Artificial Neural Networks (ANN)

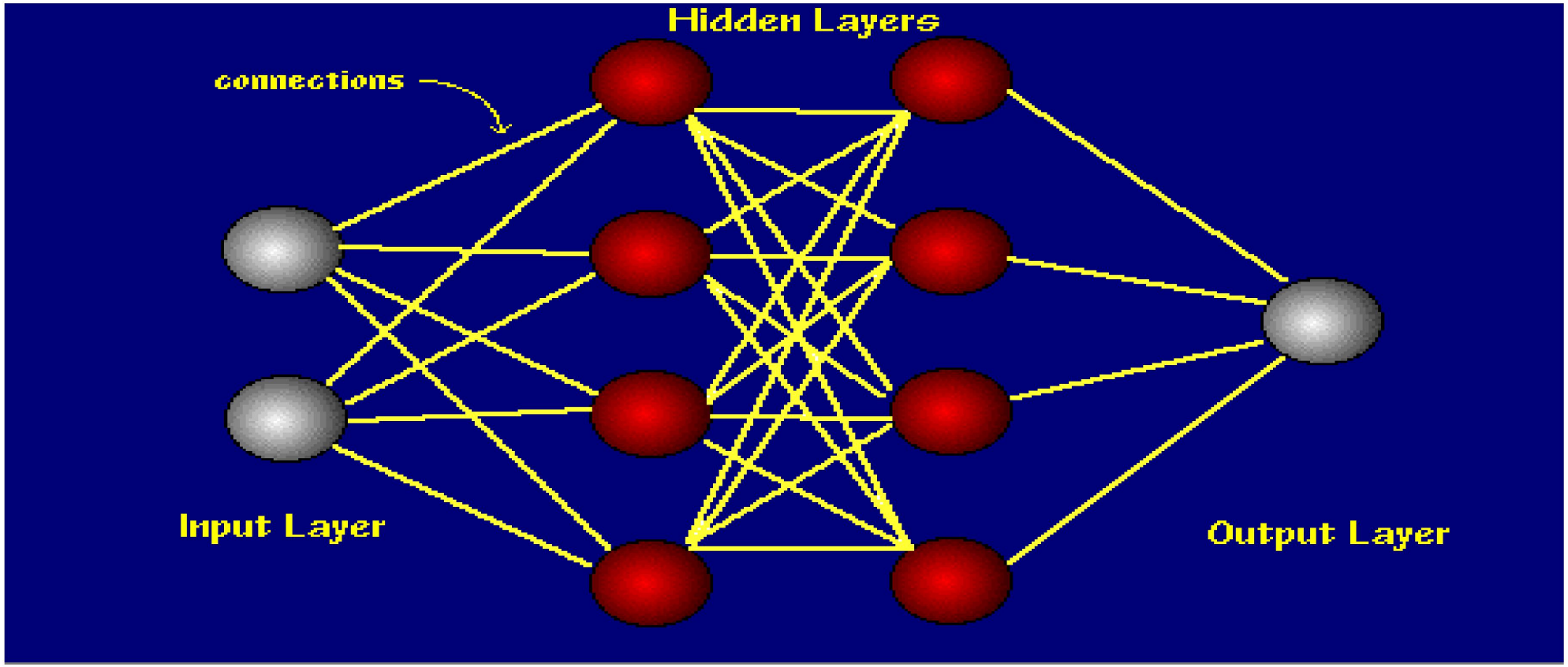
Dr. Nasima Begum
Assistant Professor
Dept. of CSE, UAP

Neural Networks

Outline:

1. Introduction to Neural Networks
2. **Perceptron**
3. Introduction to Backpropagation NN
4. Associative Memory: Hopfield Nets
5. Summary

Artificial Neural Networks

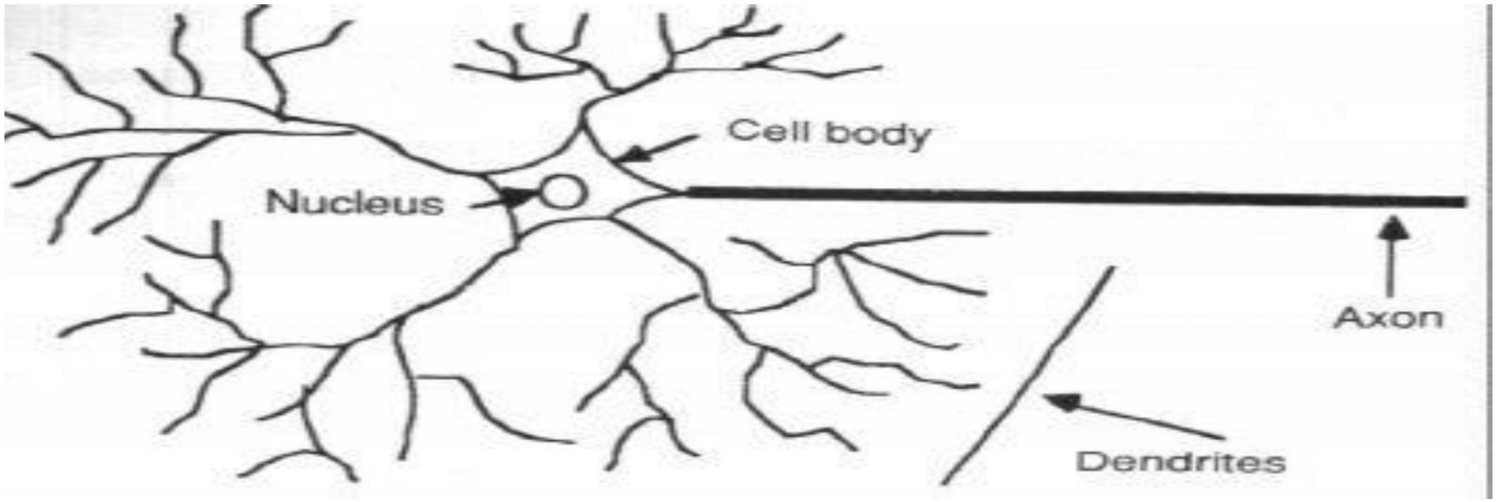


Machine Intelligence

- Machine learning involves adaptive mechanisms that enable computers to learn from experience, learn by example and learn by analogy.

