Minia University
Faculty of Computers & information

Artificial Neural Networks and Deep Learning

Slides By:

A.T. Sarah Osama Talaat

E-mail: SarahOsama.fci@gmail.com

Slides were prepared based on set of references mentioned in the last slide

Let’s Start

Agenda

- Revision
  - Gradient optimization procedures
  - Stochastic gradient optimization
- Hebbian rule
- Learning rules
- Hebb network

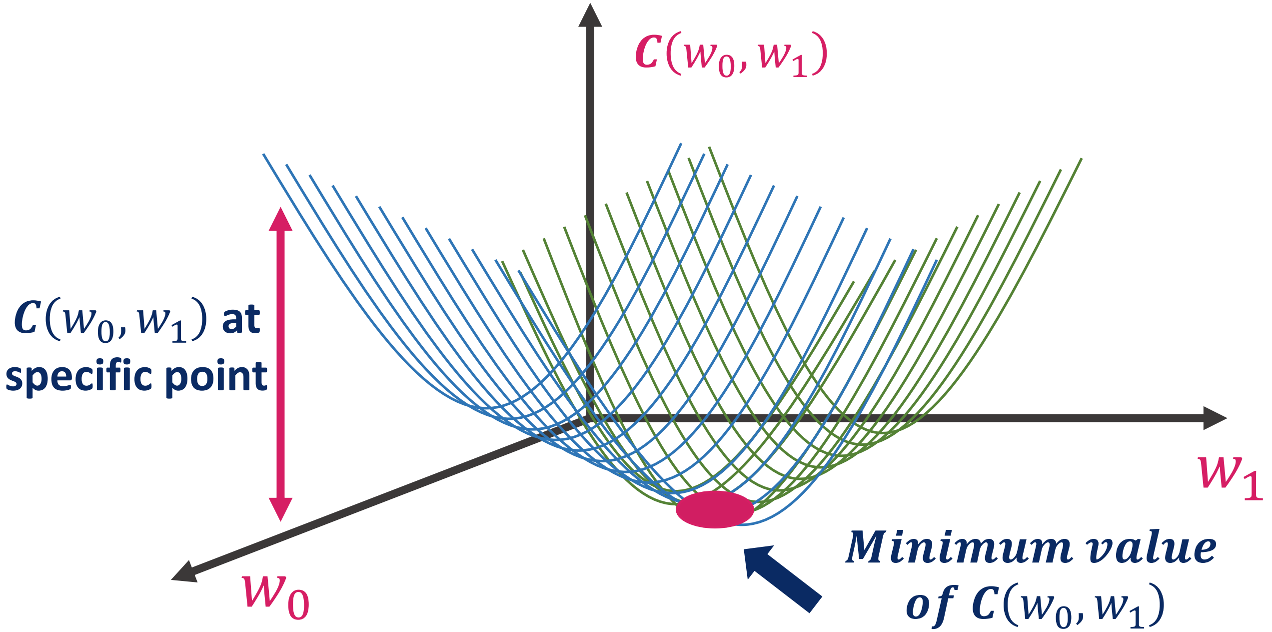
Gradient Optimization Algorithm
Introduction

- The ANN learns by backpropagation of the cost function.
- In order to establish the mathematical basis for some of the following learning procedures (i.e. algorithms) I want to explain briefly what is meant by **gradient descent: the backpropagation of error learning procedure**, for example, involves this mathematical basis and thus inherits the advantages and disadvantages of the gradient descent.

Gradient Optimization Algorithm
Graphical Description

- Assume that, we have two learning algorithm parameters w_0, w_1 and the cost function $C(w_0, w_1)$
- We want to find w_0, w_1 that minimize the cost function c, $C(w_0, w_1)$

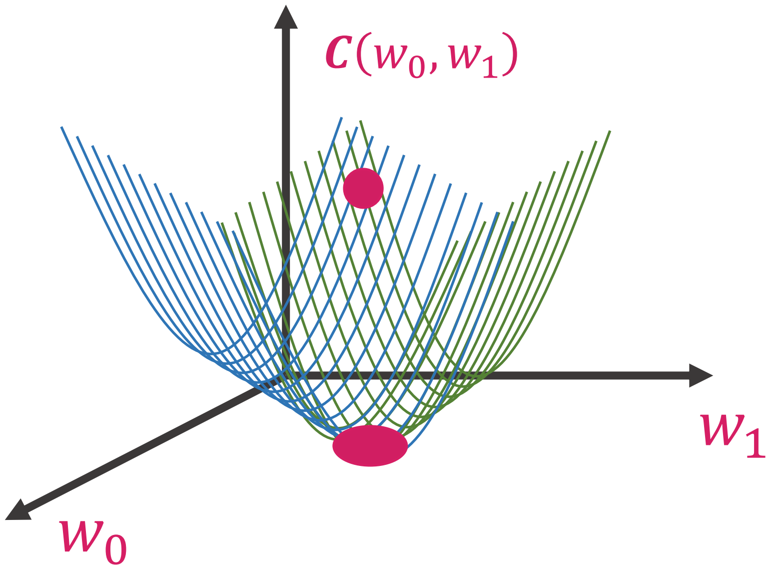
- **$C(w_0, w_1)$ is a convex function which is like a boll**



Gradient Optimization Algorithm
Graphical Description

❑ Gradient descent:

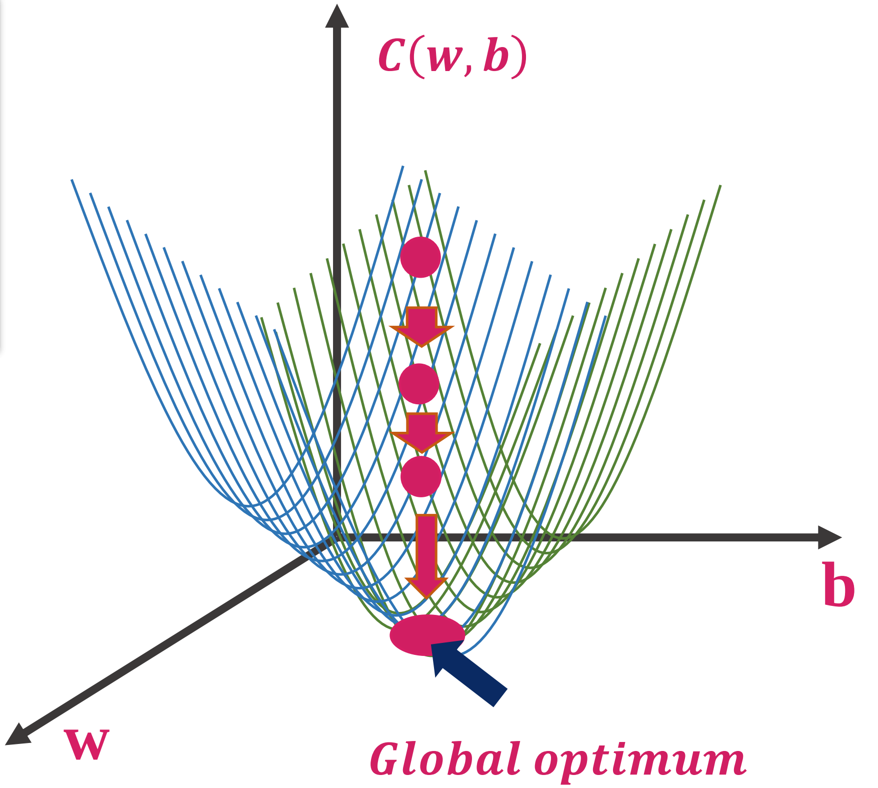
- What we can do to find a good values for the parameters w_0, w_1?
- Initialize w_0 *and* w_1 to some initial values (denoted by pink dot). There many difference initialization methods; random values or assign the values to zero.
- For any convex cost function always assign the values to zeros because we need to start from the same points



Gradient Optimization Algorithm
Graphical Description

- As mentioned before the gradient descent is defined by the following

Gradient descent starts from staring initial point $s = (s_1, s_2, \ldots, s_n)$ and then takes a step in the (steepest) **against direction** (downhill) **of *s***



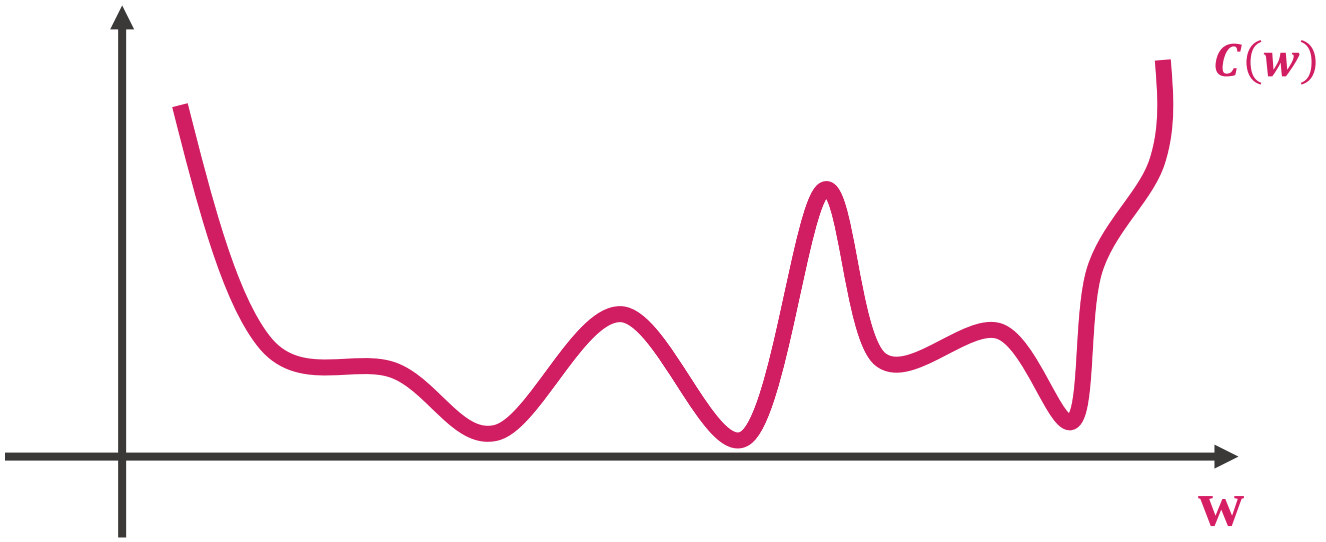
Gradient Optimization Algorithm
Algorithm

Algorithm 4.1: Gradient Decent algorithm

1. **Initialization:** Initialize all parameters to any random number
2. **Repeat until convergence (or gradient gets close to zero)**
3. **Activation:** compute the neuron input $\boldsymbol{net_i} = \sum_{i=0}^{n} \boldsymbol{w_i x_i}$, where $\boldsymbol{i} = \mathbf{0}$ is for the bias
4. Calculate the neuron output by applying one of activation functions to the neuron input $\hat{y} = o_i = \boldsymbol{a}(\boldsymbol{net_i})$
5. **Create the derivative for all parameters** p_i
6. $\boldsymbol{dp_i} := \frac{d\,C}{d\,p_i}$
7. **Learning : adjust the parameters values**
8. $p_i := p_i - \eta \frac{dC}{dp_i}$, where η is the learning rate
9. **End for**

