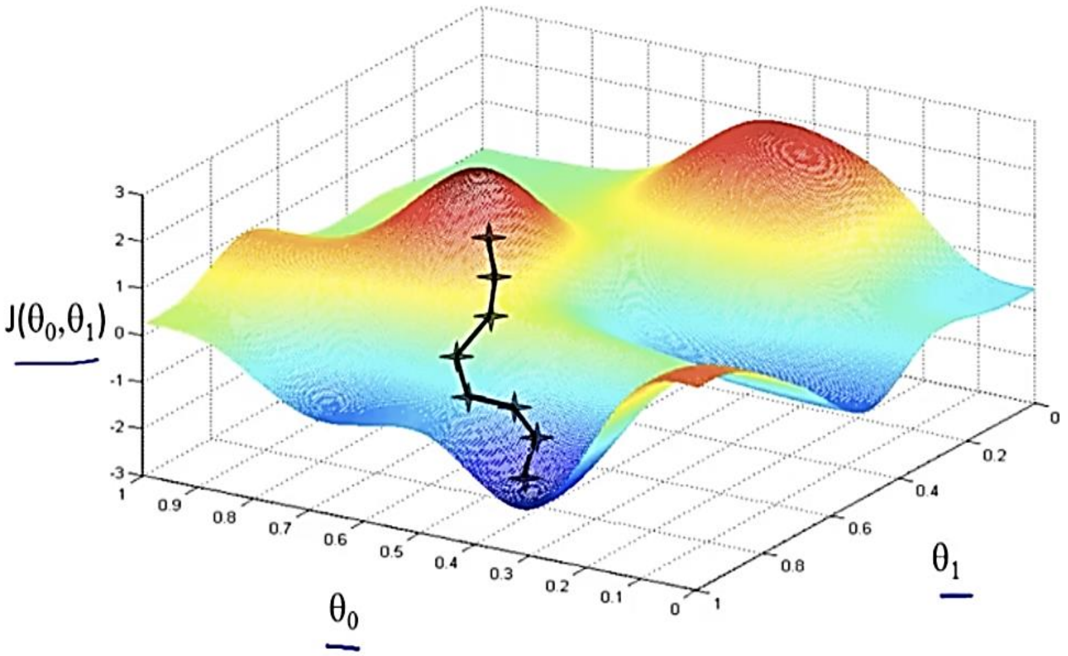
$$a = f(p, w, b)$$

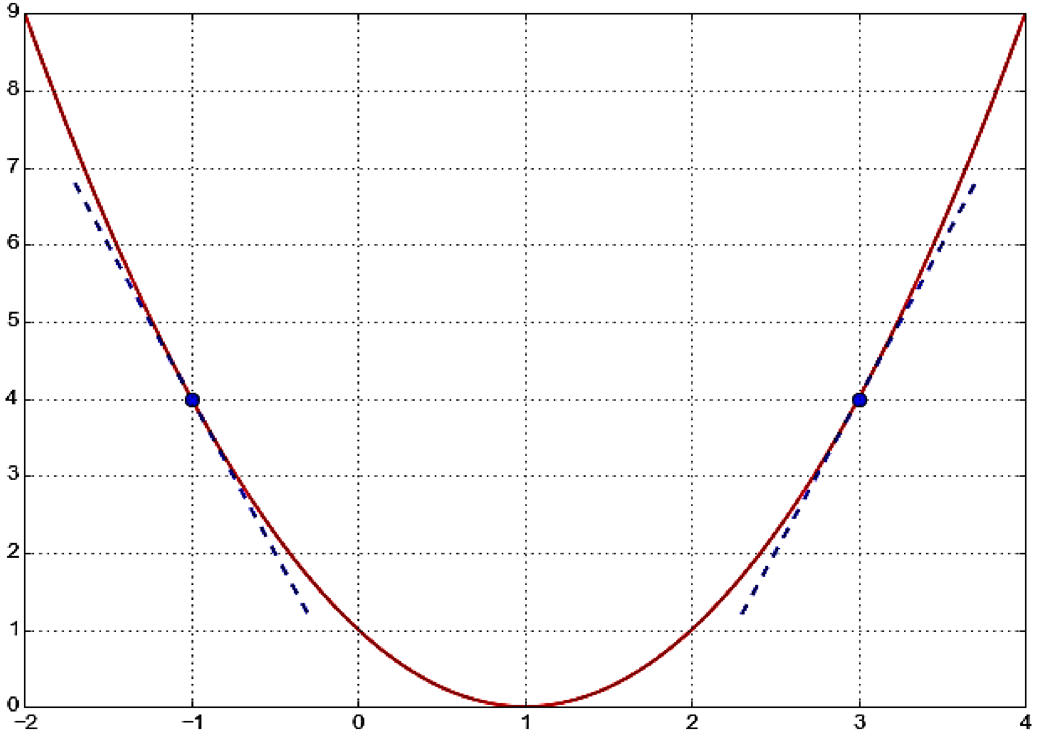
$$\boldsymbol{w}(\boldsymbol{k}+\mathbf{1}) = \boldsymbol{w}(\boldsymbol{k}) + \Delta\boldsymbol{w}$$
$$\boldsymbol{b}(\boldsymbol{k}+\mathbf{1}) = \boldsymbol{b}(\boldsymbol{k}) + \Delta\boldsymbol{b}$$

$$\Delta\boldsymbol{w} = ?$$
$$\Delta\boldsymbol{b} = ?$$

Gradient descent Algorithm

Gradient descent Algorithm





$$\Delta w = -\alpha . \frac{\partial f}{\partial w}$$

The Learning Rate

$$\Delta w = -\alpha . \frac{\partial f}{\partial w}$$

$$a_k = \mathrm{F}(n_k) = \mathrm{F}(\sum_{j=1}^{M-1} w_{k,j} * a_j + b_k)$$

$$n_k = \sum_{j=1}^{M-1} w_{i,j} * a_j + b_k$$

Chain Rule in Calculus

$$\Rightarrow \quad \frac{\partial f}{\partial w_k} = \frac{\partial f}{\partial n_k} * \frac{\partial n_k}{\partial w_k}$$

$$n_k = \Sigma_{j=1}^{M-1} w_{k,j} * a_j + b_k$$

$$\frac{\partial f}{\partial w_k} = \frac{\partial f}{\partial n_k} * \frac{\partial n_k}{\partial w_k}$$

$$\frac{\partial f}{\partial b_k} = \frac{\partial f}{\partial n_k} * \frac{\partial n_k}{\partial b_k}$$

$$\frac{\partial n_k}{\partial w_k} = a_k \qquad \frac{\partial n_k}{\partial b_k} = 1$$

$$\frac{\partial f}{\partial n_k} = s_k \quad - \ sensitivity\ of\ f\ to\ changes\ in\ the\ k.th\ element\ of\ net\ input$$

$$\boldsymbol{w(k+1) = w(k) + \Delta w}$$

$$\boldsymbol{w(k+1) = w(k)} - \alpha . \frac{\partial f}{\partial w(k)}$$

$$\boldsymbol{w(k+1) = w(k) - \alpha . s_k . a_k}$$

$$\boldsymbol{b(k+1) = b(k) - \alpha . s_k}$$

$$s_k = ?$$

- Computing the sensitivity is the process that gives the term **BACK PROPAGATION** because it describes a recurrence relationship in which the sensitivity at layer k is computed from the sensitivity at layer $k+1$

$$s_k = f'(n_k).w_{k+1}.s_{k+1}$$

$$\dot{\mathbf{F}}^m(\mathbf{n}^m) = \begin{bmatrix} \dot{f}^m(n_1^m) & 0 & \dots & 0 \\ 0 & \dot{f}^m(n_2^m) & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & & \dot{f}^m(n_{S^m}^m) \end{bmatrix},$$

$$\mathbf{s}^m \equiv \frac{\partial \hat{F}}{\partial \mathbf{n}^m} = \begin{bmatrix} \frac{\partial \hat{F}}{\partial n_1^m} \\ \frac{\partial \hat{F}}{\partial n_2^m} \\ \vdots \\ \frac{\partial \hat{F}}{\partial n_{S^m}^m} \end{bmatrix}$$