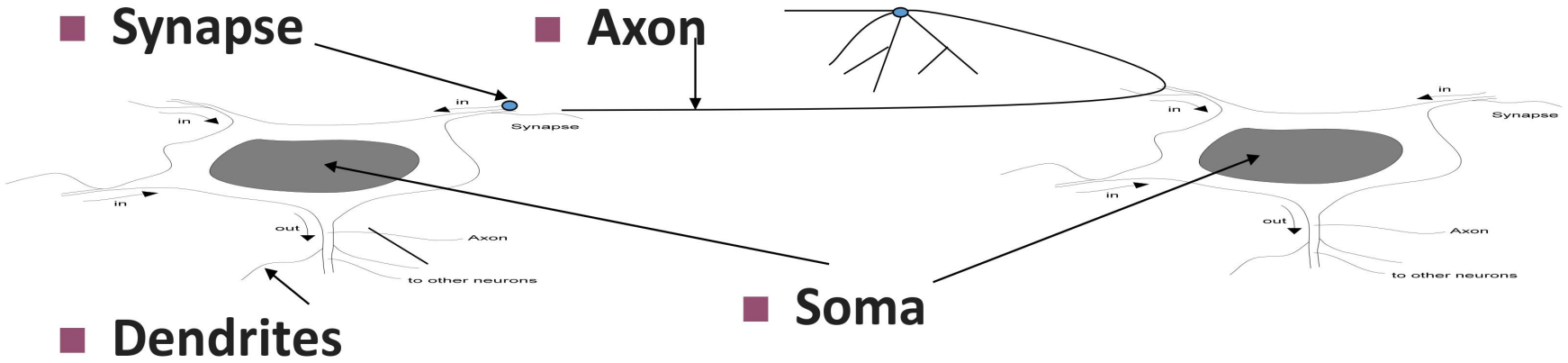
- To develop methods and systems for solving problems through modeling the human way of thinking (the way human brain works), e.g. image recognition, language and speech processing; planning, prediction, etc., thus enhancing the computer information systems.

What is a Neural Network?



- A neural network can be defined as a model of reasoning based on the human brain. **The brain** consists of a densely interconnected set of nerve cells, or basic information-processing units, **called neurons**.

Biological Neural Network



- A neuron consists of a cell body, **soma**, a number of fibers called **dendrites**, and a single long fiber called the **axon**. While dendrites branch into a network around the soma, the axon stretches out to the dendrites and somas of other neurons. The connections between neurons are realized in the **synapses**. Signals are propagated from one neuron to another by complex electrochemical reactions.

Analogy between Biological and Artificial NNs

Biological NN	**Artificial NN**
Soma	Neuron
Dendrite	Input
Axon	Output
Synapse	Weight

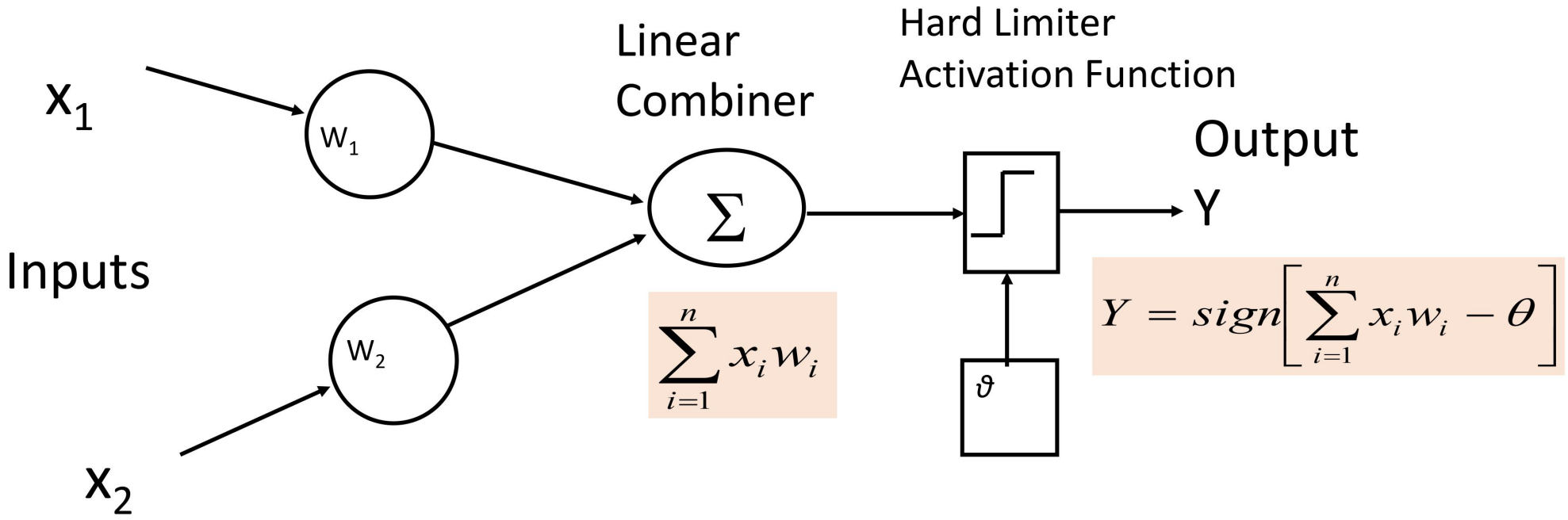
Artificial Neural Networks (ANN)

- A mathematical model to solve engineering problems
  - Group of highly connected neurons to realize compositions of non linear functions
- **Tasks:**
  - Classification
  - Recognition
  - Prediction
  - Discrimination
  - Estimation
- **Types** of networks:
  - Feed forward Neural Networks
  - Recurrent Neural Networks

- An ANN system is **characterized** by:
- its **ability to learn**; its **dynamic capability**; its **interconnectivity.**

Single Layer Perceptron

Perceptron is the simplest form of a neural network. It consists of a single neuron with adjustable synaptic weights and a hard limiter activation function.



Activation Functions

- Use different functions to obtain different models.

- 3 most common choices :

1) Step function
2) Sign function
3) Sigmoid function

Other:

4) Rectified Linear Units (ReLU): $f(x)=\max(0,x)$; if $x>0 \rightarrow f(x)=x$, $x<=0 \rightarrow f(x)=0$
5) Softmax

Recent:

6) **Leaky ReLU**: an attempt to fix the "dying **ReLU**" problem. Instead of the **function** being zero when x < 0, a **leaky ReLU** has a small negative slope (of 0.01, or so).
$f(x)=\max(\alpha x, x) \rightarrow f(x)=\max(0.01x, x) \rightarrow f(x)=\max(-0.001, -1) = -0.001$

- An output of 1 represents firing of a neuron down the axon.