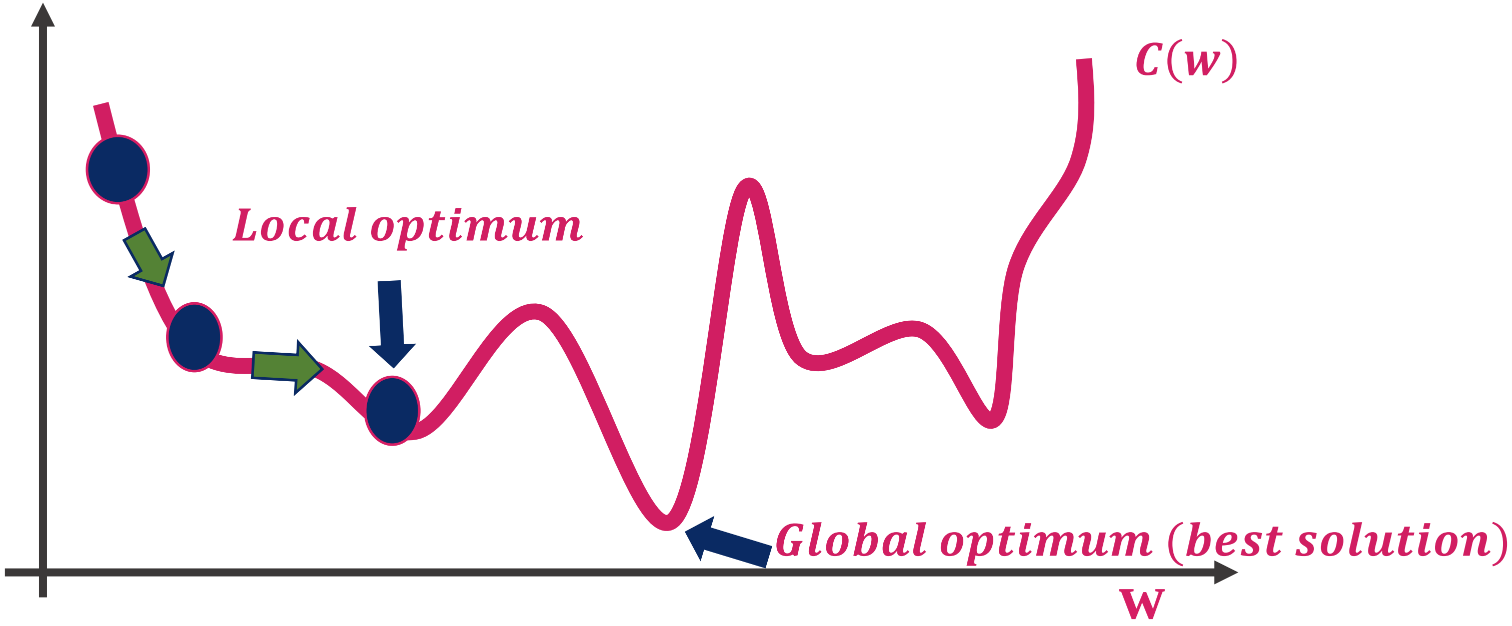
Stochastic Gradient Optimization Algorithm
Graphical Description

- Assume that, we have two learning algorithm parameters w_0, w_1 and the cost function $C(w_0, w_1)$. And, we want to find w_0, w_1 that minimize the cost function c, $C(w_0, w_1)$
- **$C(w_0, w_1)$ is a non-convex function which looks smoothing i.e. curves up and down**



Stochastic Gradient Optimization Algorithm
Graphical Description

- Assume that, we have two learning algorithm parameters w_0, w_1 and the cost function $\boldsymbol{C}(w_0, w_1)$. And, we want to find w_0, w_1 that minimize the cost function c, $\boldsymbol{C}(w_0, w_1)$
- $\boldsymbol{C(w_0, w_1)}$ **is a non-convex function which looks smoothing i.e. curves up and down**



Stochastic Gradient Optimization Algorithm Algorithm

Algorithm 4.1: Stochastic Gradient Decent algorithm

1. **Initialization:** Initialize all parameters to any small random number close to 0 (but not 0)
2. **Foreach instance in the input dataset do**
3. **Forward-propagation: From left to right the neurons are activated by the following**
4. **Activation:** compute the neuron input $\boldsymbol{y}_i = \boldsymbol{net_i} = \sum_{i=0}^{n} \boldsymbol{w_i x_i}$, where $\boldsymbol{i} = \boldsymbol{0}$ is for the bias
5. Calculate the neuron output by applying one of activation functions to the neuron input $\hat{\boldsymbol{y}}_i = o_i = \boldsymbol{a}(\boldsymbol{net_i})$
6. **Measure the error by compare the predicted result $\hat{\boldsymbol{y}}_i$ to the actual result $\boldsymbol{y}_i$(Cost Function)**
7. **Back-propagation: From right to left the error is back-propagated and adjust the weights**
8. **Learning : adjust the parameters values**
9. $p_i := p_i - \eta \frac{dC}{dp_i}$, where η is the learning rate
10. **End foreach**

Hebbian Rule
Is the basis for most other learning rules

- The Hebbian rule is formulated in 1949, which is the **basis** for most of the more complicated learning rules we will discuss in this course.
- We distinguish between the original form and the more general form, which is a kind of principle for other learning rules.

Hebbian Rule

❑ Definition 4.18 (Original Hebbian rule):

- "If ***neuron j*** receives an input from ***neuron i*** and if both neurons are strongly active at the same time**, then increase the weight** $\boldsymbol{w_{i,j}}$(i.e. the strength of the connection between i and j)." Mathematically speaking, the rule is:

$$\Delta \boldsymbol{w_{i,j}} \sim \eta o_i \, a_j$$

with $\Delta \boldsymbol{w_{i,j}}$ being the change in weight from **i to j.**

- Note that, the changes in weight $\Delta \boldsymbol{w_{i,j}}$ are simply added to the weight $\boldsymbol{w_{i,j}}$

Hebbian Rule

- **The change in weight $\Delta w_{i,j}$ from i to j which is proportional to the following factors:**
  - the output o_i of the predecessor neuron **i**
  - the activation a_i of the successor **neuron j,**
  - a constant η, i.e. the learning rate, which will be discussed latter.

Hebbian Rule

- Hebbian Rule was formulated before the specification of technical neurons. Considering that this **learning rule was preferred in binary activations**, it is clear that with the possible activations (1, 0) the weights **will either increase or remain constant** . Sooner or later they would go **ad infinitum**, since they can only be corrected "upwards" when an error occurs.
- This can be compensated by using the activations (-1,1) . Thus, the weights are decreased when the activation of the predecessor neuron dissents from the one of the successor neuron, otherwise they are increased.

Hebbian Rule

❑ Definition 4.19 (Hebbian rule, more general):

- The generalized form of the Hebbian Rule only specifies the proportionality of the change in weight to the product of two undefined functions, but with defined input values

$$\Delta \boldsymbol{w_{i,j}} = \eta \, h(o_i, w_{i,j}) \cdot g(a_j, t_j)$$

Learning Rules

❑Examples of learning rules

- Hebb’s Rule
- Delta Rule (Least Mean Square Rule)
- Hopfield Law
- The Gradient Descent Rule

Learning Rules
Hebb’s rule

❑ Definition 4.20: (Hebb’s rule):

- If a neuron receives an output from another neuron, and if both are highly active (both have same sign), **the weight between the neurons should be strengthened.**
- It means that if two interconnected neurons are both “on” at the same time, then the weight between them should be increased

$$\Delta w_{i,\Omega} = x_i \cdot o_i$$

Learning Rules

Delta rule for learning

❑ Definition 4.21 (Delta rule *or* windrow-Hoff rule):

- Continuously modifying the strengths of the input connections to reduce the difference (the delta) between the desired output value and the actual output of a processing element. Derivative of the **activation function** is used. Delta rule is also called **LME stands for Least mean square.**

Learning Rules
Delta rule for learning

Delta rule for learning versions

- **Offline delta rule**
- **Online Delta rule**

Hebb net
Definition

❑ Definition 4.21: (Hebb net):

- A **Hebb net** is a single layer feedforward neurol network (i.e. network consists of only one layer of variable weights and one layer of output neurons Ω) trained using the Hebb rule. The technical view of an Hebb net is shown in figure 4.3.
- **Remember: Hebb rule is**

$$\Delta w_{i,\Omega} = x_i \cdot o_i$$

- **Algorithm 4.1: Hebb net algorithm**

  Input: training input x_i

  Output: the output unit is t

1. **Initialization:** Initialize all weights to 0
2. **For all** input neurons *i* **do**
3. **Activation:** compute the neuron input $\boldsymbol{net_i} = \sum_{i=0}^{n} \boldsymbol{w_i x_i}$, where $\boldsymbol{i} = \mathbf{0}$ is for the bias
4. Calculate the neuron output by applying one of activation functions to the neuron input $o_i = \boldsymbol{a(net_i)}$
5. **Learning: adjust the weights**
6. $\Delta \boldsymbol{w_i} := x_i \cdot t_i$
7. $\boldsymbol{w_{i,new}} := \boldsymbol{w_{i,old,}} + \Delta \boldsymbol{w_i}$
8. **Learning : like the weights adjust the weight of bias** $\boldsymbol{b_{new,}} := \boldsymbol{b_{old,}} + t_i$
9. **End for**

Hebb net
Example

❑ Example:

- Construct a Hebb network that is used Hebb's learning algorithm to performs AND function
- Assume that the bias neuron has input 1
- The training samples are

x_1	x_2	y $= x_1$ and x_2
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

Hebb net
Example

❑ Example:

- By using Hebb's algorithm and the following training dataset construct an Sigle layer feedforward that performs AND function.
- Assume that the bias neuron has input 1.
- The training samples are

$\boldsymbol{p_1}$ = (1,1) and $\boldsymbol{t_1}$ = (1)
$\boldsymbol{p_2}$ = (1,-1) and $\boldsymbol{t_2}$ = (-1)
$\boldsymbol{p_3}$ = (-1,1) and $\boldsymbol{t_3}$ = (-1)
$\boldsymbol{p_4}$ = (-1,-1) and $\boldsymbol{t_4}$ = (-1)

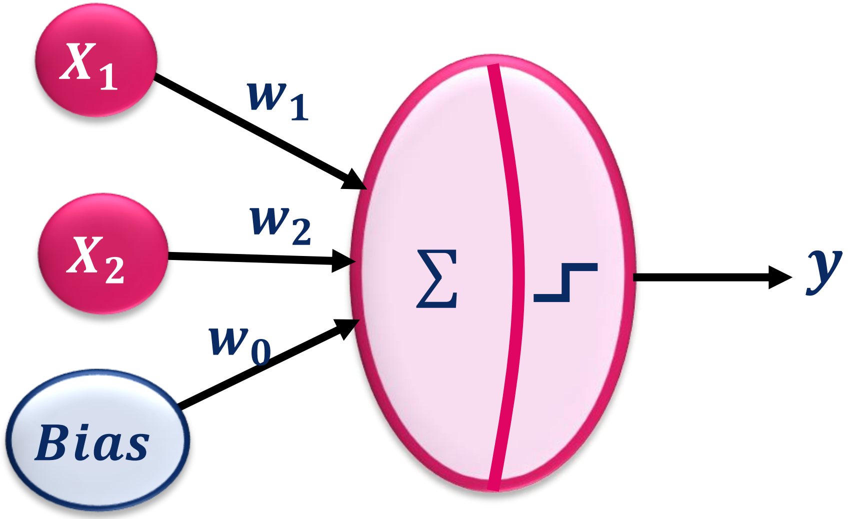
Hebb net
Example

❑ Solution:

- The and gate works as following

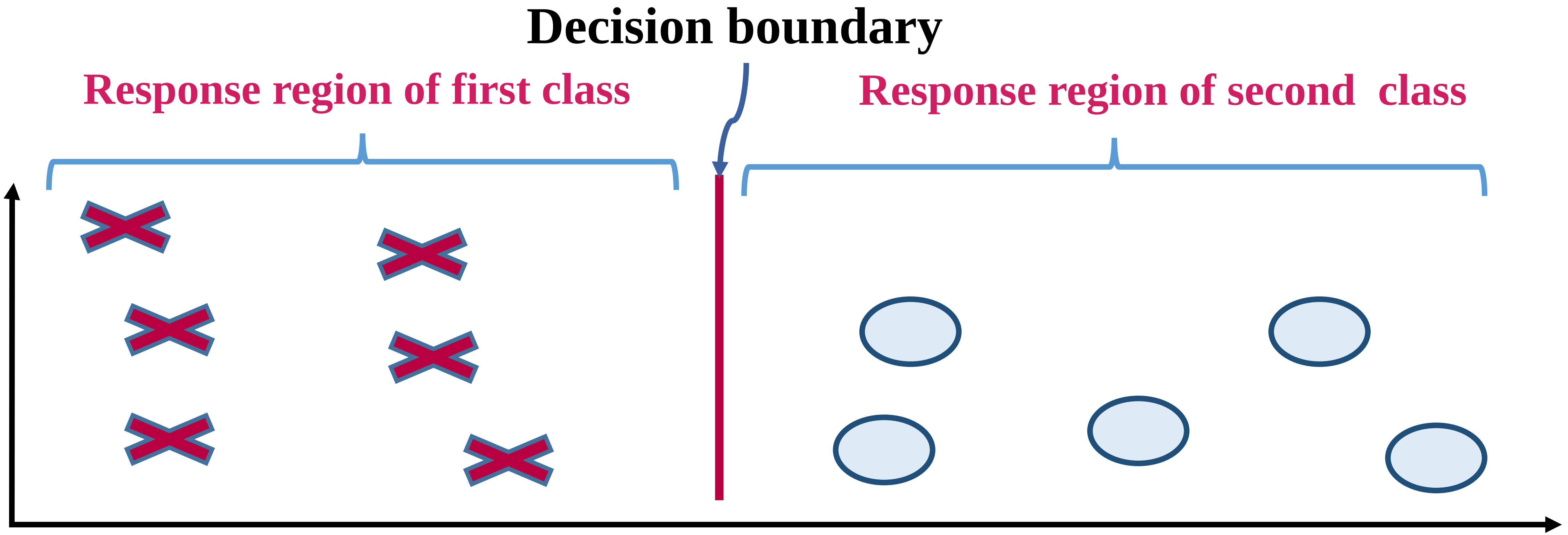
x_1	x_2	t $=x_1$ and x_2
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

- Construct the single layer feedforward architecture for the **and** gate



- In this example we have 4 patterns which are used as a training samples (inputs).
- Each pattern consists of $\mathbf{x_1}, \mathbf{x_1}$ and bias

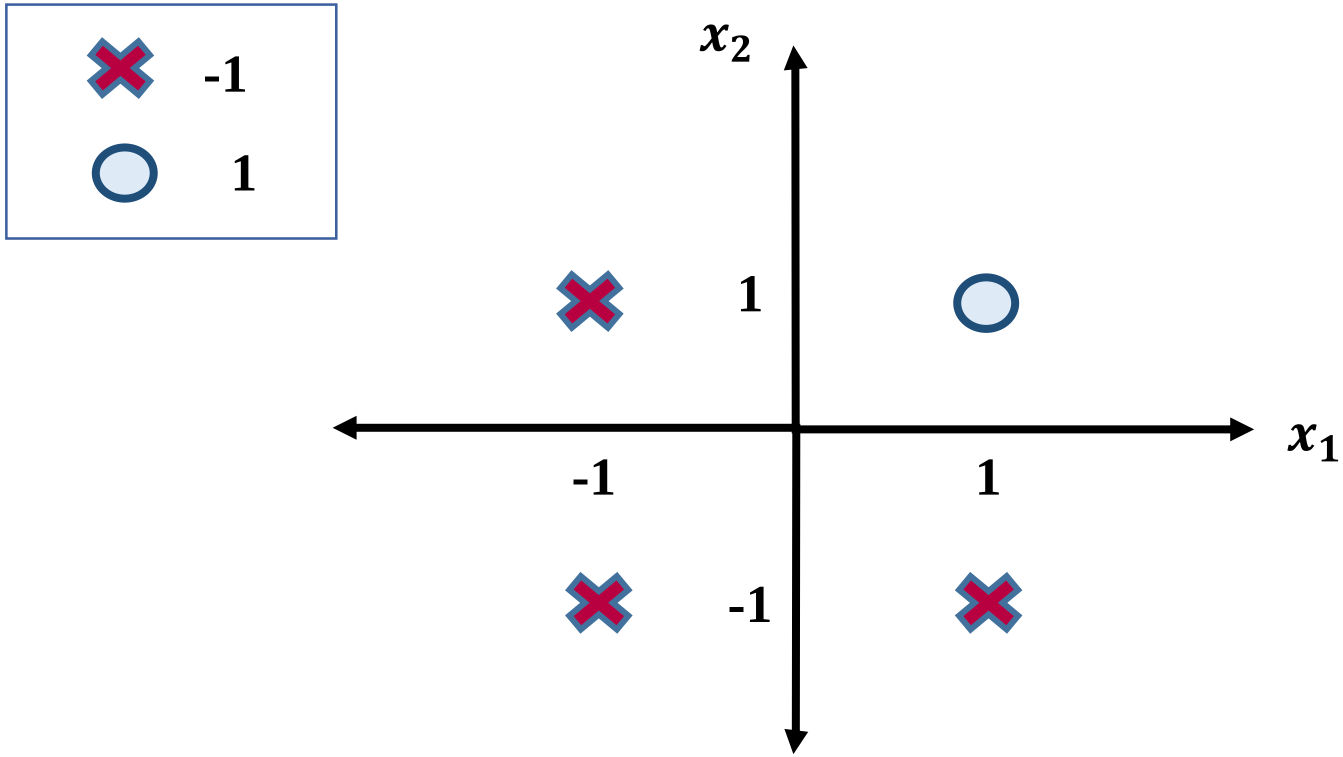
Hebb net
Remember



Linear data in 2D
(Discrete Data)

Hebb net
Response regions

❑ Example: Response region for the AND gate



And gate

The And gate (for bipolar data)

x_1	x_2	y = x_1 and x_2
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

Hebb net
Example

❑ Solution:

- In this example we use the following activation function (Let $\boldsymbol{\theta}$ =0).

$$a(net_i) = \begin{cases} 1, & if\ net_i \geq \theta \\ -1, & if\ net_i < \theta \end{cases}$$

Hebb net
Example

- **Solution: Training phase**
  - **Train the neuron by using the first pattern (1,1,1), the target here is 1**
  - Initialize all weights to 0
  - **Activation:** compute the neuron output $\boldsymbol{o_i} = \sum_{i=0}^{n} \boldsymbol{w_i x_i}$

    $\boldsymbol{net_1} = \sum_{i=0}^{n} \boldsymbol{w_i x_i} = (1\times 0) + (1\times 0) + (1\times 0) = 0$

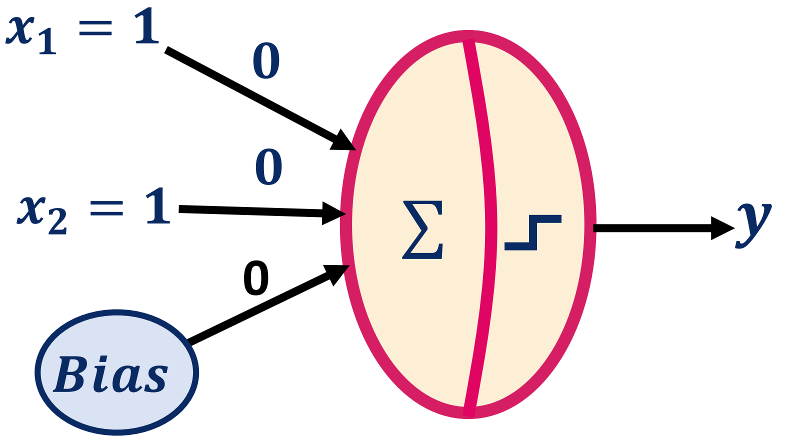
    $o_1 = a(net_1) = 1\ (\because net_1 \geq 0\ \therefore a(net_1) = 1)$
  - **Learning rules:**
    - Adjust the weights: $\boldsymbol{w_{i,new}} := \boldsymbol{w_{i,old,}} + x_i \cdot t$

      $\boldsymbol{w_{1,new}} := 0 + (1 \times 1) = 1$

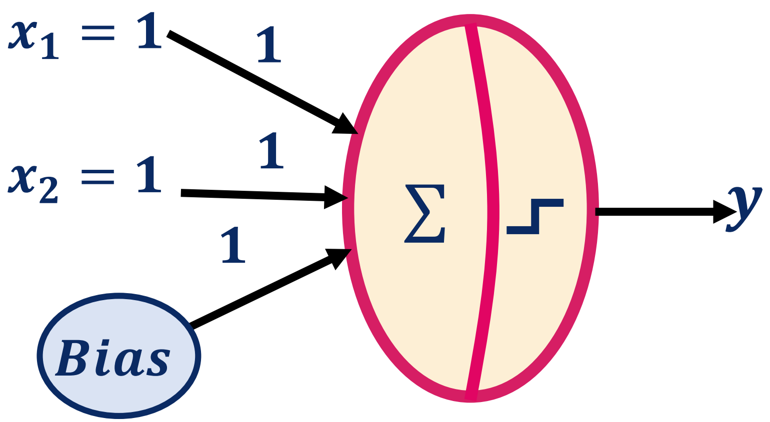
      $\boldsymbol{w_{2,new}} := 0 + (1 \times 1) = 1$
    - Adjust the weight of bias: $\boldsymbol{b_{new,}} := \boldsymbol{b_{old,}} + t$

      $\boldsymbol{b_{new}} := 0 + 1 = 1$

The neuron before training



The neuron after training



Hebb net
Example

The response regions for the AND gate after the first input pattern

- ***The decision boundary equation*** **is**

$$b + w_1 x_1 + w_2 x_2 = 0$$

- **The weights after train the first patter are**

$$w_1\mathbf{=1,}\ w_2 = 1 \textbf{ and } \mathrm{b} = 1$$

- **We will have the following separation line**

$$\because \boldsymbol{y = 1\ nad\ y > 0}$$

$$1 + x_1 + x_2 \geq 0 \Rightarrow x_1 + x_2 \geq -1$$

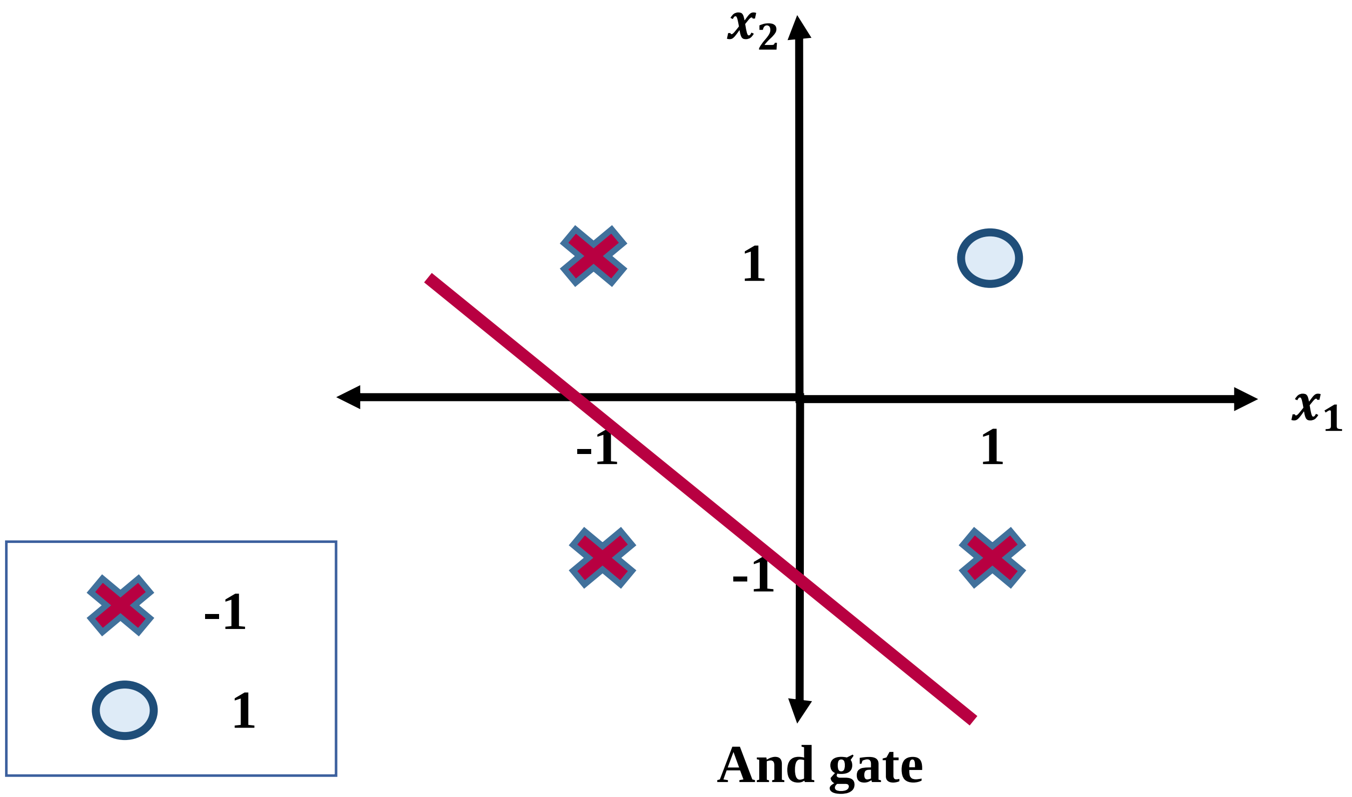
The And gate (for bipolar data)

x_1	x_2	y $=x_1$ and x_2
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

Hebb net
Example

❑ The response regions for the AND gate after the first input pattern

- **We will have the following separation line:** $1 + x_1 + x_2 \geq 0 = x_1 + x_2 \geq -1$



