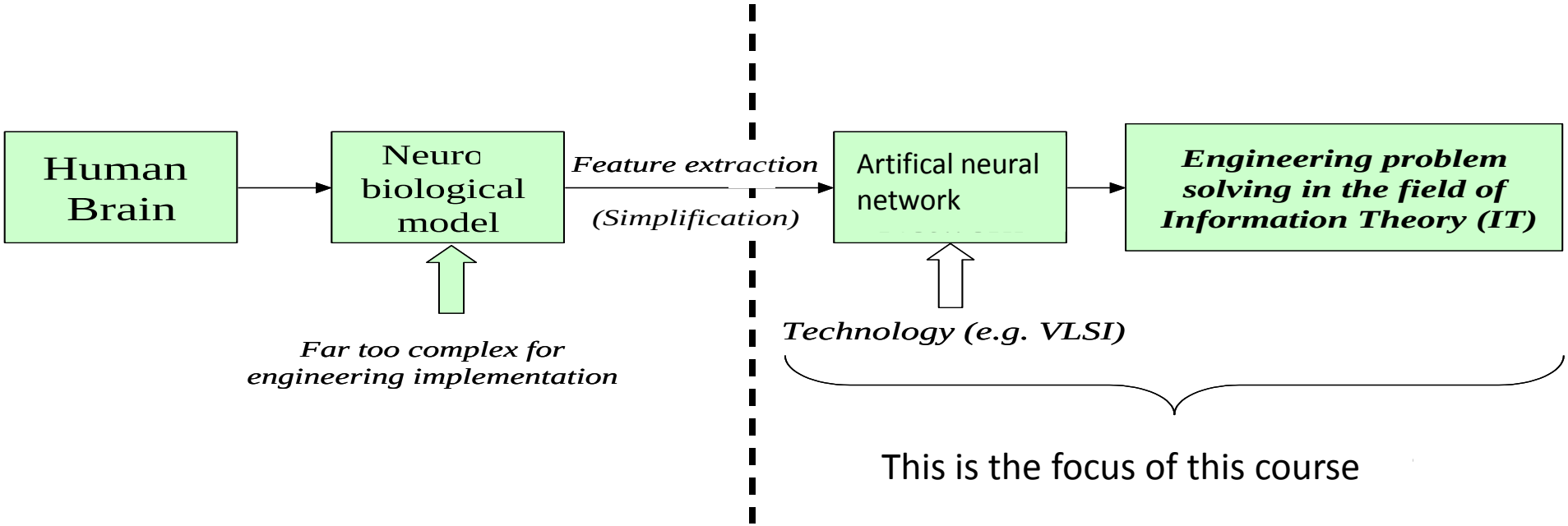
Neural Networks

(P-ITEEA-0011)

Introduction to the course
Single layer perceptron

Akos Zarandy
Lecture 1
September 10, 2018

Copying the brain?