Activation Functions

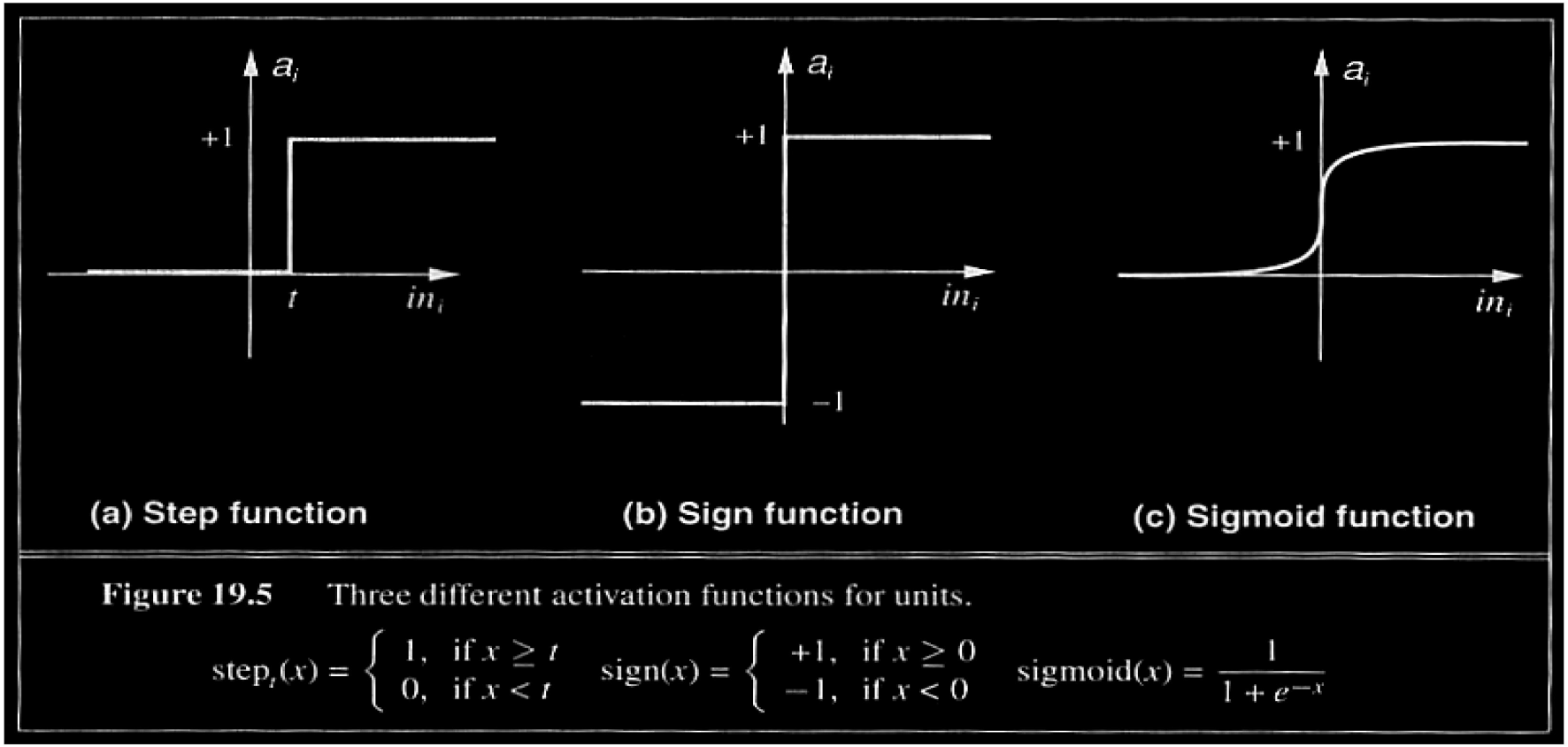
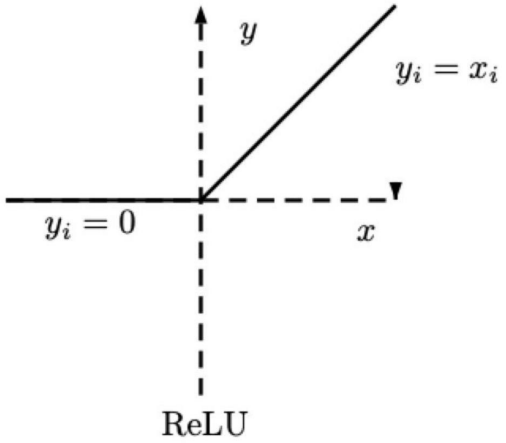
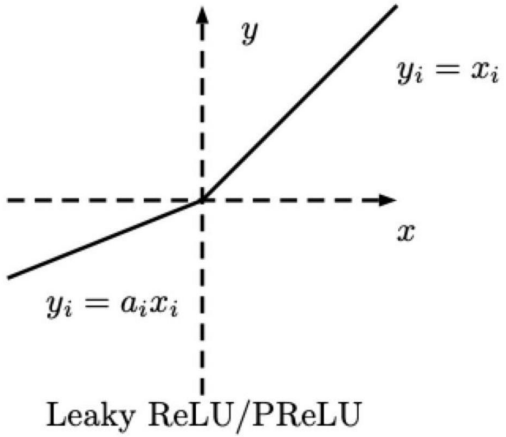


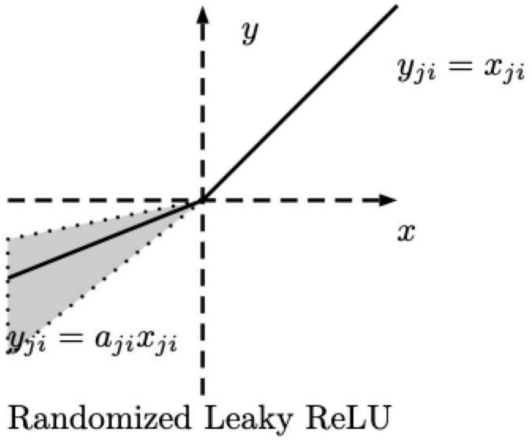
Figure 19.5 Three different activation functions for units.

$$\text{step}_t(x) = \begin{cases} 1, & \text{if } x \geq t \\ 0, & \text{if } x < t \end{cases} \qquad \text{sign}(x) = \begin{cases} +1, & \text{if } x \geq 0 \\ -1, & \text{if } x < 0 \end{cases} \qquad \text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