The And gate (for bipolar data)

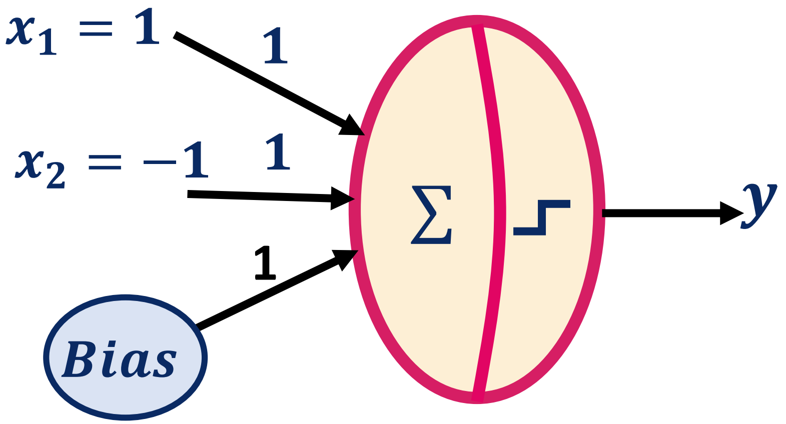
x_1	x_2	y =x_1 and x_2
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

Hebb net
Example

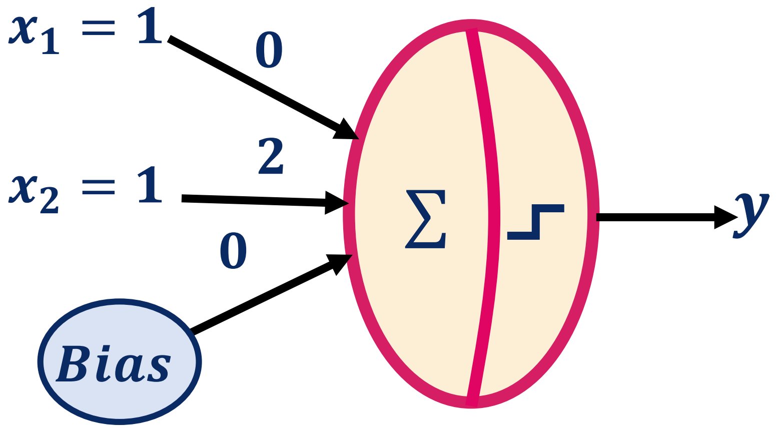
❑ Solution: Training phase

- **Train the neuron by using the second pattern (1,-1,1), the target here is -1**
- **Activation:** compute the neuron output $\boldsymbol{o_i} = \sum_{i=0}^{n} \boldsymbol{w_i x_i}$
  $\boldsymbol{net_2} = \sum_{i=0}^{n} \boldsymbol{w_i x_i} = (1\times 1) + (\text{-}1\times 1) + (1\times 1) = 1$
  $o_2 = a(net_2) = 1\ (\because net_2 \geq 0\ \therefore a(net_2) = 1)$
- **Learning rules:**
  - Adjust the weights: $\boldsymbol{w_{i,new}} := \boldsymbol{w_{i,old,}} + x_i \cdot t$
    $\boldsymbol{w_{1,new}} \coloneqq 1 + (1 \times -1) = 0$
    $\boldsymbol{w_{2,new}} \coloneqq 1 + (-1 \times -1) = 2$
  - Adjust the weight of bias: $\boldsymbol{b_{new,}} := \boldsymbol{b_{old,}} + t$
    $\boldsymbol{b_{new}} \coloneqq 1 + (-1) = 0$

The neuron before training



The neuron after training



Hebb net
Example

The response regions for the AND gate after the second input pattern

- ***The decision boundary equation*** **is**

$$b + w_1 x_1 + w_2 x_2 = 0$$

- **The weights after train the first patter are**

$$w_1 = \mathbf{0},\ w_2 = 2 \text{ and } b = 0$$

- **We will have the following separation line**

$$\because \boldsymbol{y = -1\ nad\ y < 0}$$

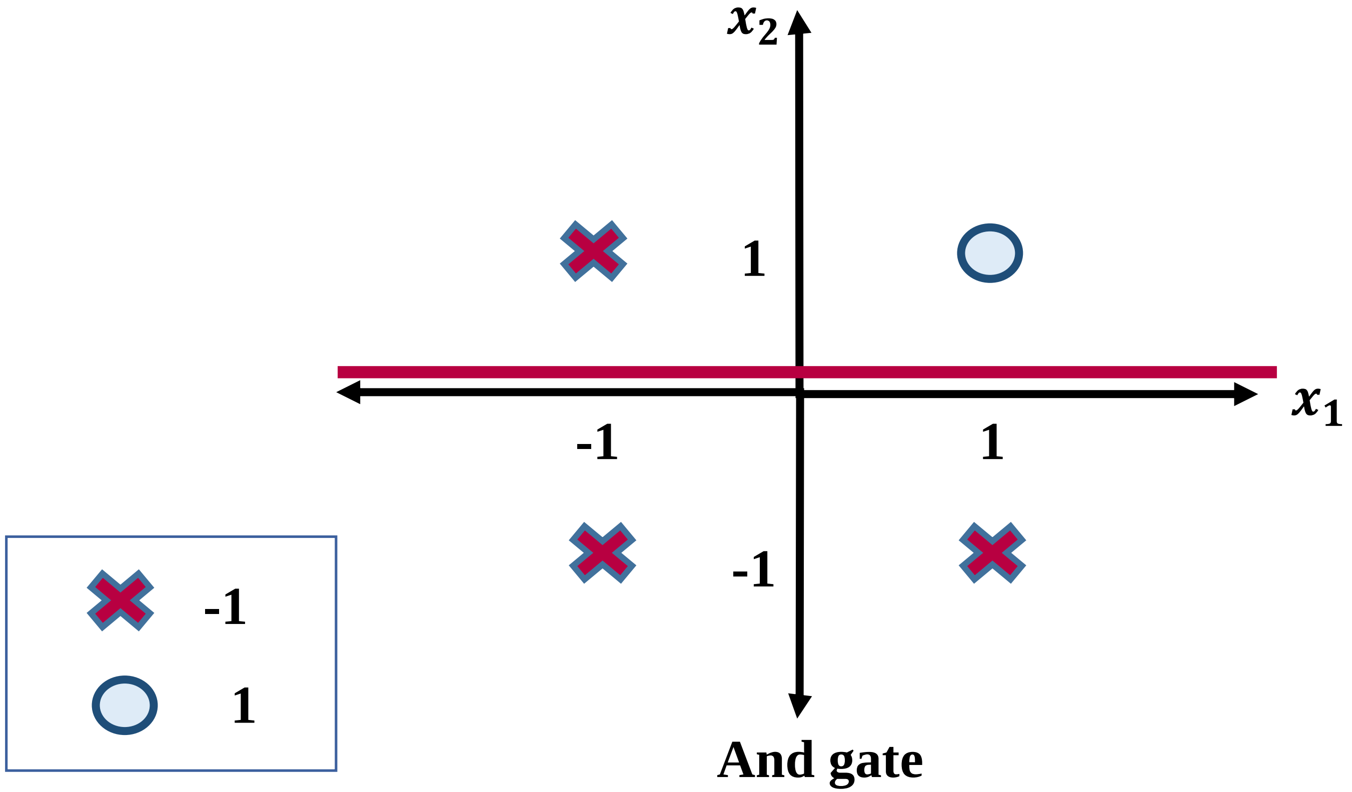
$$2x_2 = 0 \Rightarrow x_2 = 0$$

The And gate (for bipolar data)

x_1	x_2	y = x_1 and x_2
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

Hebb net
Example

- **The response regions for the AND gate after the second input pattern**
  - **We will have the following separation line:** $x_2 = 0$



The And gate (for bipolar data)

x_1	x_2	y = x_1 and x_2
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

Hebb net
Example

- **Solution: Training phase**
  - **Train the neuron by using the third pattern (-1,1,1), the target here is -1**
  - **Activation:** compute the neuron output $\boldsymbol{o_i} = \sum_{i=0}^{n} \boldsymbol{w_i x_i}$

    $\boldsymbol{net_3} = \sum_{i=0}^{n} \boldsymbol{w_i x_i} = (1\times 0) + (-1\times 0) + (1\times 2) = 2$

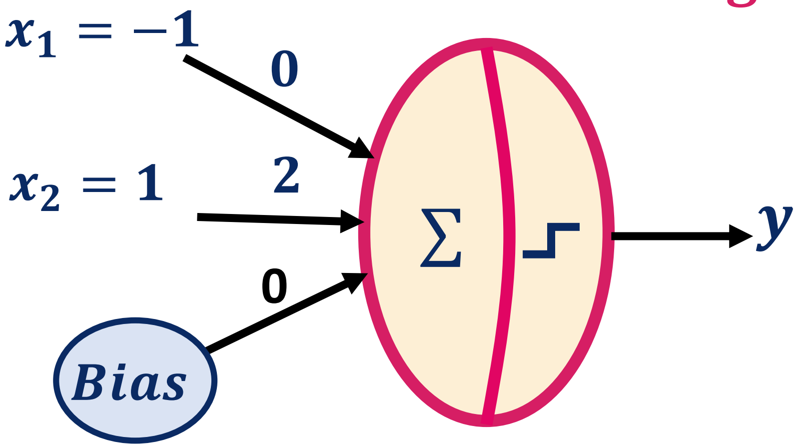
    $o_3 = a(net_3) = 1\ (\because net_3 \geq 0 \ \therefore a(net_3) = 1)$
  - **Learning rules:**
    - Adjust the weights: $\boldsymbol{w_{i,new}} := \boldsymbol{w_{i,old,}} + x_i \cdot t$