- We compute $\boldsymbol{s_i}$ from the last layer to the first layer.

$$\boldsymbol{s_M} \rightarrow \boldsymbol{s_{M-1}} \rightarrow \boldsymbol{s_{M-2}} \rightarrow \ldots\ldots\ldots \rightarrow \boldsymbol{s_2} \rightarrow \boldsymbol{s_1}$$

$$s_M = ?$$

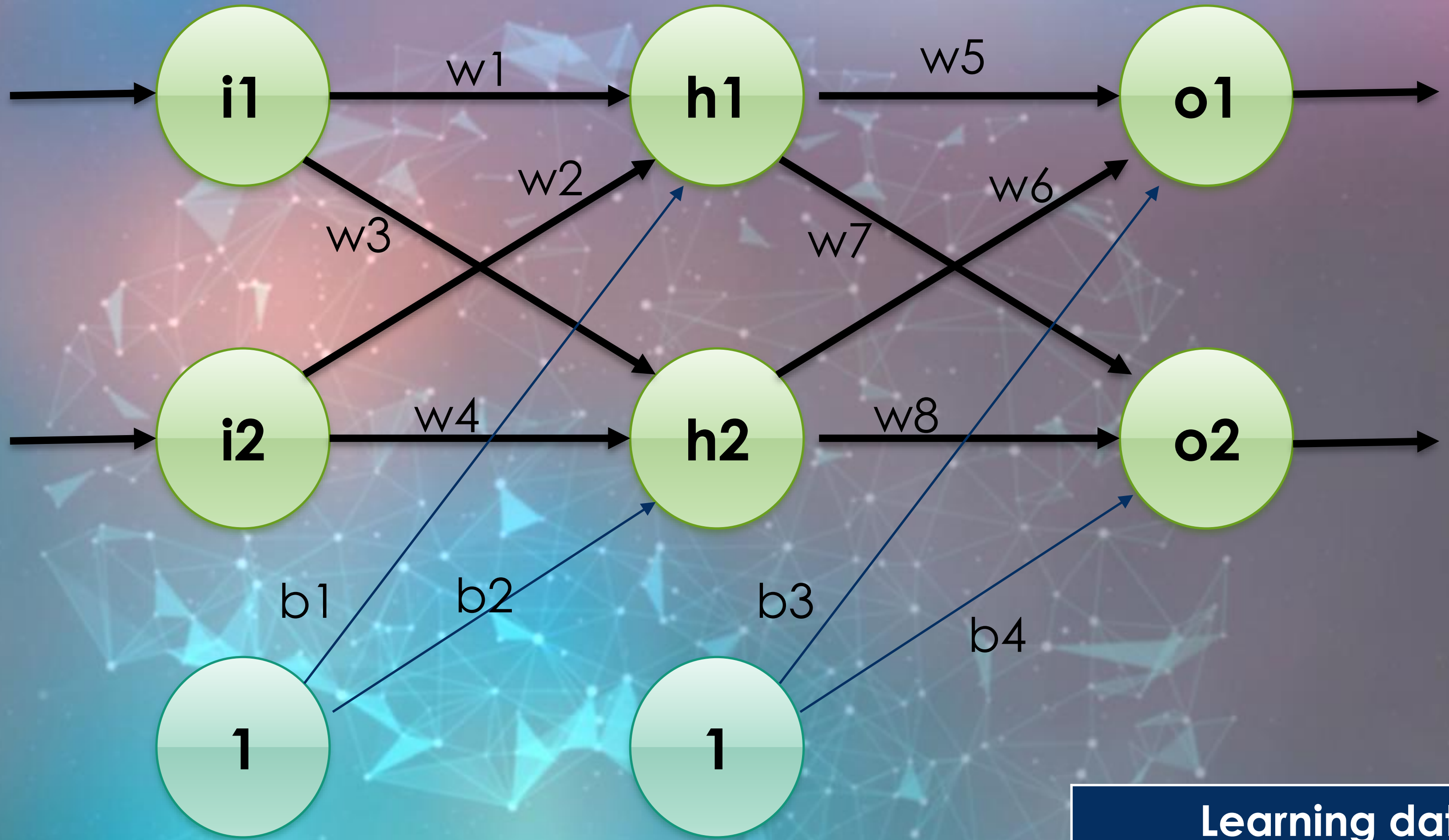
$$s_M = \frac{\partial f}{\partial n_M} = \frac{\partial\ [(t-a)^T.(t-a)]}{\partial n_M} = \frac{\partial}{\partial n_M}\left[\Sigma_{i=1}^{M}(t_i - a_i)^2\right]$$

$$s_M = -2.f'(n_M)(t-a)$$

SUMMARY

1. $$a^0 = p$$
2. $$a^{m+1} = f^{m+1}(w^{m+1}.\,a^m + b^{m+1})$$
3. $$a = a^M$$
4. $$s_M = -2.\,f'(n_M)(t - a)$$
5. $$s_k = f'(n_k).\,w_{k+1}.\,s_{k+1}$$
6. $$w(k+1) = w(k) - \alpha.\,s_k.\,a_k$$
7. $$b(k+1) = b(k) - \alpha.\,s_k$$

Example:

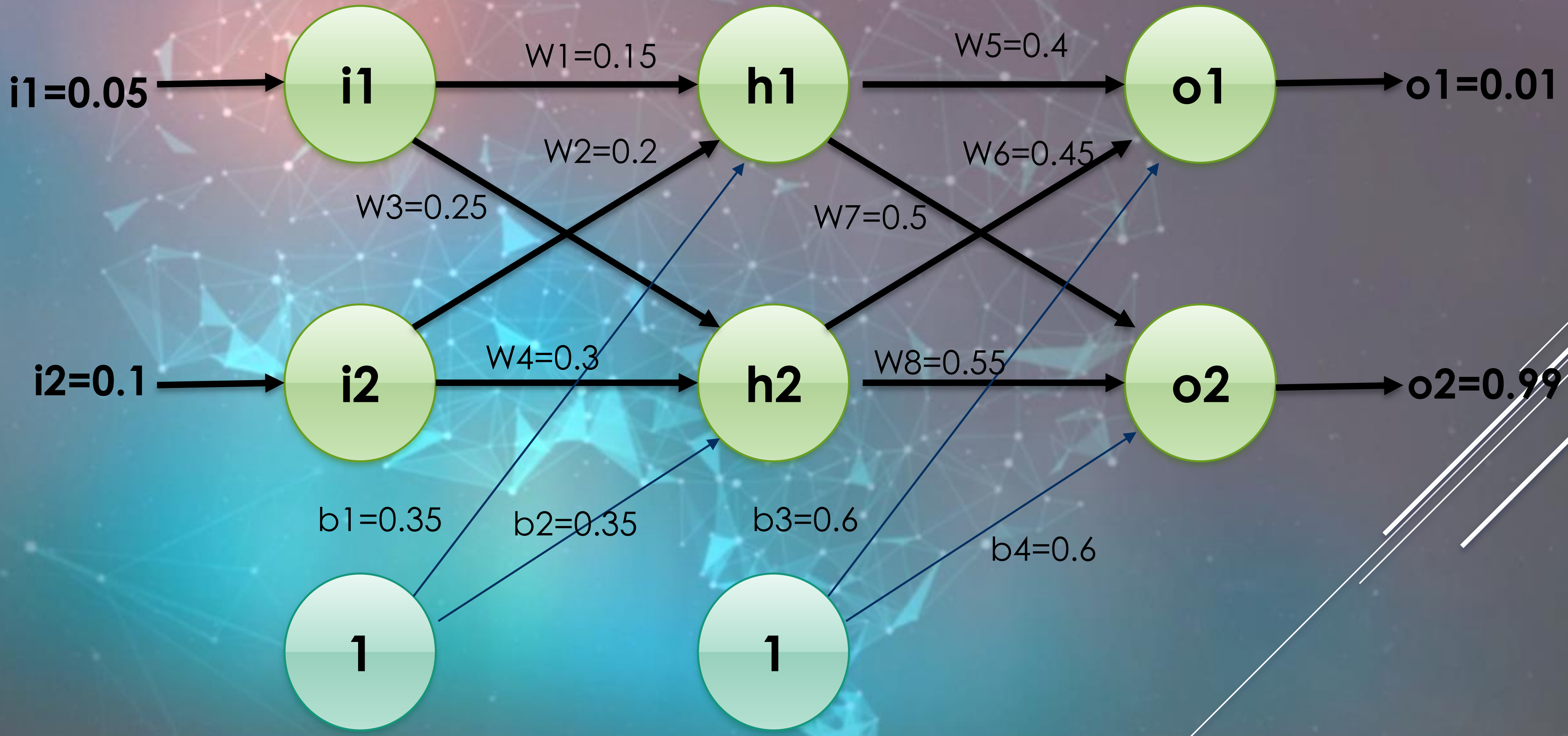


Sigmoid function as a Transfer function

Learning rate=0.5

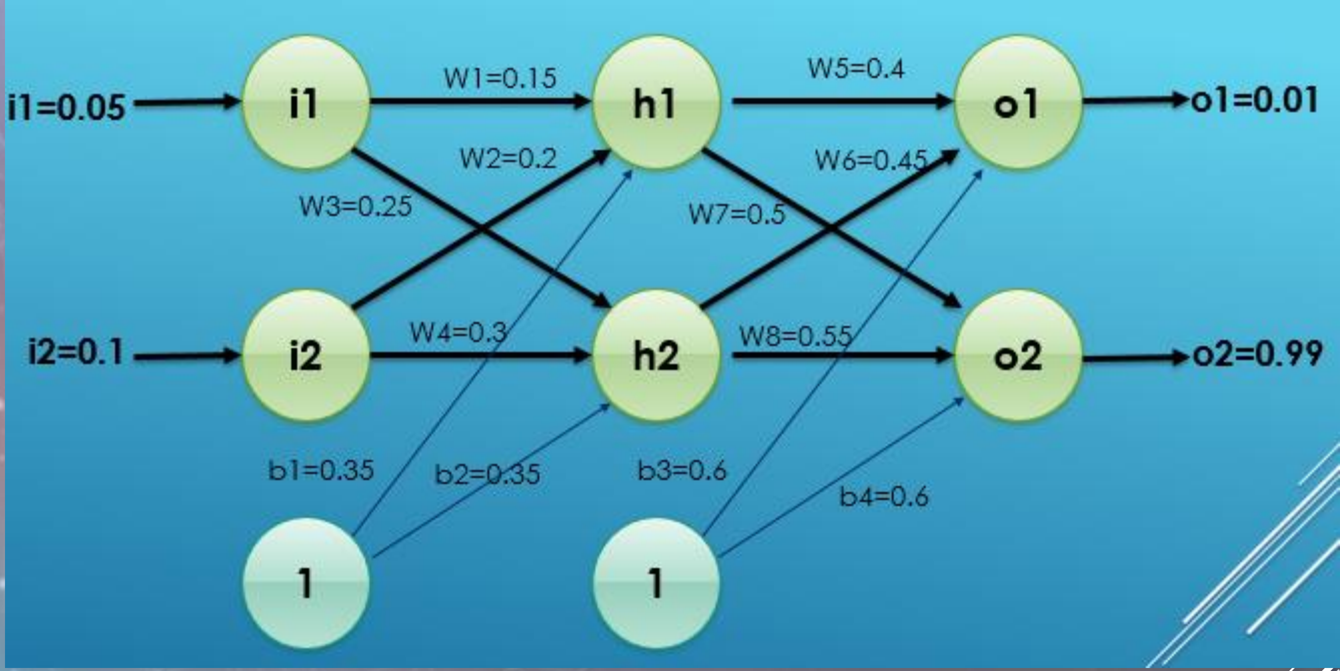
Learning data	
i1=0.05	O1=0.01
i2=0.1	O2=0.99

Random Initial weights



Forward pass

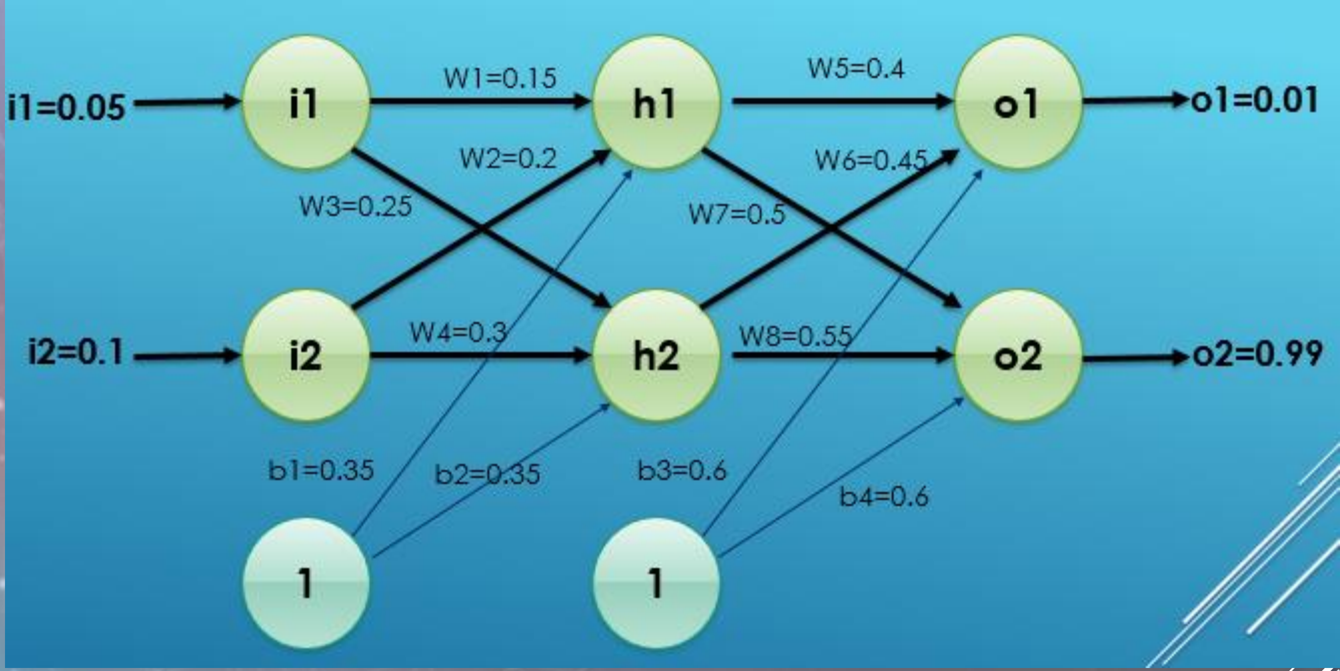
$$n_{h1} = w1 * i1 + w2 * i2 + b1$$
$$n_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35$$
$$n_{h1} = 0.3775$$



$$O_{h1} = \frac{1}{1+e^{-n_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.5932699$$