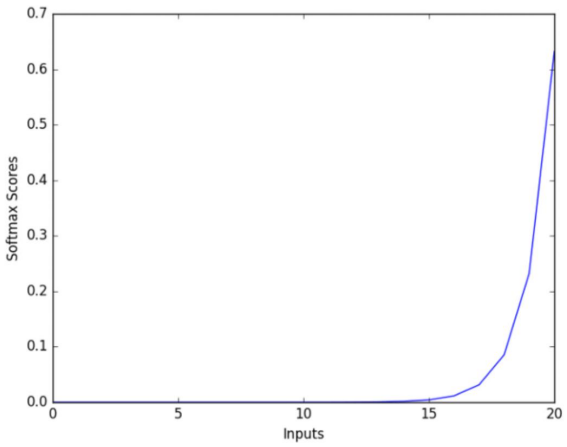
Activation Functions







$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

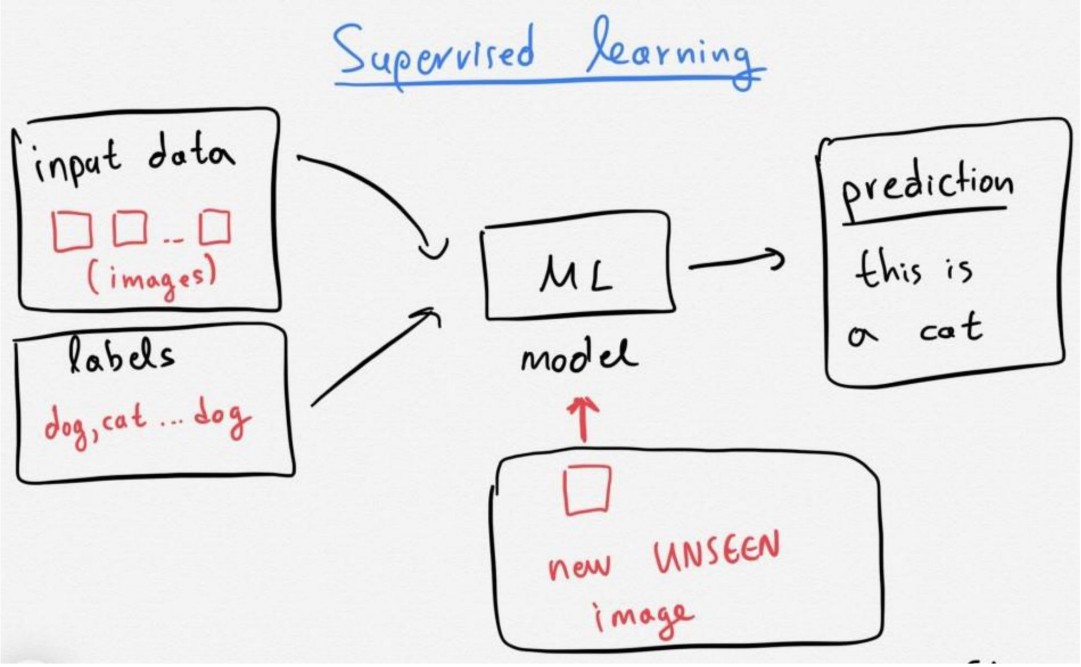


Learning

- The procedure that consists in estimating the parameters of neurons so that the whole network can perform a specific task.
- An artificial neural network's **learning rule** or **learning process** is a method, mathematical logic or algorithm which improves the network's performance and/or training time.
- The learning rule is one of the factors which decides how fast or how accurately the artificial network can be developed.
- Types of learning:
  - The supervised learning
  - The unsupervised learning
  - Reinforcement learning
- The Learning process (supervised)
  - Present the network a number of inputs and their corresponding outputs
  - See how closely the actual outputs match the desired ones
  - Modify the parameters to better approximate the desired outputs.

Supervised Learning

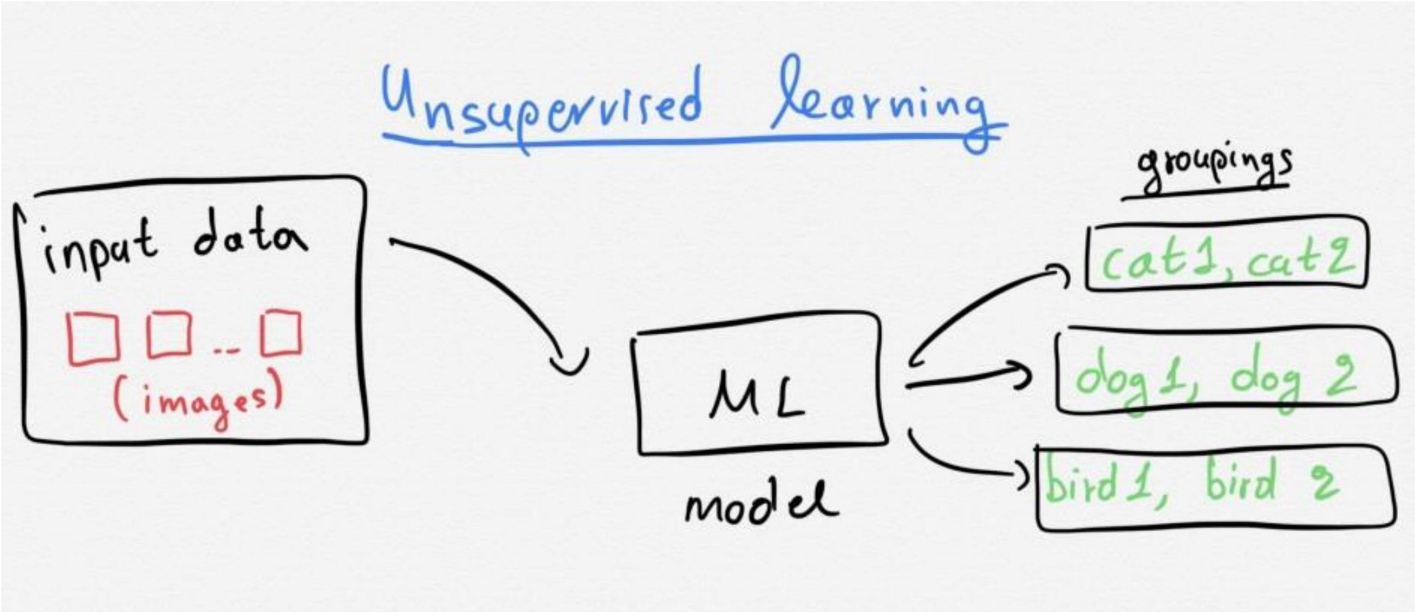
- All the observations in the dataset are labeled and the algorithms learn to predict the output from the input data.
  - Present the network a number of inputs and their corresponding outputs
  - See how closely the actual outputs match the desired ones
  - Modify the parameters to better approximate the desired outputs.
  - **E.x: Perceptron**



Unsupervised Learning

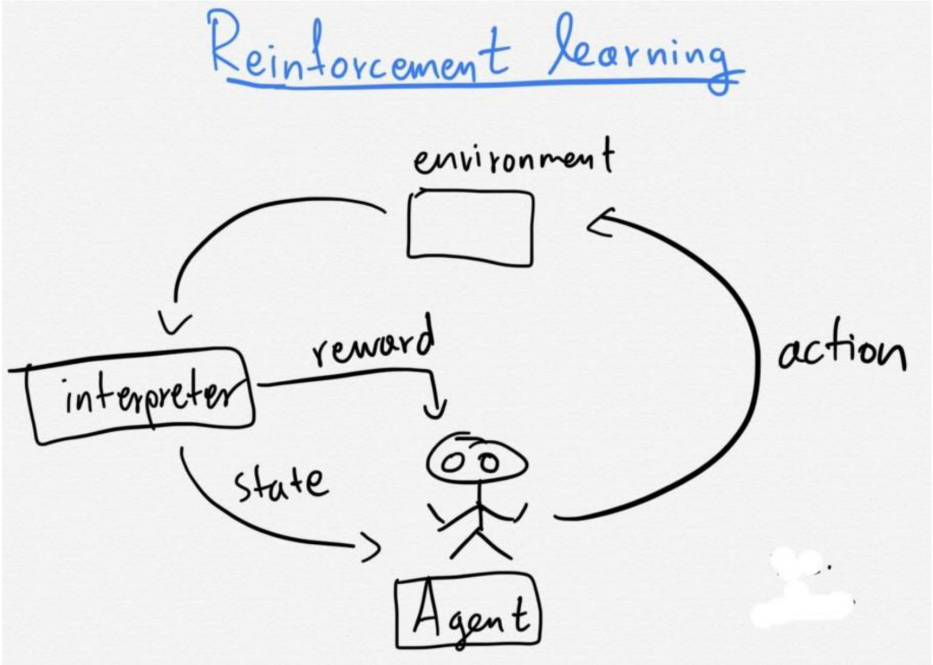
All the observations in the dataset are unlabeled and the algorithms learn to inherent structure from the input data.