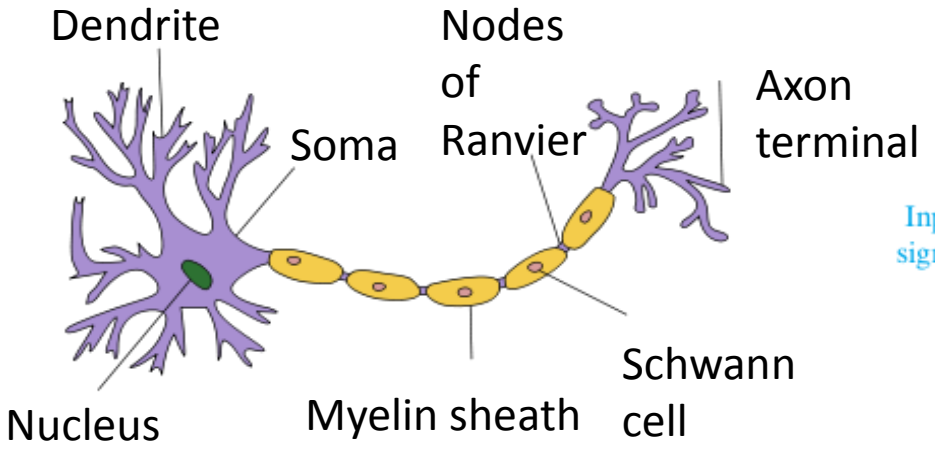


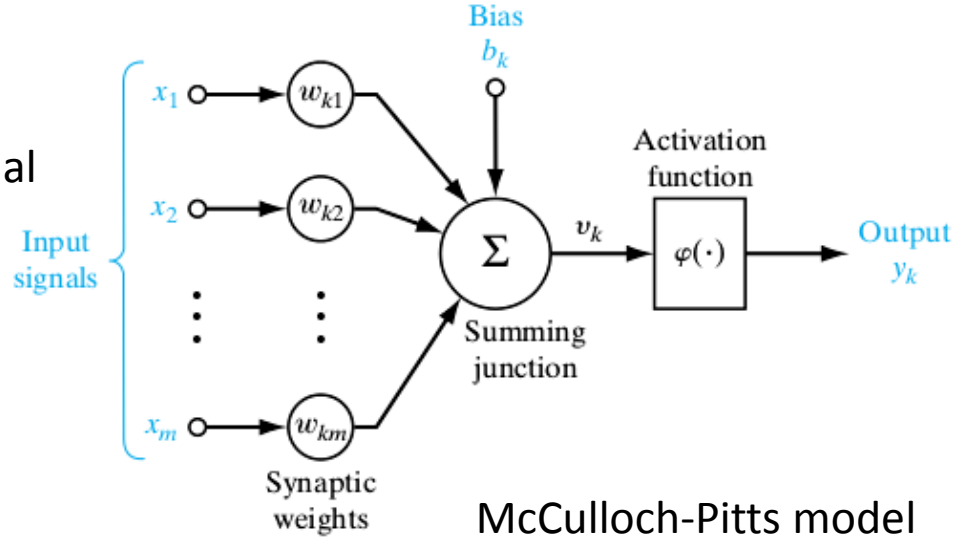
History of the artificial neural networks

- Artificial neuron model, 40's (McCulloch-Pitts, J. von Neumann);
- Synaptic connection strenghts increase for usage, 40's (Hebb)
- Perceptron learning rule, 50's (Rosenblatt);
- ADALINE, 60's (Widrow)
- Critical review ,70's (Minsky)
- Feedforward neural nets, 80's (Cybenko, Hornik, Stinchcombe..)
- Back propagation learning, 80's (Sejnowsky, Grossberg)
- Hopfield net, 80's (Hopfield, Grossberg);
- Self organizing feature map, 70's - 80's (Kohonen)
- CNN, 80's-90's (Roska, Chua)
- PCA networks, 90's (Oja)
- Applications in IT, 90's - 00's
- SVMs, statistical machines 2000-2010's
- Deep learning, Convolutional Neural Networks 2010-

The artificial neuron (McCulloch-Pitts)

- The artificial neuron is an information processing unit that is basic constructing element of an artificial neural network.
- Extracted from the biological model

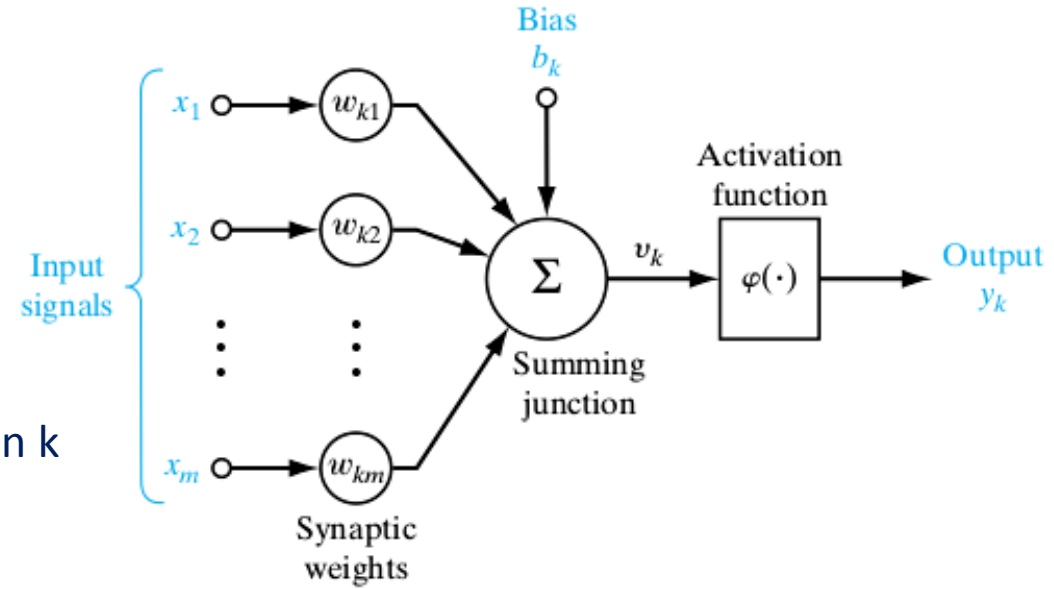




McCulloch-Pitts model

The artificial neuron

- Receives input through its synapsis (x_i)
- Synapsis are weighted (w_i)
  - if $w_i > 0$: amplified input from that source (excitatory input)
  - if $w_i < 0$: attenuated input from that source (inhibitory input)
- A *b* value biases the sum to enable asymmetric behavior
- A weighted sum is calculated
- Activation function shapes the output signal



x_i : input vector
w_{ki} : weight coefficient vector of neuron k
b_k : bias value of neuron k
o_k : output value of neuron k