$$O_{h2} = \frac{1}{1+e^{-n_{h2}}} = 0.5968843$$

Forward pass



$$n_{o1} = w5 * O_{h1} + w6 * O_{h2} + b3$$

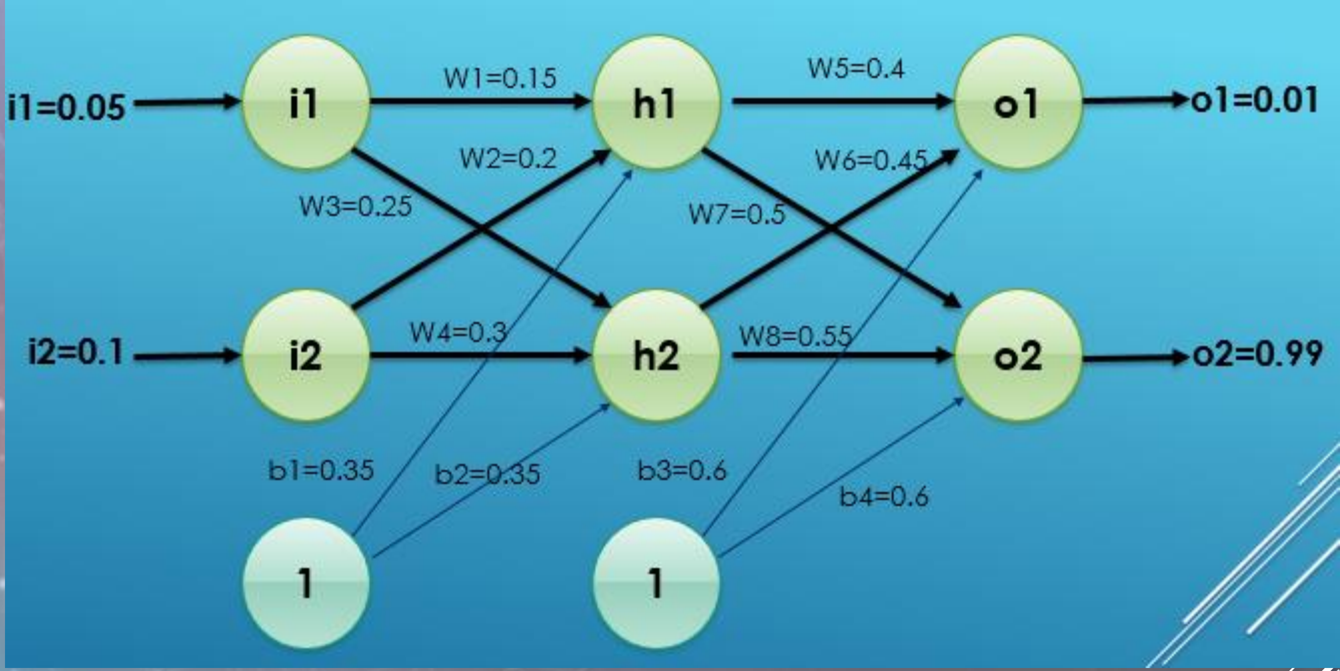
$$n_{o1} = 0.4 * 0.593269 + 0.45 * 0.5968843 + 0.6$$

$$n_{o1} = 1.1059059$$

$$O_{o1} = \frac{1}{1+e^{-n_{o1}}} = \frac{1}{1+e^{-1.1059059}} = 0.75136507$$

$$O_{o2} = \frac{1}{1+e^{-n_{o2}}} = 0.772928465$$

Error calculation



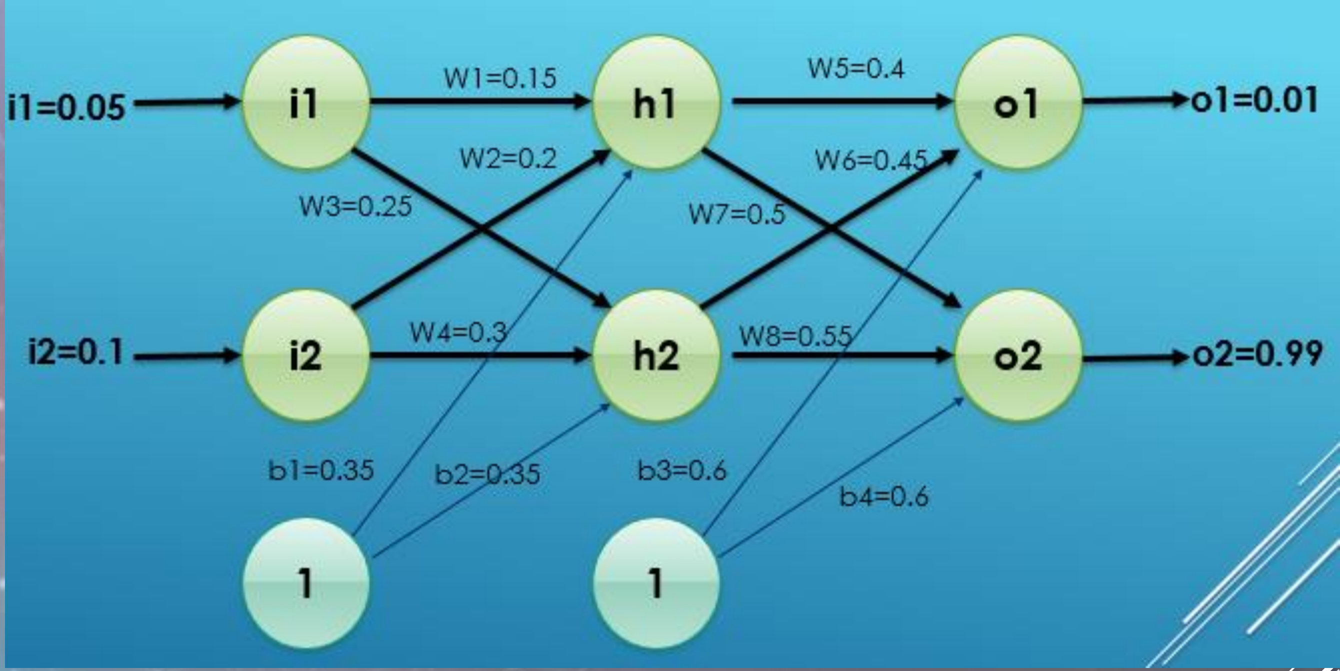
$$E = \Sigma \frac{1}{2} (target - output)^{2}$$

$$E = \frac{1}{2} [(o1 - O_{o1})^{2} + (o2 - O_{o2})^{2}]$$

$$E = \frac{1}{2} [(0.01 - 0.7513650)^{2} + (0.99 - 0.7729284)^{2}]$$

$$E = 0.298371109$$

Backward pass



Consider w5

$$w5' = w5 + \Delta w5$$

$$\Delta w5 = -\alpha . \frac{\partial E}{\partial w5}$$

$$\mathrm{E} = \frac{1}{2}\left[(o1 - O_{o1})^{2} + (o2 - O_{o2})^{2} \right]$$

$$O_{o1} = \frac{1}{1 + e^{-n_{o1}}}$$

$$n_{o1} = w5 * O_{h1} + w6 * O_{h2} + b3$$

$$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{w5}} = \underbrace{\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{O_{o1}}} * \frac{\partial \boldsymbol{O_{o1}}}{\partial \boldsymbol{n_{o1}}}}_{\mathbf{S5}} * \frac{\partial \boldsymbol{n_{o1}}}{\partial \boldsymbol{w5}}$$

$$\frac{\partial E}{\partial w5} = \boxed{\frac{\partial E}{\partial O_{o1}}} * \frac{\partial O_{o1}}{\partial n_{o1}} * \frac{\partial n_{o1}}{\partial w5}$$

$$\mathrm{E} = \frac{1}{2}\left[(o1 - O_{o1})^{2} + (o2 - O_{o2})^{2} \right]$$

$$\frac{\partial E}{\partial O_{o1}} = 2 * 0.5\,(o1 - O_{o1})^{2-1} * (-1) + 0$$

$$\frac{\partial E}{\partial O_{o1}} = -\left(o1 - O_{o1}\right) = -(0.01 - 0.75136507)$$

$$\frac{\partial E}{\partial O_{o1}} = 0.74136507$$

$$\frac{\partial E}{\partial w5} = \frac{\partial E}{\partial O_{o1}} * \boxed{\frac{\partial O_{o1}}{\partial n_{o1}}} * \frac{\partial n_{o1}}{\partial w5}$$

$$O_{o1} = \frac{1}{1 + e^{-n_{o1}}}$$

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

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

$$f'(x) = \frac{e^x}{e^x + 1} * \frac{1 + e^x - e^x}{e^x + 1} = \frac{e^x}{e^x + 1} * \left[\frac{1 + e^x}{e^x + 1} - \frac{e^x}{e^x + 1}\right]$$