      $\boldsymbol{w_{1,new}} := 0 + (-1 \times -1) = 1$

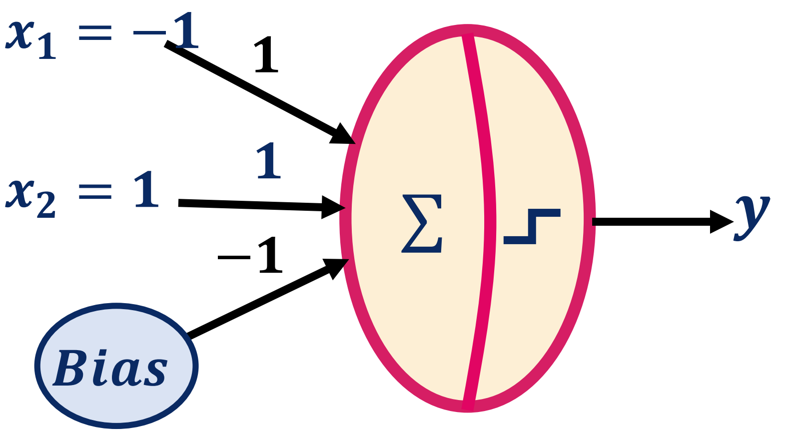
      $\boldsymbol{w_{2,new}} := 2 + (1 \times -1) = 1$
    - Adjust the weight of bias: $\boldsymbol{b_{new,}} := \boldsymbol{b_{old,}} + t$

      $\boldsymbol{b_{new}} := 0 + (-1) = -1$

The neuron before training



The neuron after training



Hebb net
Example

- The response regions for the AND gate after the third input pattern

  - ***The decision boundary equation*** **is**

$$b + w_1 x_1 + w_2 x_2 = 0$$

  - **The weights after train the first patter are**

$$w_1\mathbf{=1,}\ w_2 = 1 \textbf{ and } \mathrm{b = -1}$$

  - **We will have the following separation line**

$$\because \boldsymbol{y = -1\ nad\ y < 0}$$

$$-1 + x_1 + x_2 = 0 \Rightarrow x_1 + x_2 = 1$$

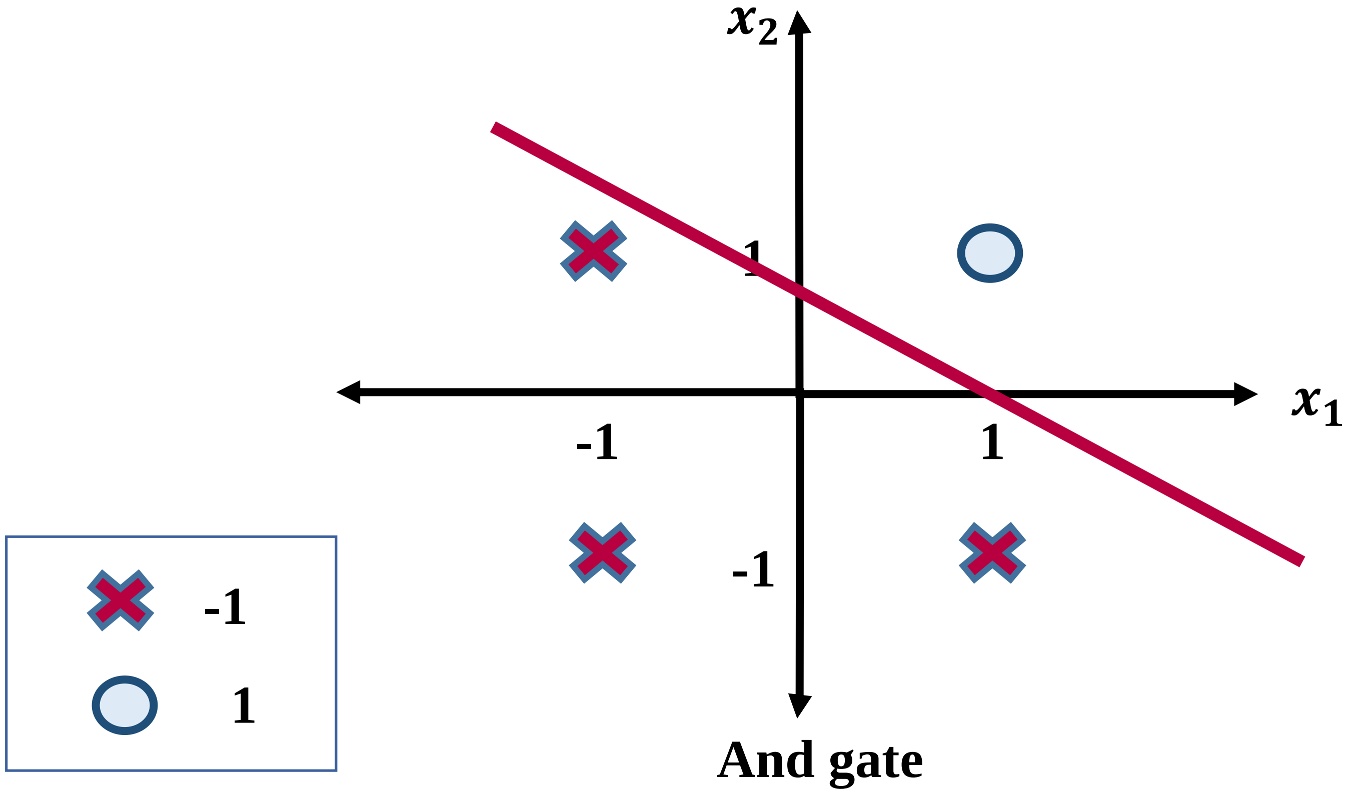
The And gate (for bipolar data)

x_1	x_2	y =x_1 and x_2
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

Hebb net
Example

The response regions for the AND gate after the third input pattern

- **We will have the following separation line:** $x_1 + x_2 = 1$



The And gate (for bipolar data)

x_1	x_2	t $= x_1$ and x_2
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

Hebb net
Example

- ❑ **Solution: Training phase**
  - **Train the neuron by using the fourth pattern (-1,-1,1), the target here is -1**
  - **Activation:** compute the neuron output $\boldsymbol{o_i} = \sum_{i=0}^{n} \boldsymbol{w_i x_i}$

    $\boldsymbol{net_4} = \sum_{i=0}^{n} \boldsymbol{w_i x_i} = (1\times -1) + (1\times -1) + (-1\times 1) = -3$

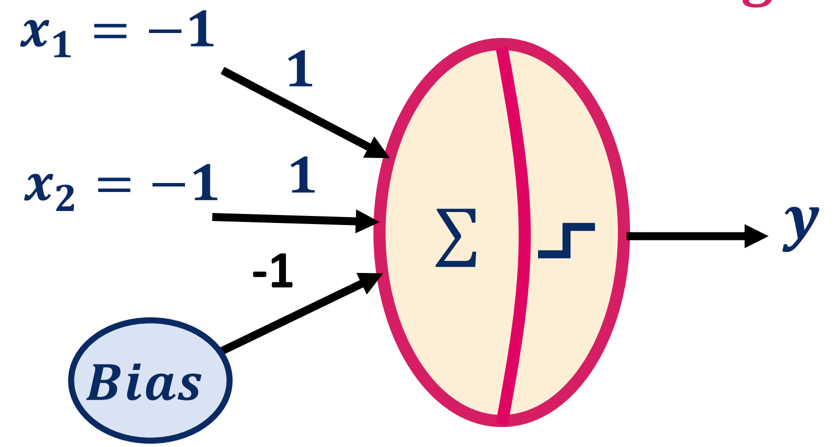
    $o_4 = a(net_4) = -1\ (\because net_4 < 0\ \therefore a(net_4) = -1)$
  - **Learning rules:**
    - Adjust the weights: $\boldsymbol{w_{i,new}} := \boldsymbol{w_{i,old,}} + x_i \cdot t$

      $\boldsymbol{w_{1,new}} := 1 + (-1 \times -1) = 2$

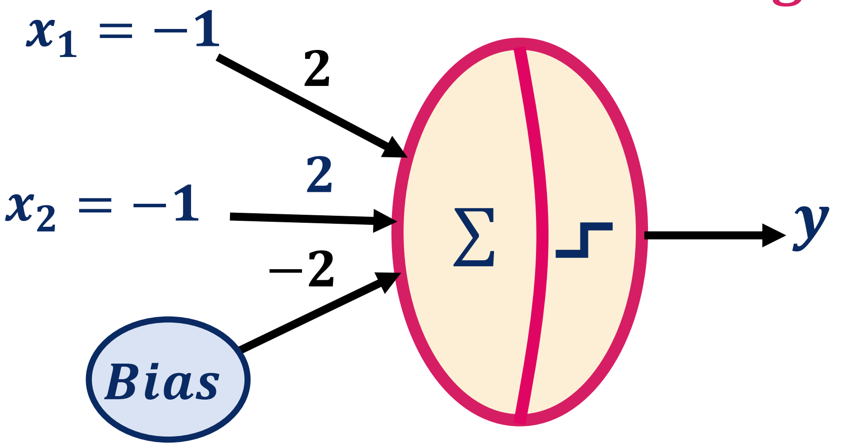
      $\boldsymbol{w_{2,new}} := 1 + (-1 \times -1) = 2$
    - Adjust the weight of bias: $\boldsymbol{b_{new,}} := \boldsymbol{b_{old,}} + t$

      $\boldsymbol{b_{new}} := -1 + (-1) = -2$

The neuron before training



The neuron after training



Hebb net
Example

- **The response regions for the AND gate after the fourth input pattern**

  - $\boldsymbol{The\ decision\ boundary\ equation}$ **is**

$$b + w_1 x_1 + w_2 x_2 = 0$$

  - **The weights after train the first patter are**

$$w_1\text{=}\mathbf{2,}\ w_2 = 2 \ \textbf{and}\ \text{b = -2}$$

  - **We will have the following separation line**

$$\because \boldsymbol{y = -1\ nad\ y < 0}$$

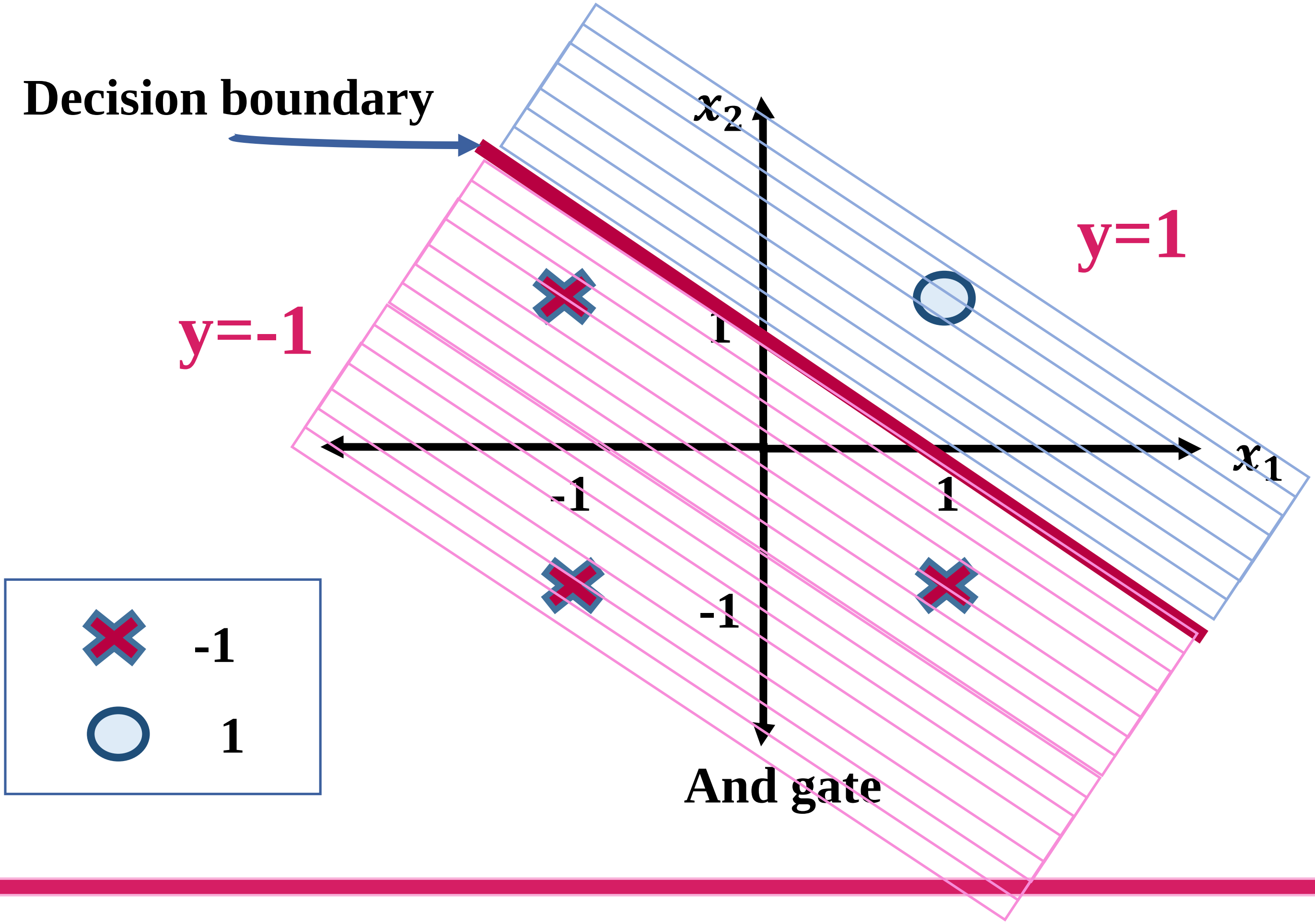
$$-2 + 2x_1 + 2x_2 = 0 \Rightarrow 2x_1 + 2x_2 = 2 \Rightarrow x_1 + x_2 = 1$$