The artificial neuron

- Output equation:

$$y_k = \varphi\left(\sum_{i=1}^{m} w_{ki} x_i + b_k\right)$$

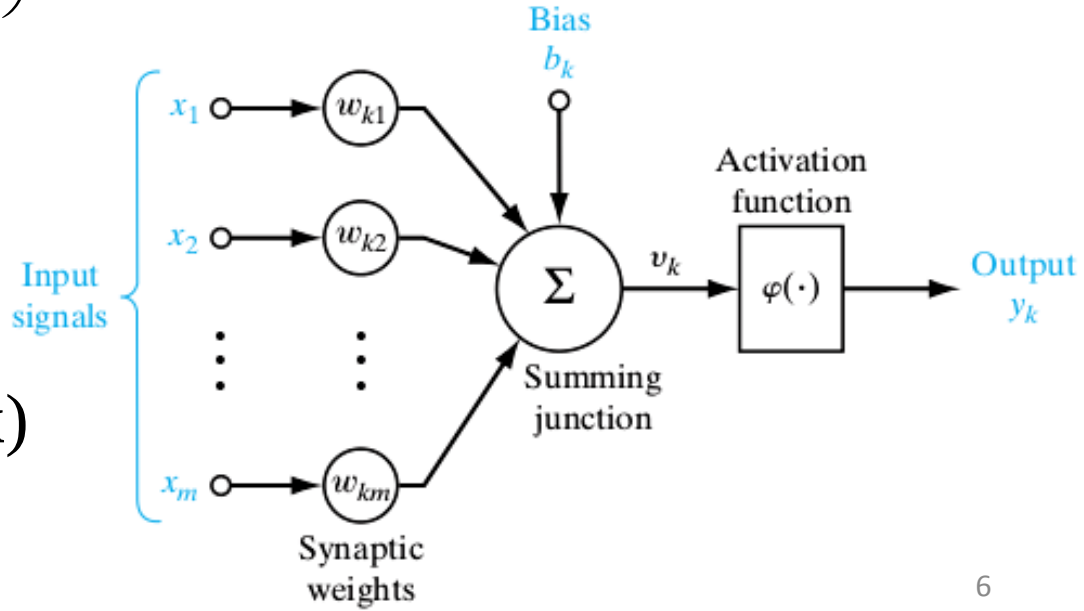
x_i : input vector (*i: 1….m*)
w_{ki} : weight coefficient vector of neuron k
b_k : bias value of neuron k
o_k : output value of neuron k

- Bias can be included as:

$w_0 = b$

$x_0 = 1$

$$y_k = \varphi\left(\sum_{i=0}^{m} w_{ki} x_i\right) = \varphi(\mathbf{w}^T \mathbf{x})$$



Activation functions (1)

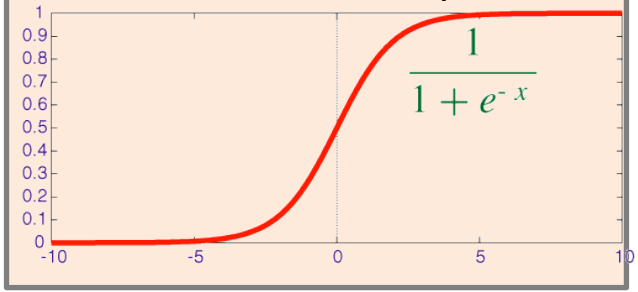
- Activation or threshold function: φ(.)
- φ(.): monotonic differentiable (not necessarily continiously differentiable) increasing function,
- Typically sigmoidal function is used as activation function (bipolar)

$$y = \varphi(u) = \frac{2}{1+e^{-\lambda u}} - 1$$

- where

$$u = \sum_{i=0}^{m} w_i x_i \quad = \mathbf{w}^T \mathbf{x}$$

Note: the original sigmoid function is unipolar.