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

$$\frac{\partial E}{\partial w5} = \frac{\partial E}{\partial O_{o1}} * \boxed{\frac{\partial O_{o1}}{\partial n_{o1}}} * \frac{\partial n_{o1}}{\partial w5}$$

$$O_{o1} = \frac{1}{1 + e^{-n_{o1}}}$$

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

$$\frac{\partial O_{o1}}{\partial n_{o1}} = O_{o1}(1\text{-}O_{o1}) = 0.75136507(1\text{-}\ 0.75136507)$$

$$\frac{\partial O_{o1}}{\partial n_{o1}} = 0.186815602$$

$$\frac{\partial E}{\partial w5} = \frac{\partial E}{\partial O_{o1}} * \frac{\partial O_{o1}}{\partial n_{o1}} * \boxed{\frac{\partial n_{o1}}{\partial w5}}$$

$$n_{o1} = w5 * O_{h1} + w6 * O_{h2} + b3$$

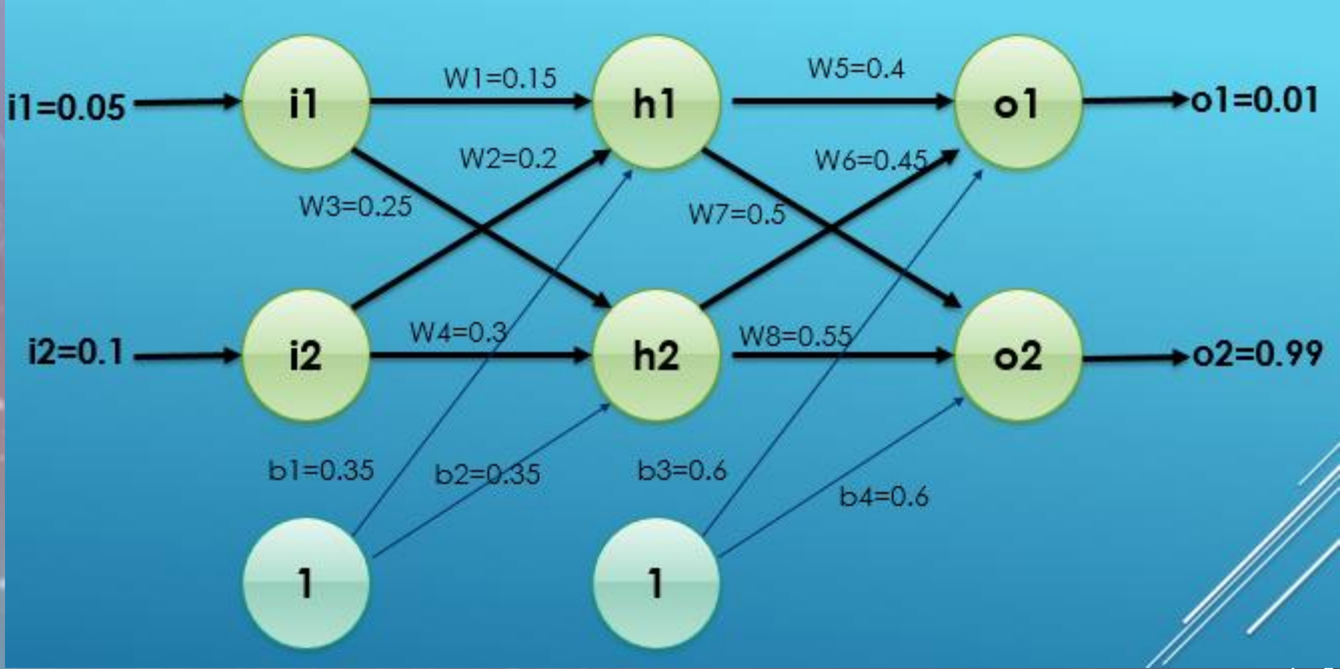
$$\frac{\partial n_{o1}}{\partial w5} = O_{h1} = 0.5932699$$

$$\frac{\partial E}{\partial O_{o1}} = 0.74136507$$

$$\frac{\partial O_{o1}}{\partial n_{o1}} = 0.186815602$$

$$\frac{\partial E}{\partial w5} = 0.74136507 * 0.186815602 * 0.5932699 = 0.082167041$$

$$w5' = w5 - \alpha.\frac{\partial E}{\partial w5}$$



$$w5' = 0.4 - 0.5 * 0.08216704 = 0.35891648$$

$$w6' = 0.408666186$$

$$w7' = 0.511301271$$

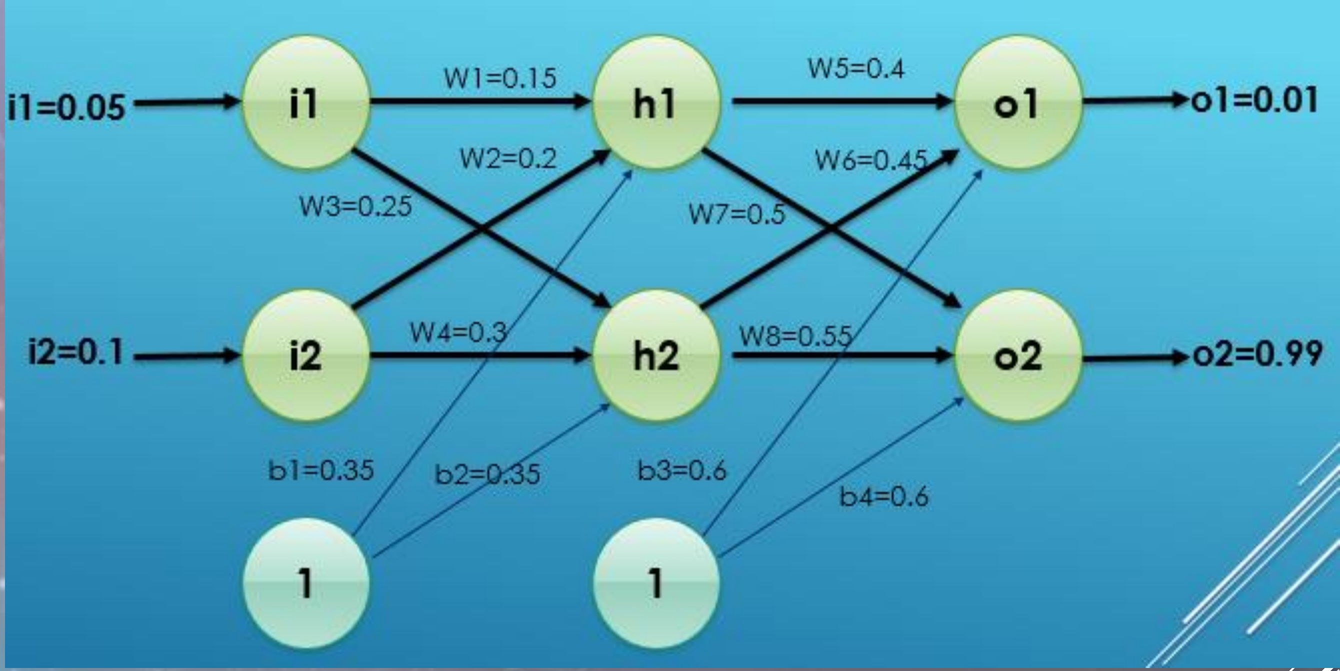
$$w8' = 0.561370121$$

Backward pass

Consider w1

$$w1' = w1 - \alpha . \frac{\partial E}{\partial w1}$$

$$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{w1}} = \frac{\partial \boldsymbol{E}}{\partial O_{h1}} * \frac{\partial O_{h1}}{\partial n_{h1}} * \frac{\partial n_{h1}}{\partial \boldsymbol{w1}}$$