The And gate (for bipolar data)

x_1	x_2	t =x_1 and x_2
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

Hebb net
Example

- The correct response regions for the AND gate



The And gate (for bipolar data)

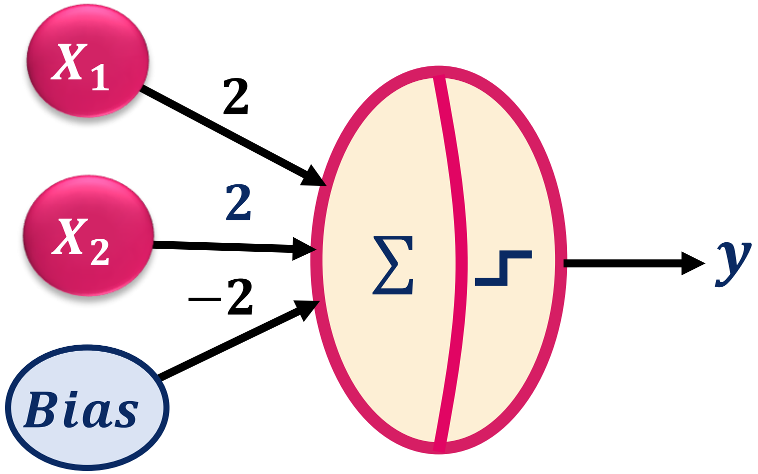
x_1	x_2	t = x_1 and x_2
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

Hebb net
Example

❑ Solution: Training phase

- **Now, the training phase is finished because we input all given patterns.**
- **We get the learned neuron that is abled to predict a new value.**

The final neuron



Hebb net
Example

❑ Solution: Check the final neuron

x_1	x_2	t
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

- **Check the first pattern (1,1)**

$\boldsymbol{net_1}= \sum_{i=1}^{n} \boldsymbol{w_i x_i} + \boldsymbol{b} = (1\times 2) + (1\times 2) + (-2) = 2$, $\boldsymbol{o_1}$= a(2) = **1**

- **Check the second pattern (1,-1)**

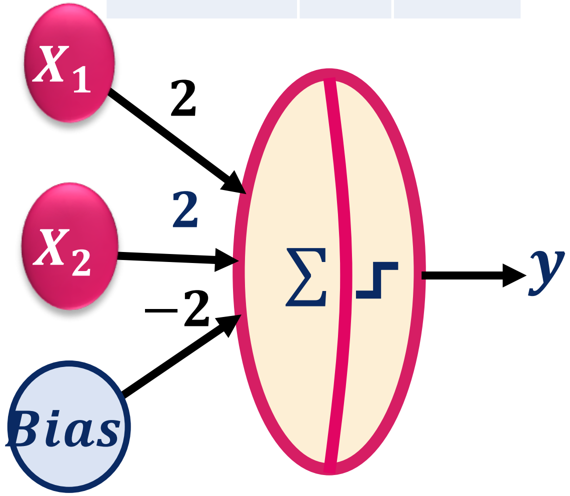
$\boldsymbol{net_2}= \sum_{i=1}^{n} \boldsymbol{w_i x_i} + \boldsymbol{b} = (1\times 2) + (-1\times 2) + (-2) = -2$, $\boldsymbol{o_2}$=a(-2) = **-1**

- **Check the third pattern (-1,1)**

$\boldsymbol{net_3}= \sum_{i=1}^{n} \boldsymbol{w_i x_i} + \boldsymbol{b} = (-1\times 2) + (1\times 2) + (-2) = 2$, $\boldsymbol{o_3}$ =a(-2) = **-1**

- **Check the fourth pattern (-1,-1)**

$\boldsymbol{net_4}= \sum_{i=1}^{n} \boldsymbol{w_i x_i} + \boldsymbol{b} = (-1\times 2) + (-1\times 2) + (-2) = -2$, $\boldsymbol{o_4}$= a(-6) = **-1**



Hebb network
Alternative view: weight matrix

- **How can associate an input vector with a specific output vector in a neural net?**
  - In this case, Hebb's Rule is the same as taking the outer product of the two vectors:

$$\boldsymbol{p_i} = (x_1, x_2, \dots, x_n) \text{ and } \boldsymbol{t_i} = (o_1, o_2, \dots, o_m)$$

$$\boldsymbol{pt_i} = \begin{bmatrix} \boldsymbol{x_1} \\ \vdots \\ \boldsymbol{x_n} \end{bmatrix} [\boldsymbol{o_1}, \dots, \boldsymbol{o_m}] = \boxed{\begin{bmatrix} \boldsymbol{x_1 o_1} & \cdots & \boldsymbol{x_1 o_m} \\ \vdots & \vdots & \vdots \\ \boldsymbol{x_n o_1} & \cdots & \boldsymbol{x_n o_m} \end{bmatrix}}$$

Weight matrix

Hebb network
Weight matrix

❑ Weight matrix

- Is used to store more than one association in a neural net using **Hebb's Rule**
- That occurs by adding the individual weight matrices
- This method works only if the input vectors for each association are **orthogonal** (**uncorrelated**). That means, if their dot product is 0

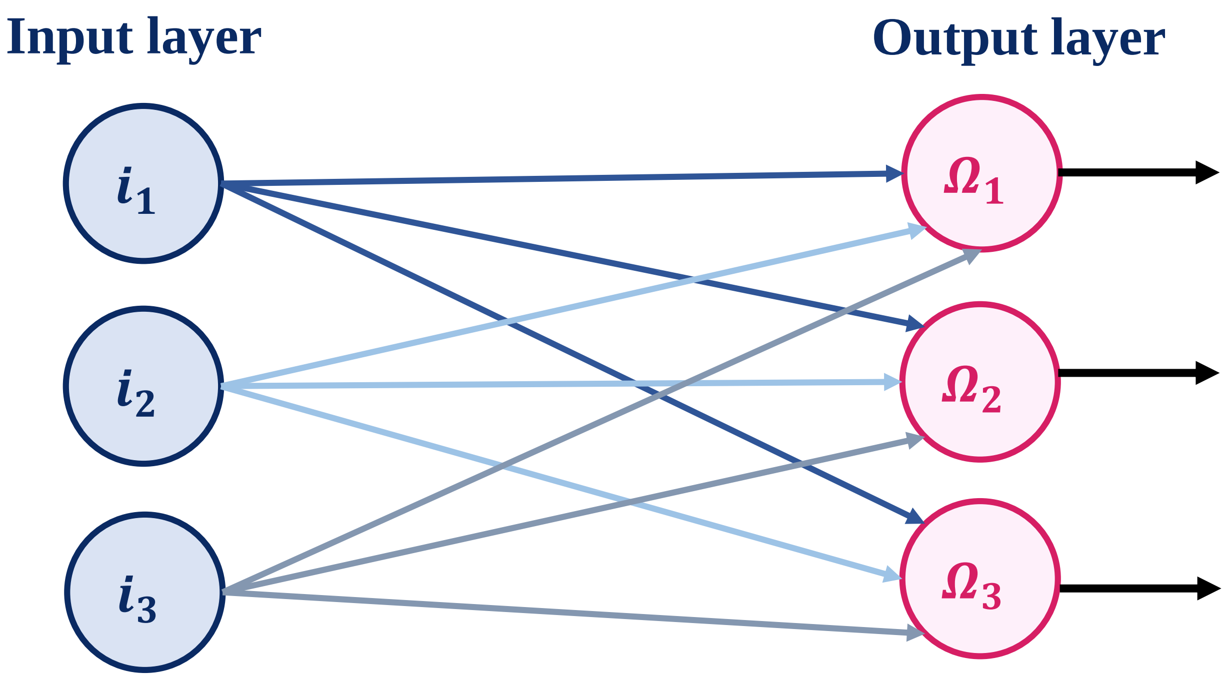
$$\boldsymbol{p_i} = (x_1, x_2, \dots, x_n) \text{ and } \boldsymbol{t_i} = (o_1, o_2, \dots, o_m)$$

$$\boldsymbol{pt_i} = [\boldsymbol{p_1}, \dots, \boldsymbol{p_n}]\begin{bmatrix} \boldsymbol{t_1} \\ \vdots \\ \boldsymbol{t_m} \end{bmatrix} = \mathbf{0}$$

Hebb network
Weight matrix

❑ Weight matrix

- There are **n** input units and **m** output units with each input connected to each output unit.



Hebb network
Weight matrix

❑ Example:

- By using Hebb's algorithm and the following training dataset construct an Hebb artificial neurol network that associates the following training samples.
- The training samples and the activation function((let $\boldsymbol{\theta}$ =0).) are

$$\boldsymbol{p_1} = (1,-1,-1,-1) \text{ and } \boldsymbol{t_1} = (1,-1,-1)$$
$$\boldsymbol{p_2} = (-1,1,-1,-1) \text{ and } \boldsymbol{t_2} = (1,-1,1)$$
$$\boldsymbol{p_3} = (-1,-1,1,-1) \text{ and } \boldsymbol{t_3} = (-1,1,-1)$$
$$\boldsymbol{p_4} = (-1,-1,-1,1) \text{ and } \boldsymbol{t_4} = (-1,1,1)$$

$$a(o_i) = \begin{cases} 1, & if\ o_i > \theta \\ 0, & if\ o_i = \theta \\ -1, & if\ o_i < \theta \end{cases}$$

Hebb network
Weight matrix

❑ Solution:

- In this training set we have 4 input and 3 output neurons.
- That means, we are going to use the weight matrix by find the four outer products and adding them.