- Data clustering
- No need of a teacher
  - The network finds itself the correlations between the data
  - Examples of such networks :
    - Kohonen self-organized feature maps (SOFM)



Reinforcement Learning

- Consists of algorithms that use the estimated errors as rewards or penalties.
- If the error is **big**, then the **penalty** is **high** and the **reward low**.
- If the error is **small**, then the **penalty** is **low** and the **reward high**.
- Trial error search and delayed reward are the most relevant characteristics of reinforcement learning.



Rosenblatt's Perceptron

Step 1: **Initialization**

Set initial weights w_1, w_2, ..w_n and threshold ϑ to random numbers [-0.5,0.5].

Step 2: **Activation**

Activate the perceptron by applying inputs $x_1(p)$, $x_2(p)$, $x_3(p)$, ……$x_n(p)$ and desired output $Y_d(p)$. Calculate the actual output at iteration $p = 1$:

$$Y_a(p) = step\left[\sum_{i=1}^{n} x_i(p) w_i(p) - \theta\right]$$

Where ***n*** is the number of the **perceptron inputs**, and *step* is a step activation function.

Step 3: **Weight Training**

Update the weights of the perceptron

$$w_i(p+1) = w_i(p) + \alpha \times x_i(p) \times e(p),$$

where **e(p) = Error at iteration (p =1) = Y_d - Y_a,** $\alpha = learning\ rate$

Step 4: **Iteration**

Increase iteration *p* by one, go back to Step 2 and repeat the process until convergence.

Example of Perceptron Learning

- Threshold: $\theta = 0.2$, Learning rate: $\alpha = 0.1$

Epoch	Inputs X_1	Inputs X_2	Desired output Y_d	Initial Weights W_1	Initial Weights W_2	Actual Output Y_a	Error (e)	Final Weights W_1	Final Weights W_2
1	0	0	0	0.3	-0.1	0	0	0.3	-0.1
	0	1	0						
	1	0	0						
	1	1	1						
2									
3									
4									

Rosenblatt's Perceptrons

Consider the following set of training vectors x1, x2, and x3, which are to be used in training a Rosenblatt's perceptron together with the weights etc. Show how the learning proceeds:

$$x_1 = \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix}; x_2 = \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix}; x_3 = \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix}$$

with the desired responses and weights initialized as:

$$\begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}; w = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix}; \alpha = 0.1$$

Rosenblatt's Perceptron

Learning cycles for the perceptron above. Note the initial weights are selected at random:

$$w^{1t} = [1 \quad -1 \quad 0 \quad 0.5]$$

Step 1. Input is x1 and the desired output is d1:

$$net^1 = w^{1t} x_1 = [1 \ -1 \ 0 \ 0.5] \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = 2.5$$

$$y = sign(net^1) = 1$$

- **Correction** in this step **is necessary** since d1= -1 and the input of the perceptron in response to x1 is 2.5; the error signal is: e ≈ -1 - 1 = -2

Rosenblatt's Perceptron

Step 1 (contd). The updated weight vector is

$$w^2 = w^1 + 0.1(-1 \quad -1) * x_1$$

$$w^2 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0.5 \end{bmatrix} + 0.1(-1 \quad -1) * \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix}$$

$$w^{2t} = [0.8 \quad -0.6 \quad 0 \quad 0.7]$$

Rosenblatt's Perceptron

Step 2. The updated weight vector w2

$$w^{2t} = [0.8 \;\; -0.6 \;\; 0 \;\; 0.7]$$

$$net^2 = w^{2t} \; x_2 = [0.8 \;\; -0.6 \;\; 0 \;\; 0.7] \begin{bmatrix} 0 \\ 1.5 \\ -0.5 \\ -1 \end{bmatrix} = -1.6$$

- **Correction** in this step **is not necessary** since d2 = -1 and the output of the perceptron in response to x2 is sign(-1.6) = -1 ; the error signal is: e ≈ -1 - (-1) = 0.

Rosenblatt's Perceptron

Step 3. Input is x3 and the desired output is d3:

$$w^{3t} = w^{2t} = [0.8 \quad -0.6 \quad 0 \quad 0.7]$$

$$net^3 = w^{3t} \; x_3 = [-1 \quad 1 \quad 0.5 \quad -1] \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix} = -2.1$$

- Correction in this step is necessary since d3 = +1 and the output of the perceptron in response to x3 is sign(-2.1) = -1 ; the error signal is: e ≈ -1 - (-1) = 2.

Rosenblatt's Perceptron

Step 3 (contd). The updated weight vector is

$$w^{4t} = w^{3t} + 0.1 * (1 + 1) * x_3$$

$$w^4 = \begin{bmatrix} 0.8 \\ -0.6 \\ 0 \\ 0.7 \end{bmatrix} + 0.2 * \begin{bmatrix} -1 \\ 1 \\ 0.5 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.6 \\ -0.4 \\ 0.1 \\ 0.5 \end{bmatrix}$$

$$w^{4t} = [0.6 \ -0.4 \ 0.1 \ 0.5]$$

Rosenblatt's Perceptron

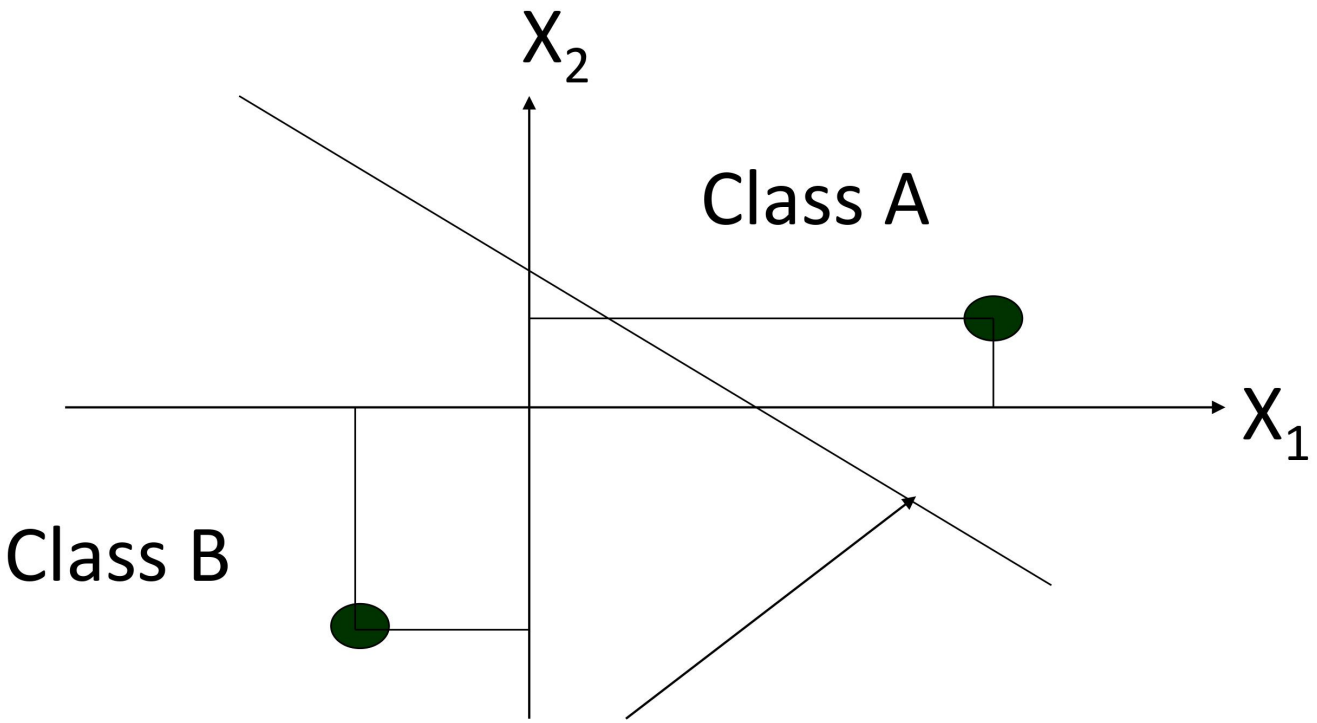
Step 4. Input is x1 and the desired output is d1:

$$w^{4t} = [0.6 \;\; -0.4 \;\; 0.1 \;\; 0.5]$$

$$net^4 = w^{4t} * x_1 = [0.6 \;\; -0.4 \;\; 0.1 \;\; 0.5] * \begin{bmatrix} 1 \\ -2 \\ 0 \\ -1 \end{bmatrix} = 0.9$$

- Correction in this step is necessary since d1 = -1 and the output of the perceptron in response to x3 is sign(0.9) = +1 ; the error signal is: e ≈ -1 - (1) = -2; recall that net1 was 2.5 and e was -2.

Pattern Classification

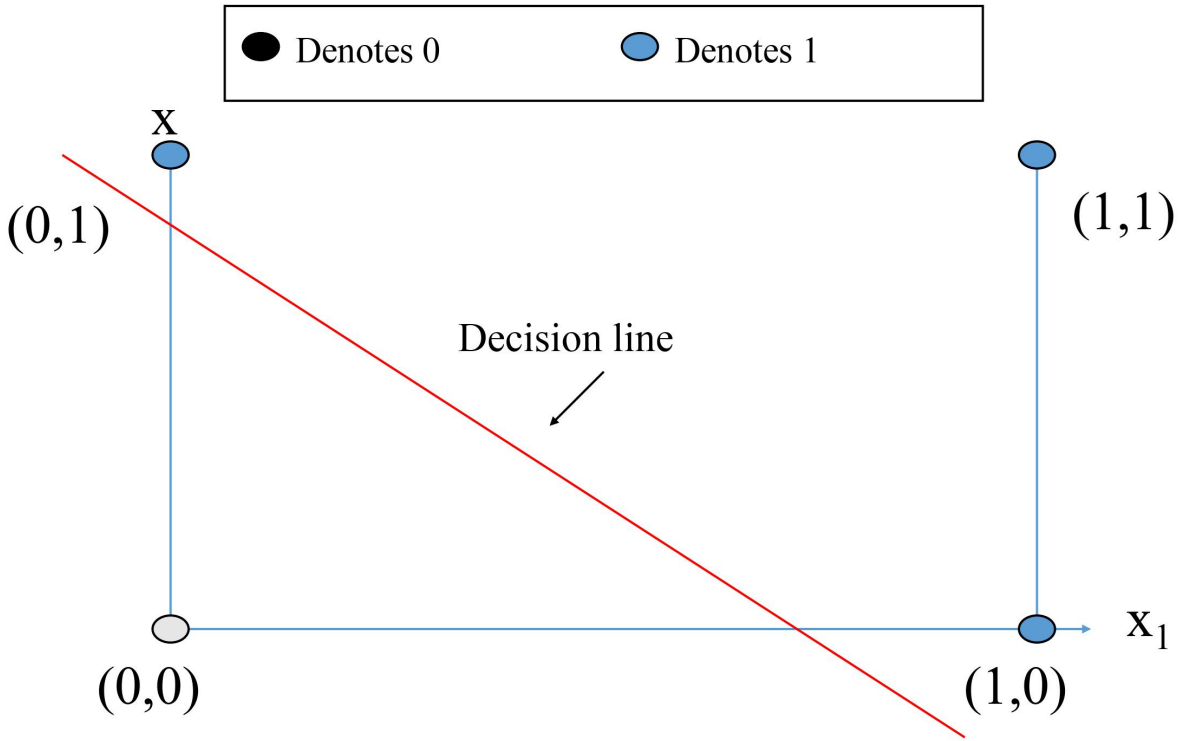


$$x_1 w_1 + x_2 w_2 - \theta = 0$$

Rosenblatt's Perceptron

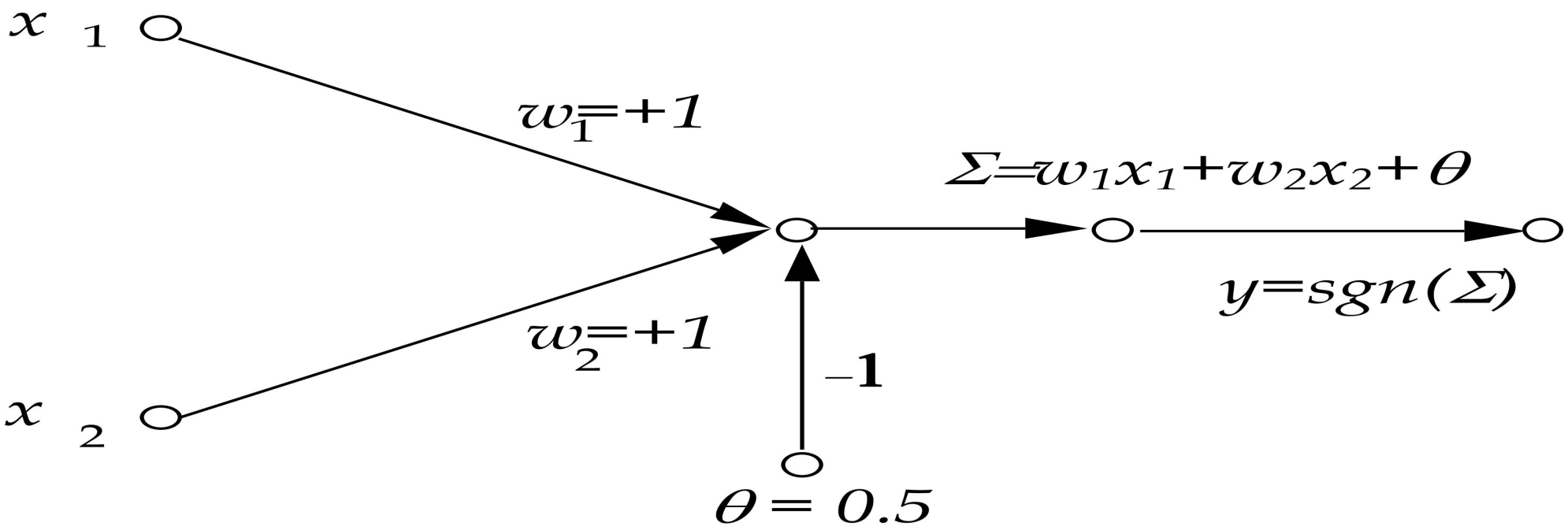
Definition of OR

Input		Output
X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	1



Rosenblatt's Perceptron ...