$$\mathrm{E} = \frac{1}{2} [(o1 - O_{o1})^{2} + (o2 - O_{o2})^{2}]$$

$$O_{o1} = \frac{1}{1 + e^{-n_{o1}}} \qquad n_{o1} = w5 * O_{h1} + w6 * O_{h2} + b3$$

$$O_{o2} = \frac{1}{1 + e^{-n_{o2}}} \qquad n_{o2} = w7 * O_{h1} + w8 * O_{h2} + b4$$

$$O_{h1} = \frac{1}{1 + e^{-n_{h1}}}$$

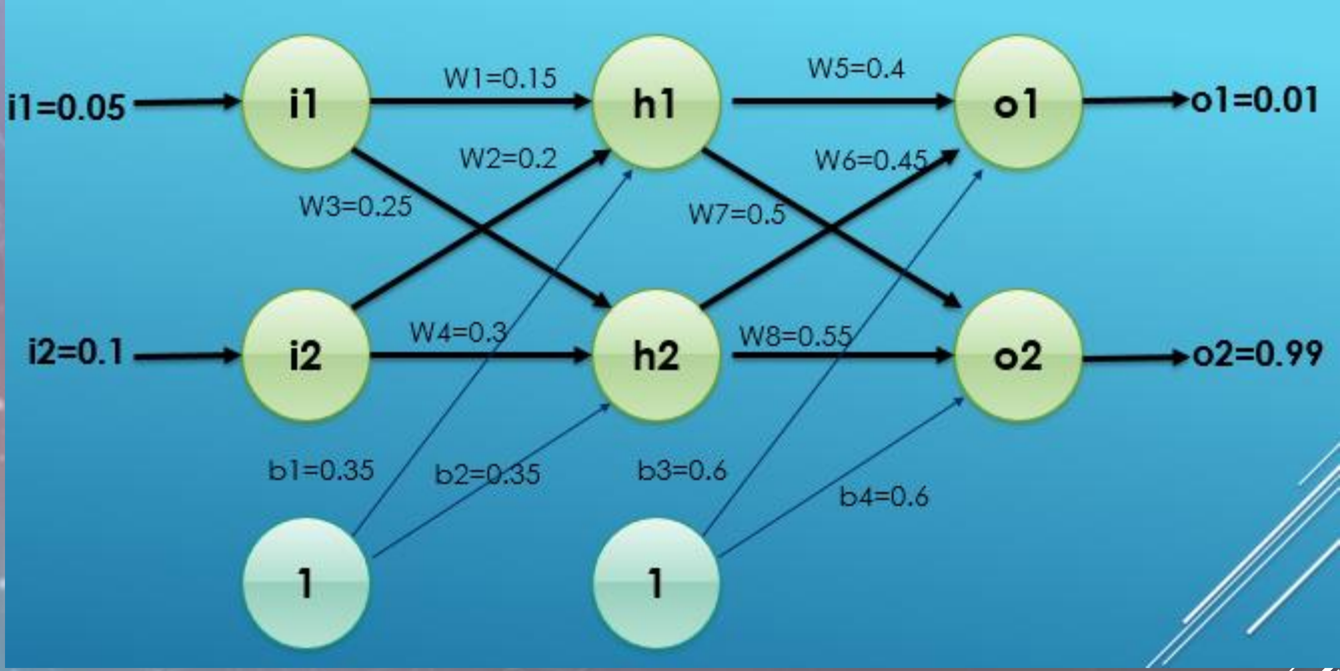
$$n_{h1} = w1 * i_1 + w2 * i_2 + b1$$

$$w1' = 0.194780716$$

$$w2' = 0.19956143$$

$$w3' = 0.24975114$$

$$w4' = 0.29950229$$



Repeat while E > E_stop

VARIATIONS ON BACKPROPAGATION

- The basic algorithm is too slow for most practical applications.
- There are many variations of BP algorithm that provide significant speedup and make the algorithm more practical.

  - Momentum
  - Variable learning rate
  - Conjugate gradient
  - Levenberg-marquardt algorithm

Momentum

The most common is to alter the weight-update rule by making the weight update on the *n* th iteration *depend partially on* the update that occurred during the (*n* -1)th iteration, as follows:

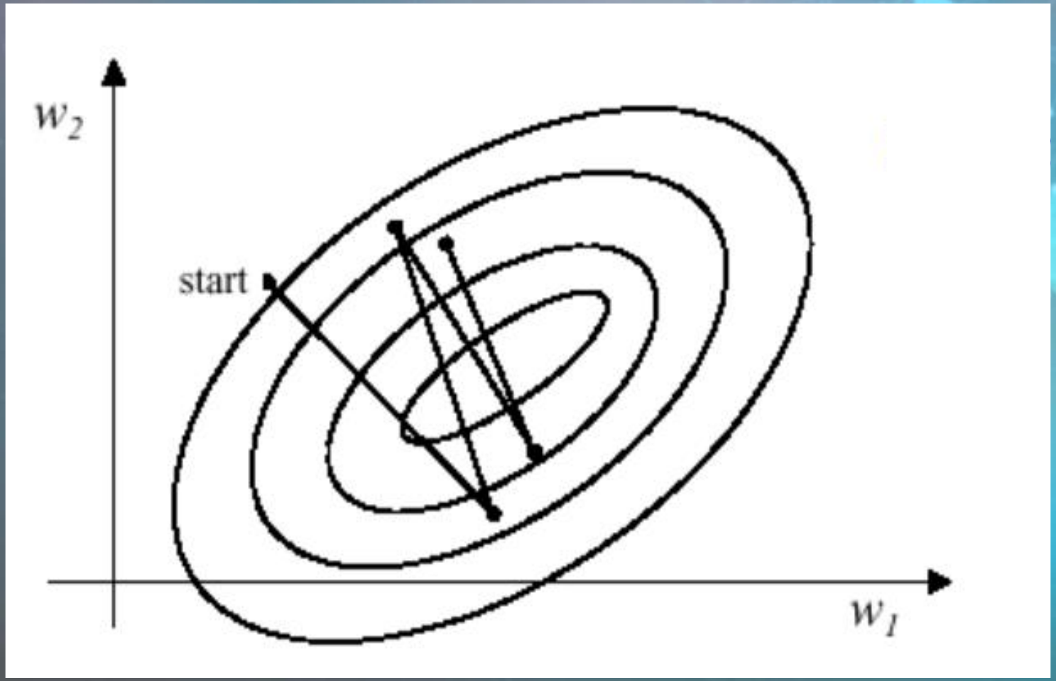
$$\Delta w = -\alpha . \frac{\partial f}{\partial w}$$