Hebb network
Weight matrix

Solution:

1. Find the four outer products

First pair:

p_1 = (1,-1,-1,-1) and t_1 = (1,-1,-1)

$$pt_1 = \begin{bmatrix} 1 \\ -1 \\ -1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \end{bmatrix}$$

Second pair:

p_2 = (-1,1,-1,-1) and t_2 = (1,-1,1)

$$pt_2 = \begin{bmatrix} -1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 1 & -1 \\ 1 & -1 & 1 \\ -1 & 1 & -1 \\ -1 & 1 & -1 \end{bmatrix}$$

Third pair:

p_3 = (-1,-1,1,-1) and t_3 = (-1,1,-1)

$$pt_3 = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} \begin{bmatrix} -1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 \\ 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix}$$

Fourth pair:

p_4 = (-1,-1,-1,1) and t_4 = (-1,1,1)

$$pt_4 = \begin{bmatrix} -1 \\ -1 \\ -1 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & -1 \\ 1 & -1 & 1 \\ 1 & -1 & -1 \\ -1 & 1 & 1 \end{bmatrix}$$

Hebb network
Weight matrix

❑ Solution:

2. Find the weight matrix by Add all four individual weight matrices

$$\begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \end{bmatrix} + \begin{bmatrix} -1 & 1 & -1 \\ 1 & -1 & 1 \\ -1 & 1 & -1 \\ -1 & 1 & -1 \end{bmatrix} + \begin{bmatrix} 1 & -1 & 1 \\ 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & -1 & -1 \\ 1 & -1 & 1 \\ 1 & -1 & -1 \\ -1 & 1 & 1 \end{bmatrix} =$$

$$\begin{array}{c} \begin{bmatrix} 2 & -2 & -2 \\ 2 & -2 & 2 \\ -2 & 2 & -2 \\ -2 & 2 & 2 \end{bmatrix} \\ \begin{array}{ccc} y_1 & y_2 & y_3 \end{array} \end{array}$$

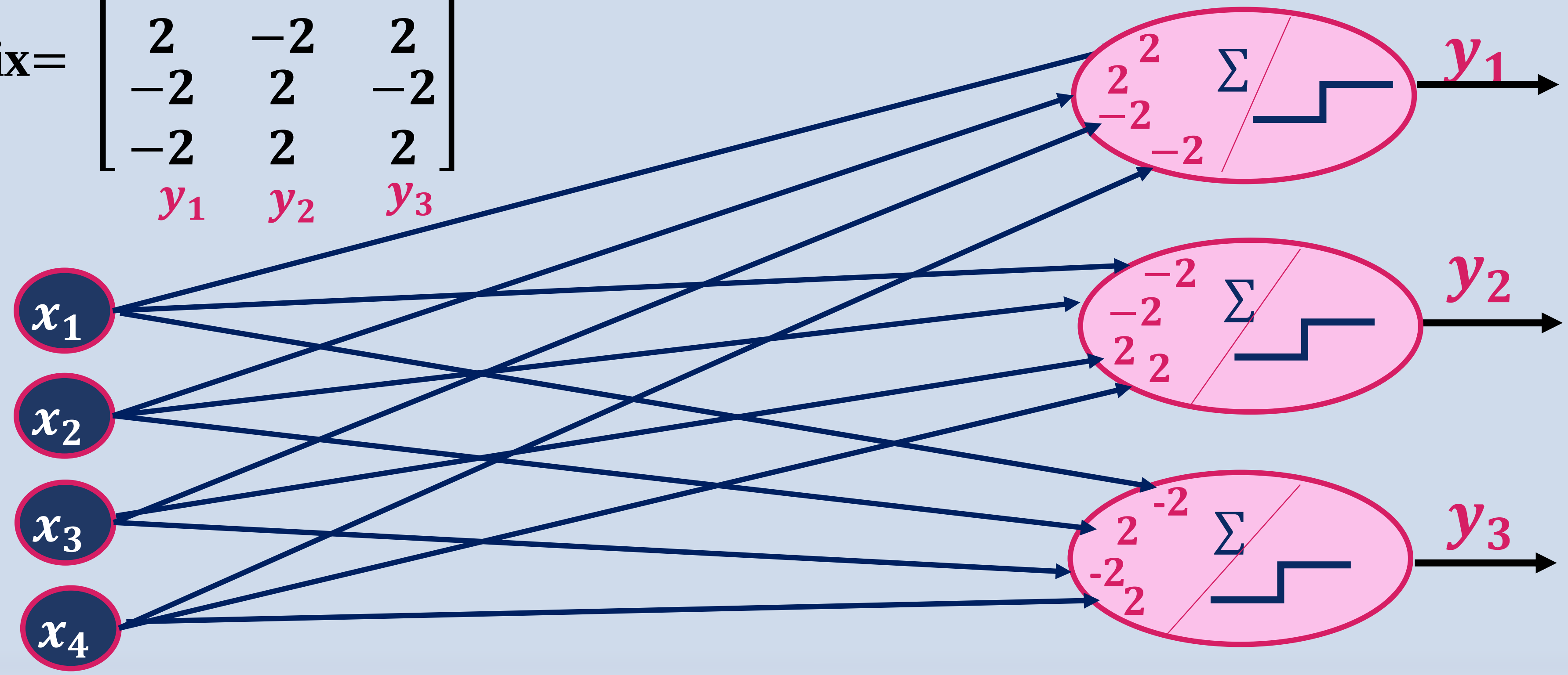
- Each column in final weight matrix defines the weights for an output neuron.

Hebb network
Weight matrix

❑ Solution:

3. Construct the neurol architecture

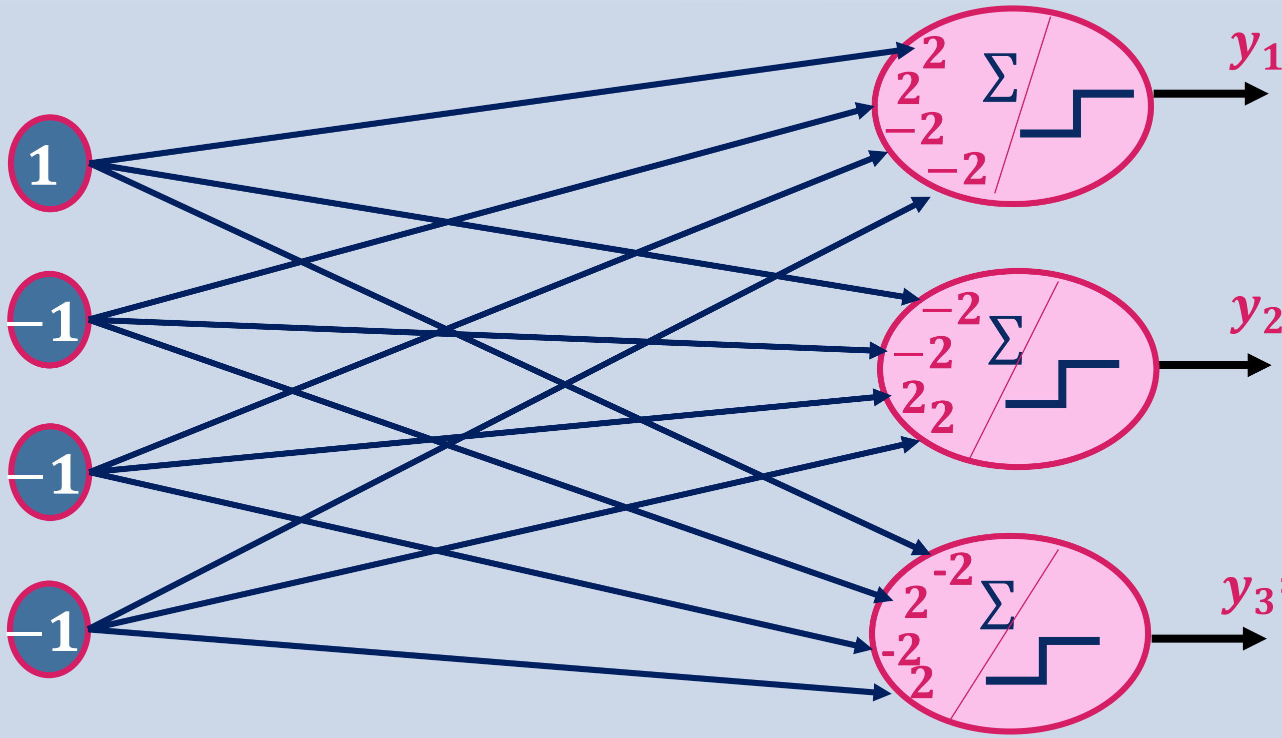
Final weight matrix=
$$\begin{array}{c} \begin{bmatrix} 2 & -2 & -2 \\ 2 & -2 & 2 \\ -2 & 2 & -2 \\ -2 & 2 & 2 \end{bmatrix} \\ \begin{array}{ccc} y_1 & y_2 & y_3 \end{array} \end{array}$$



Hebb network
Weight matrix

❑ Solution:

4. **Train the neuron by using the following input p_1 = (1,-1,-1,-1) and t_1 = (1,-1,-1)**



y_1= (2×1)+(2×-1)+(-2×-1)+(-2×-1) =
2-2+2+2= 4 and a(4) = **1**

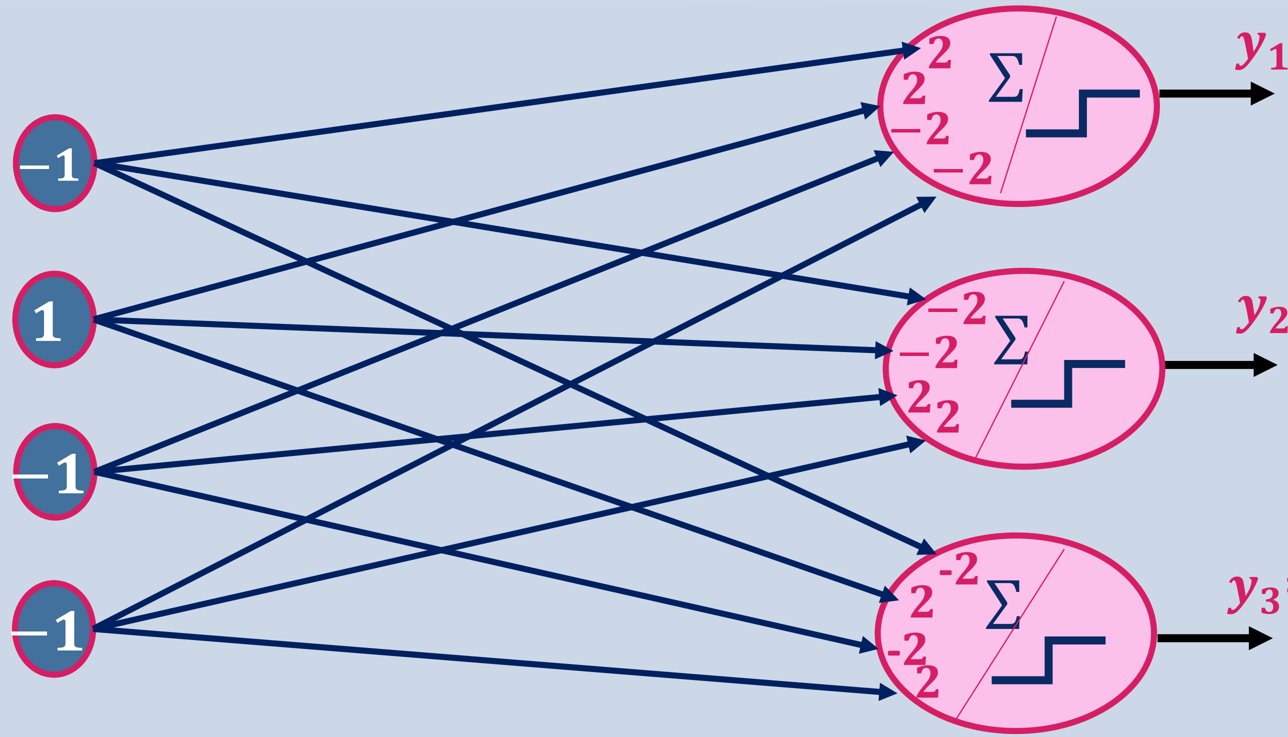
y_2= (-2×1)+(-2×-1)+(2×-1)+(2×-1) =
-2+2-2-2= -4 and a(-4) = **-1**

y_3= (-2×1)+(2×-1)+(-2×-1)+(2×-1) =
-2-2+2-2= -4 and a(-4) = **-1**

Hebb network
Weight matrix

❑ Solution:

5. **Train the neuron by using the following input p_2 = (-1,1,-1,-1) and t_2 = (1,-1,1)**



$y_1 = (2 \times -1)+(2 \times 1)+(-2 \times -1)+(-2 \times -1) =$
-2+2+2+2= 4 and a(4) = 1