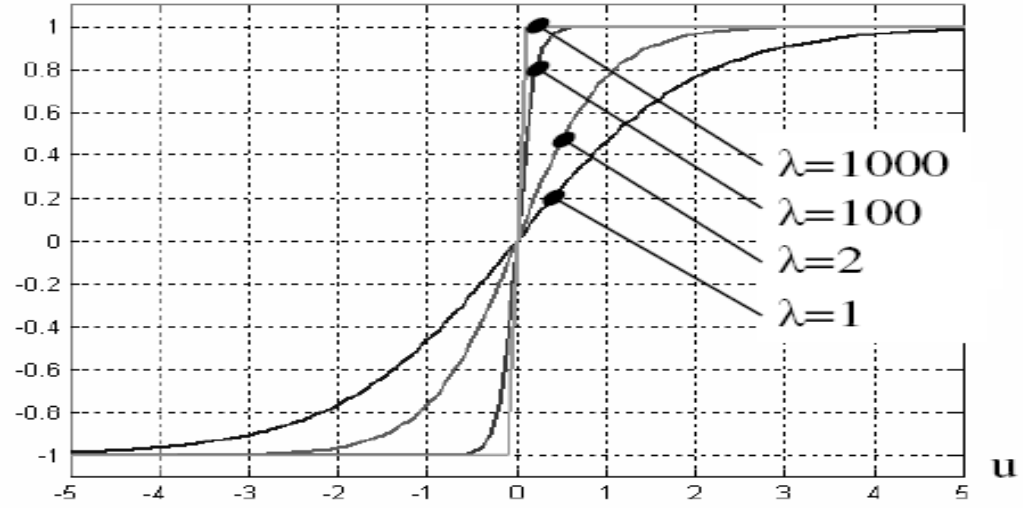


Activation functions (2)

- Bipolar sigmoidal type nonlinearities :

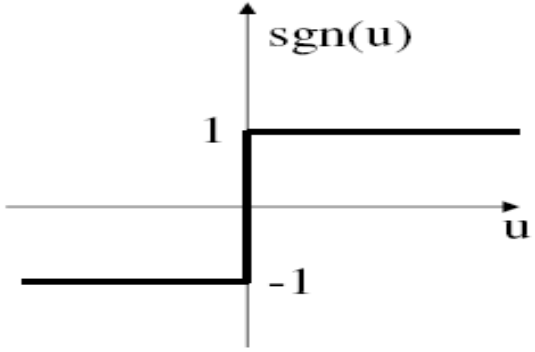
Soft nonlinearity

$$f(u) = \frac{2}{1+e^{-\lambda \cdot u}} - 1 \quad \xrightarrow{\lambda \to \infty}$$



Hard nonlinearity

$$f(u) = \mathrm{sgn}(u) = \begin{cases} 1 & \text{if} \quad u \geq 0 \\ -1 & \text{if} \quad u < 0 \end{cases}$$



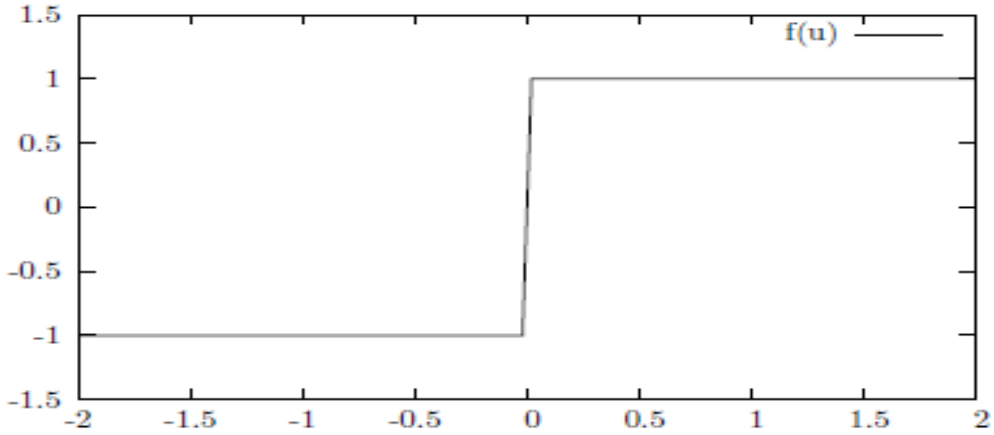
Activation function (3)

- Hard nonlinearity type activation function is often used for simplification
  - *It cannot be differentiated everywhere!*
- In this case the output

$$y = \operatorname{sgn}(u) = \begin{cases} 1, & \text{if } u \geq 0 \\ -1, & \text{else} \end{cases},$$