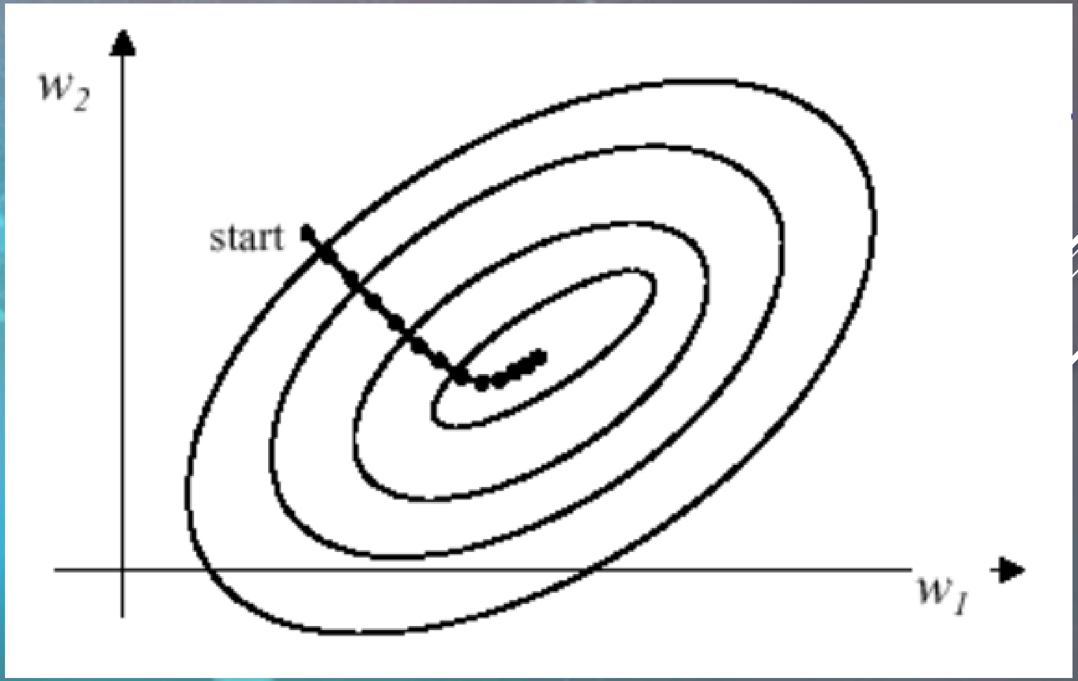
$$\Delta w_k = \gamma . \Delta w_{k-1} - (1-\gamma)\alpha . \frac{\partial f}{\partial w}$$

γ is the Momentum constant

Role of momentum term:

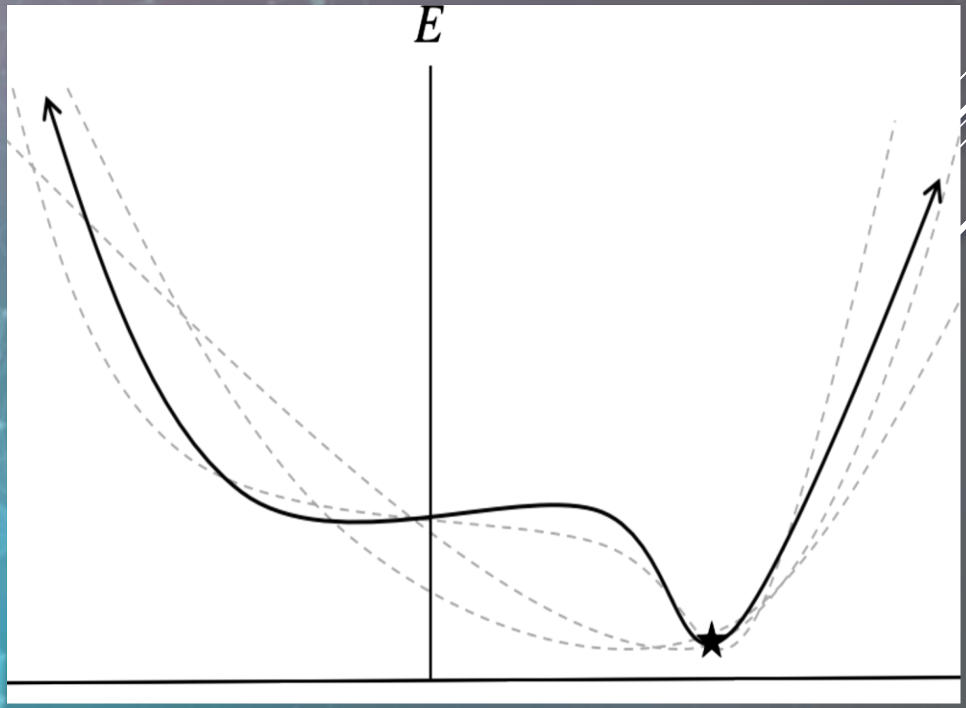
Gradually increase the step size of the search in regions where the gradient is unchanging, thereby speeding convergence.





Variable learning Rate *(VLBP)*

- ➢ we might be able to speed up convergence if we increase the learning rate on flat surfaces and then decrease the learning rate when the slope increases.
- ➢ There are many different approaches for varying the learning rate. We will describe an approach where the learning rate is varied according to the performance of the algorithm



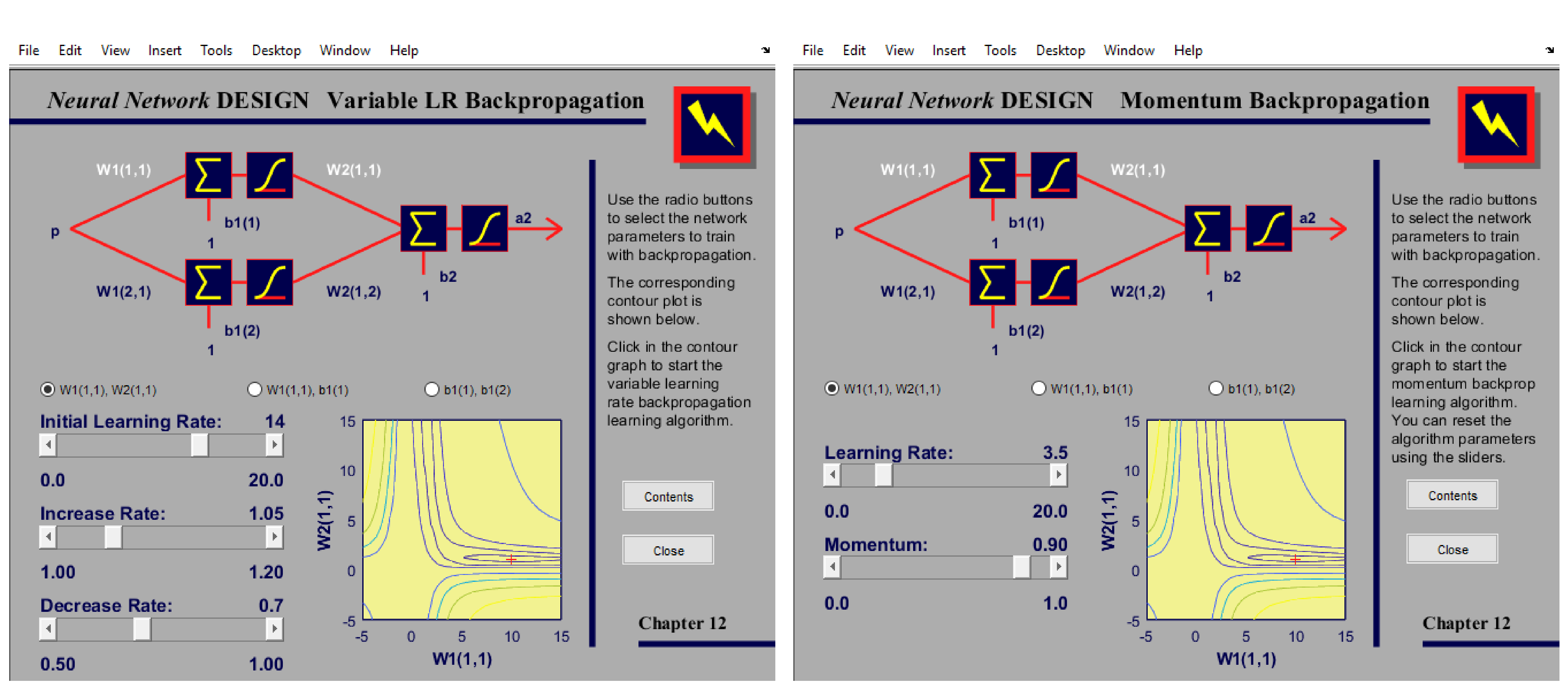
The rules of (*VLBP*) are:

- 1. If the squared error increases by more than some set percentage ζ (typically one to five percent) after a weight update, then the weight update is **discarded**, the learning rate is multiplied by some factor $0 < \rho < 1$, and the momentum coefficient γ (if it is used) is set to zero.
- 2. If the squared error decreases after a weight update, then the weight update is **accepted** and the learning rate is multiplied by some factor $\eta > 1$. If γ has been previously set to zero, it is reset to its original value.