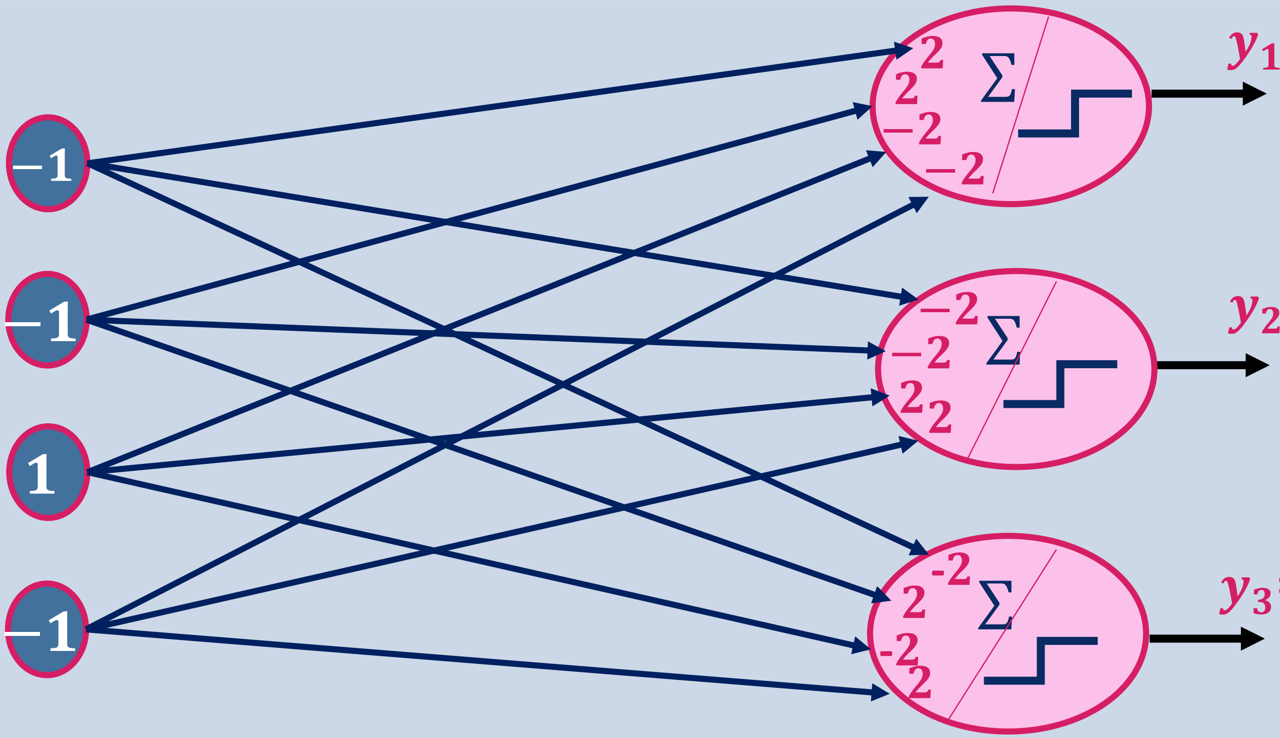
$y_2 = (-2 \times -1)+(-2 \times 1)+(2 \times -1)+(2 \times -1) =$
2-2-2-2= -4 and a(-4) = -1

$y_3 = (-2 \times -1)+(2 \times 1)+(-2 \times -1)+(2 \times -1) =$
2+2+2-2= 4 and a(4) = 1

Hebb network
Weight matrix

❑ Solution:

6. **Train the neuron by using the following input p_3 = (-1,-1,1,-1) and t_3 = (-1,1,-1)**



y_1= $(2\times-1)+(2\times-1)+(-2\times1)+(-2\times-1)$ =
-2-2-2+2= -4 and a(-4) = -1

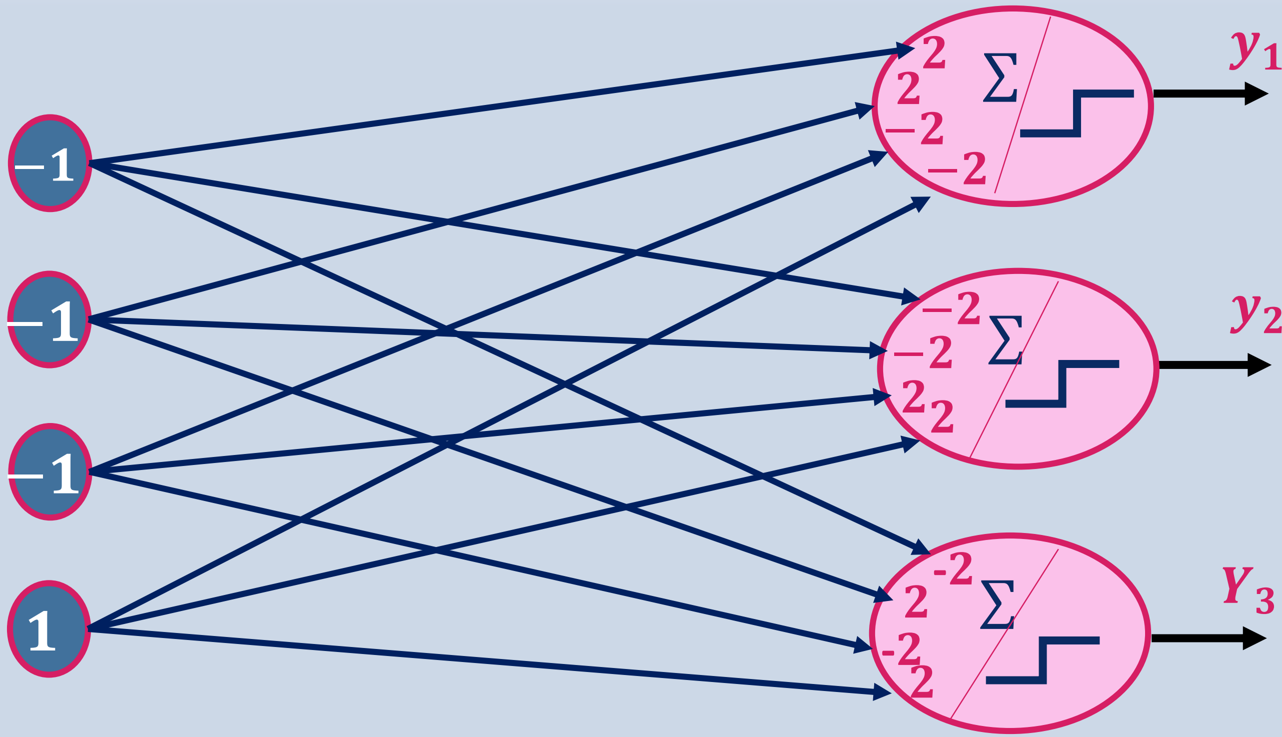
y_2= $(-2\times-1)+(-2\times-1)+(2\times1)+(2\times-1)$ =
2+2+2-2= 4and a(4) = 1

y_3= $(-2\times-1)+(2\times-1)+(-2\times1)+(2\times-1)$ =
2-2-2-2= -4 and a(-4) = -1

Hebb network
Weight matrix

❑ Solution:

7. **Train the neuron by using the following input p_4 = (-1,-1,-1,1) and t_4 = (-1,1,1)**



y_1= (2×-1)+(2×-1)+(-2×-1)+(-2×1) =
-2-2+2-2= -4 and a(-4) = -1

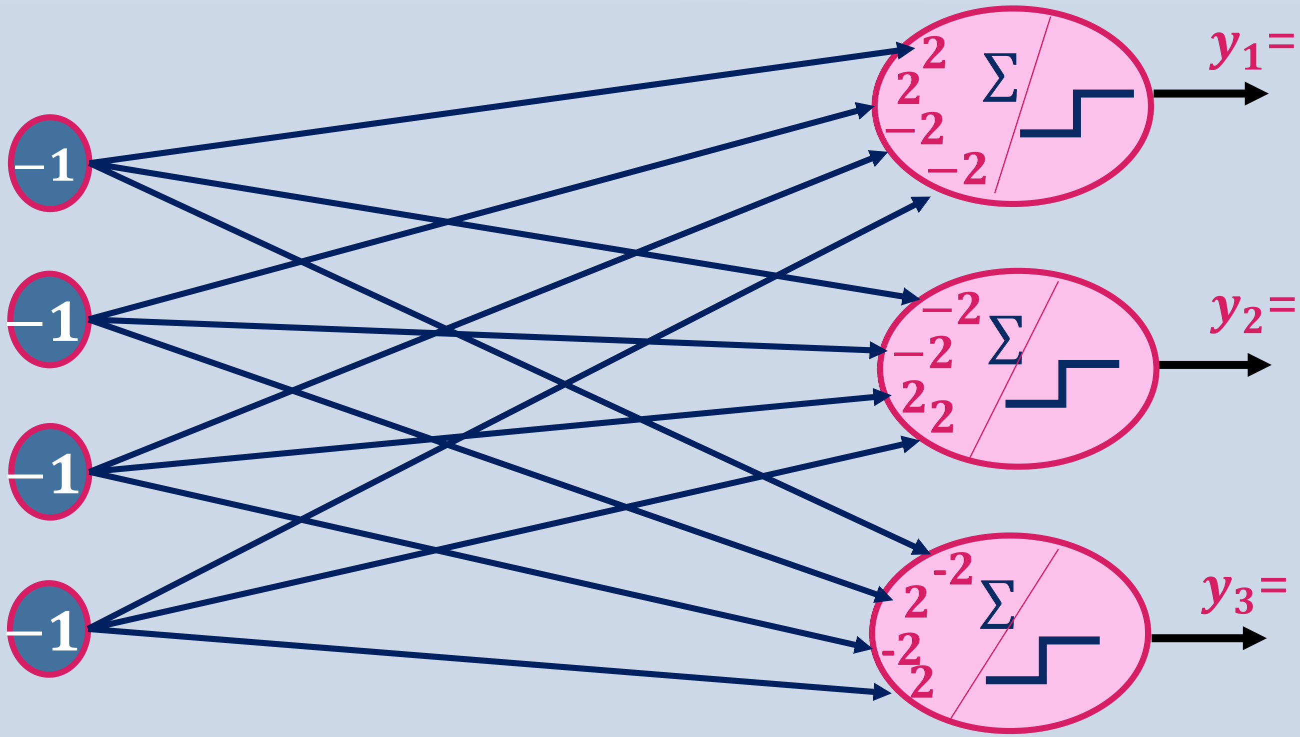
y_2= (-2×-1)+(-2×-1)+(2×-1)+(2×1) =
2+2-2+2= 4and a(4) = 1

Y_3= (-2×-1)+(2×-1)+(-2×-1)+(2×1) =
2-2+2+2= 4 and a(4) = 1

Hebb network
Weight matrix

❑ Solution:

8. **Test the neuron by using unseen input** p **= (-1,-1,-1,-1) and** t **= (0,0,0)**



$y_1 = (2 \times -1)+(2 \times -1)+(-2 \times -1)+(-2 \times -1) =$
-2-2+2+2= 0 and a(0) = **0**

$y_2 = (-2 \times -1)+(-2 \times -1)+(2 \times -1)+(2 \times -1) =$
2+2-2-2= 0 and a(0) = **0**

$y_3 = (-2 \times -1)+(2 \times -1)+(-2 \times -1)+(2 \times -1) =$
2-2+2-2= 0 and a(0) = **0**

Assignments

❑ Assignment (4.1)

- Train a Hebb network to classify the following training set:
- Assume that, the bias is 1 and initial weights (0,1,-1).

x_1	x_2	target
4	5	T
6	1	T
4	1	F
1	2	F

References

- Kriesel, David. "A Brief Introduction to Neural Networks. 2007." URL http://www. dkriesel. com (2007).
- da Fontoura Costa, Luciano, and Gonzalo Travieso. "Fundamentals of neural networks: By Laurene Fausett. Prentice-Hall, 1994, pp. 461, ISBN 0-13-334186-0." (1996): 205-207.

Any Questions!?

Thank you