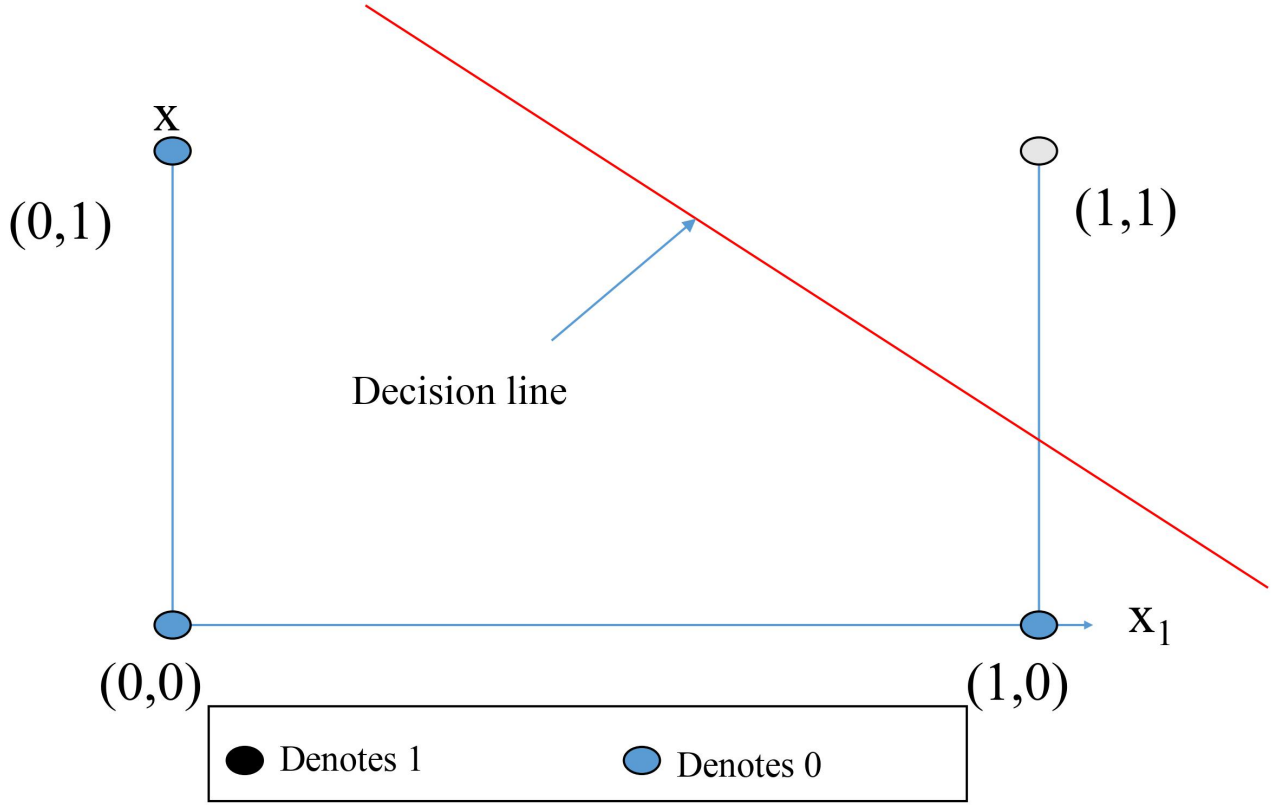
The perceptron below performs the OR operation:



Rosenblatt's Perceptron's …

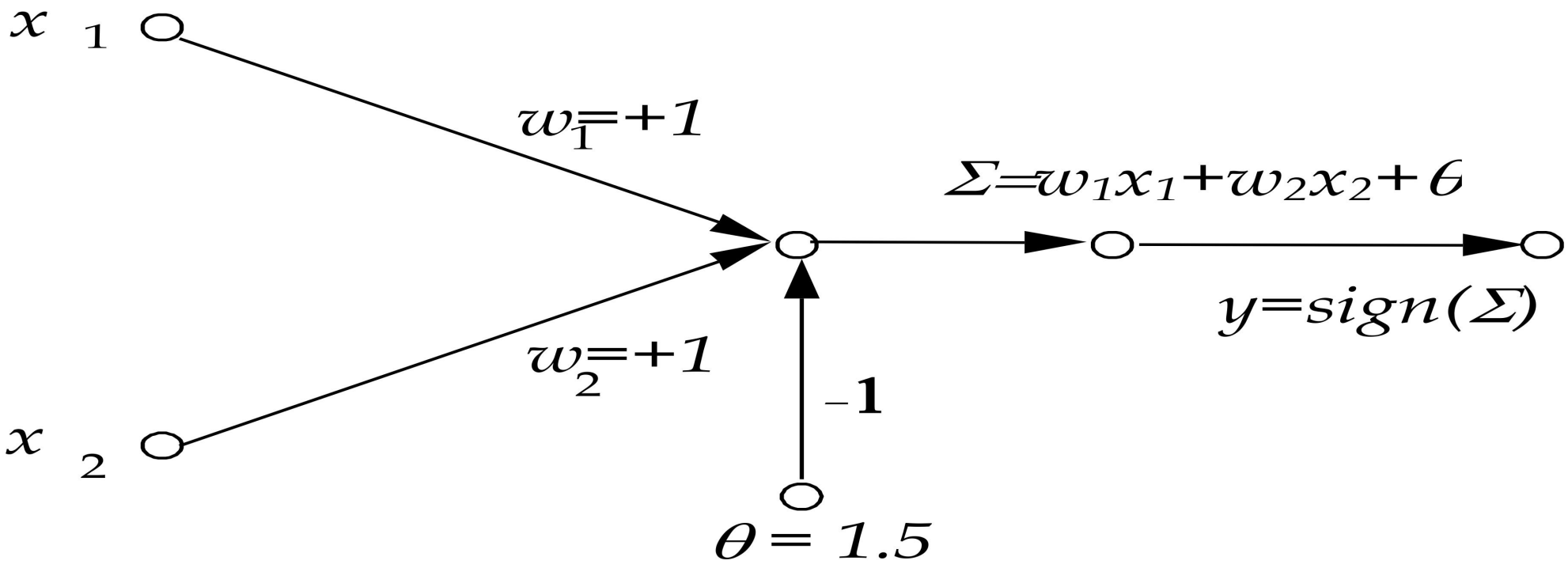
Definition of AND

Input		Output
X_1	X_2	Y
0	0	0
0	1	0
1	0	0
1	1	1



Rosenblatt's Perceptron ...