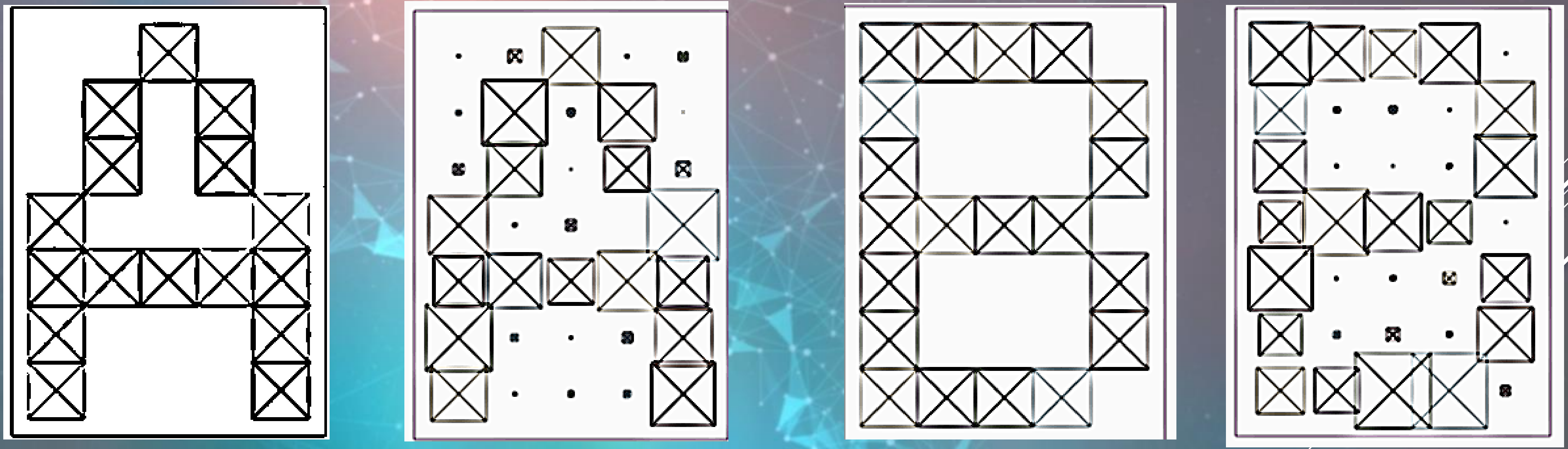
- 3. If the squared error increases by less than ζ , then the weight update is **accepted** but the learning rate is unchanged. If γ has been previously set to zero, it is reset to its original value.

ζ	Max percentage error increase	1-5%
ρ	Learning rate decrease	0.7
η	Learning rate increase	1.05
γ	Momentum constant	0.9

nnd tool box– neural networks design



Optical Character Recognition
OCR



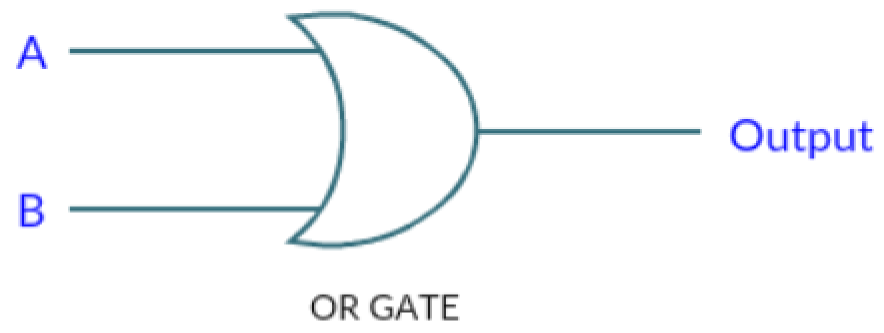
35 inputs / 26 outputs

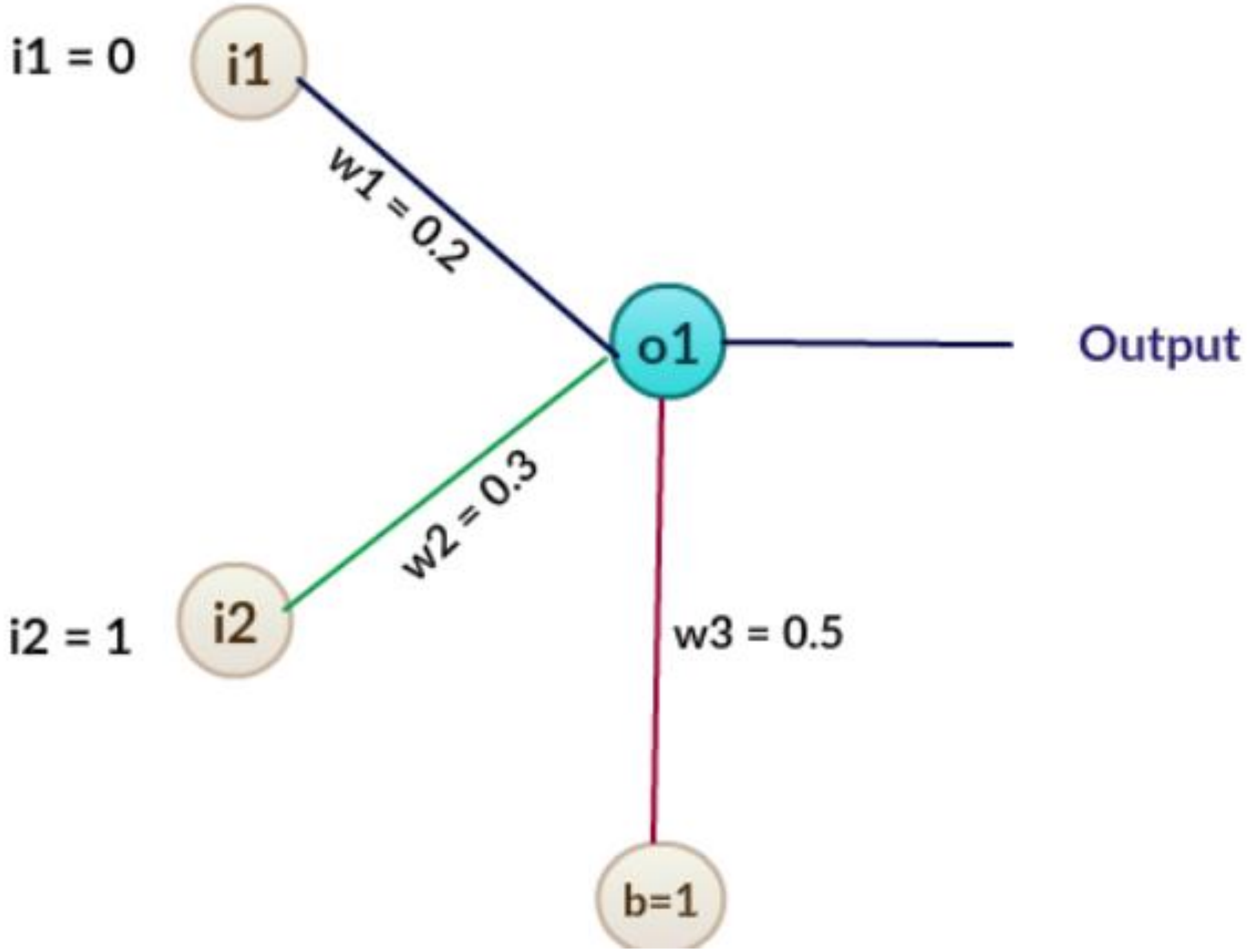
Go to Matlab …

Python example

OR Gate

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	1




i1 = 0
i1
w1 = 0.2
o1
Output
w2 = 0.3
i2 = 1
i2
w3 = 0.5
b=1

Go to python editor …