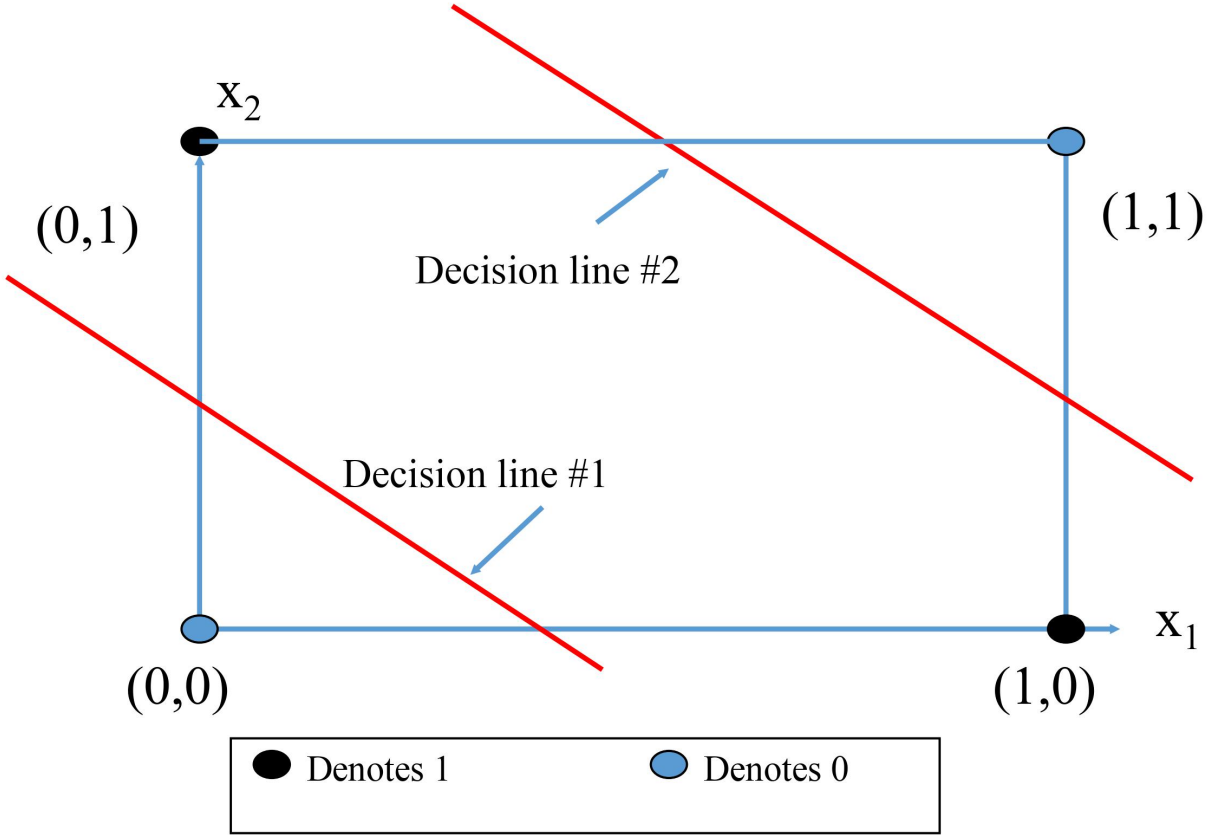
The perceptron below performs the AND operation:



Rosenblatt's Perceptron ...

Definition of XOR

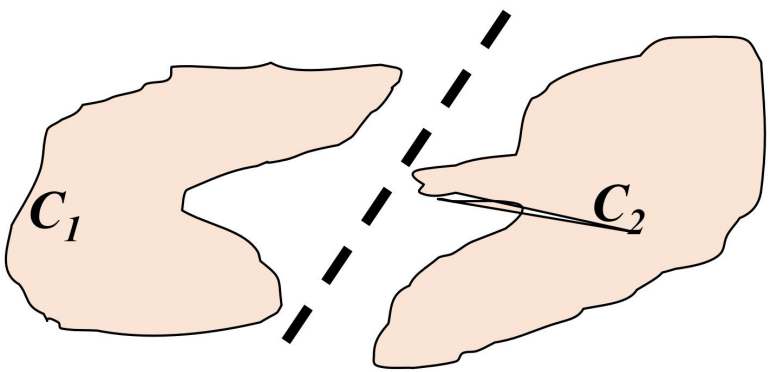
Input		Output
X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0



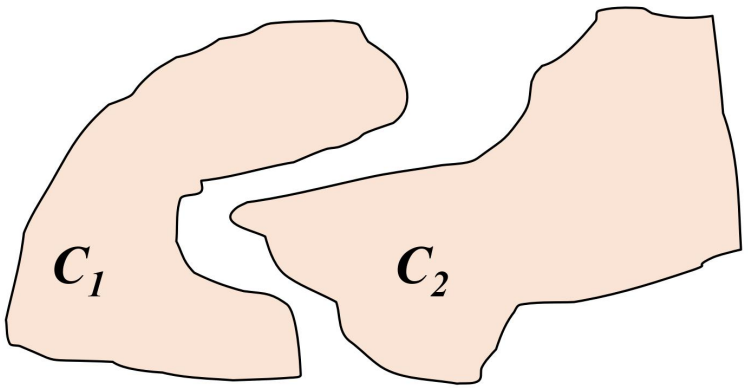
Rosenblatt's Perceptron

- A single layer perceptron can perform a number of logical operations which are performed by a number of computational devices.
- However, the single layer perceptron cannot perform the *exclusive-OR* or *XOR* operation.
- The reason is that a single layer perceptron can only classify two classes, say C_1 and C_2, should be sufficiently separated from each other to ensure the decision surface consists of a hyper-plane.

Linearly separable classes



Linearly non-separable classes



The X-OR Problem

- The simple perceptron cannot learn a linear decision surface to separate the different outputs, *because no such decision surface exists.*
- Such a non-linear relationship between inputs and outputs as that of an XOR-gate are used to simulate vision systems that can tell whether a line drawing is connected or not, and in separating figure from ground in a picture.
- For simulating the behaviour of an XOR-gate we need to draw *elliptical decision surfaces* that would encircle two '1' outputs: A simple perceptron is unable to do so.
- Solution? Employ two separate line-drawing stages.

The X-OR Problem ...

- One line drawing to separate the pattern
  - where both the inputs are ‘0’ leading to an output ‘0’

and another line drawing to separate the remaining three I/O patterns

  - where either of the inputs is ‘0’ leading to an output ‘1’
  - where both the inputs are ‘0’ leading to an output ‘0’

The X-OR Problem ...

- In effect we use two perceptron's to solve the XOR problem: The output of the first perceptron becomes the input of the second.
- If the first perceptron sees both inputs as ‘1’. it sends a massive inhibitory signal to the second perceptron causing it to output ‘0’.
- If either of the inputs is ‘0’ the second perceptron gets no inhibition from the first perceptron and outputs 1, and outputs ‘1’ if either of the inputs is ‘1’.

Rosenblatt's Perceptron

- **The XOR 'solution'**

- The multilayer perceptron designed to solve the XOR problem has a serious problem.

- The perceptron convergence theorem does not extend to multilayer perceptron. The perceptron learning algorithm can adjust the weights between the inputs and outputs, but it cannot adjust weights between perceptron.

- For this we have to wait for the back-propagation learning algorithms.

Acknowledgement

- AIMA = Artificial Intelligence: A Modern Approach by Stuart Russell and Peter Norving (3rd edition)
- UC Berkeley (Some slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley)
- U of toronto
- Other online resources

Thank You