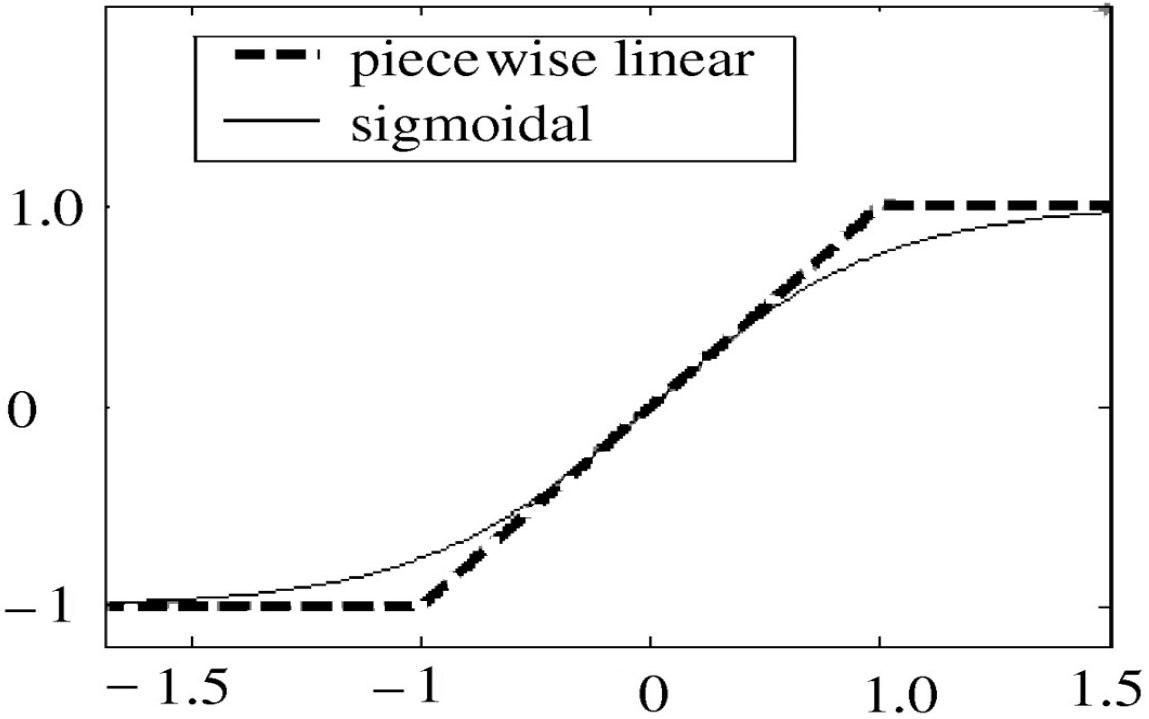
- where

$$u = \sum_{i=0}^{m} w_i x_i = \mathbf{w}^T \mathbf{x}$$



Activation function (4)

- Soft sigmoidal nonlinearity if approximated with piece-wise linear activation function
  - *It cannot be differentiated continuously!*



Elementary set separation by a single neuron (1)

- Let be the φ(.) hard nonlinear function, hence the output is discrete -1 or 1 with this assumption:

$$\varphi(u) = \operatorname{sgn}(u) = \begin{cases} +1, & \text{if } u \geq 0 \\ -1, & \text{else} \end{cases}.$$

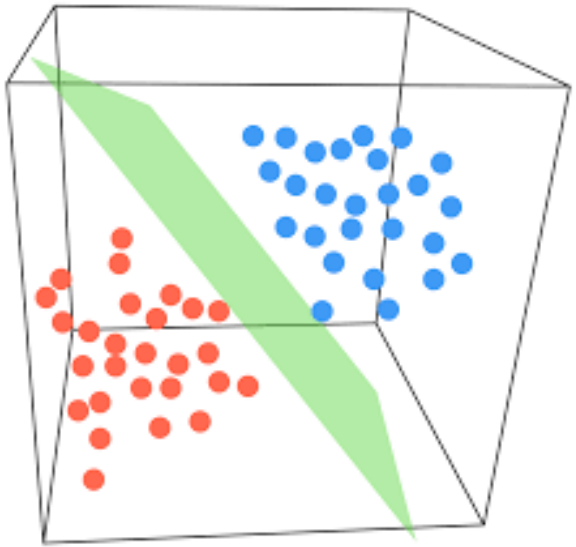
- Use the formula substituting *u* to $\mathbf{w}^T\mathbf{x}$ and then the output *+1*, if the weighted sum of the input is greater than zero or *−1* if the argument is smaller than zero.

$$y = \varphi(u) = \operatorname{sgn}\left(\mathbf{w}^T\mathbf{x}\right) = \begin{cases} +1, & \text{if } \mathbf{w}^T\mathbf{x} \geq 0 \\ -1, & \text{else} \end{cases}.$$ DECISION!

Elementary set separation by a single neuron (2)

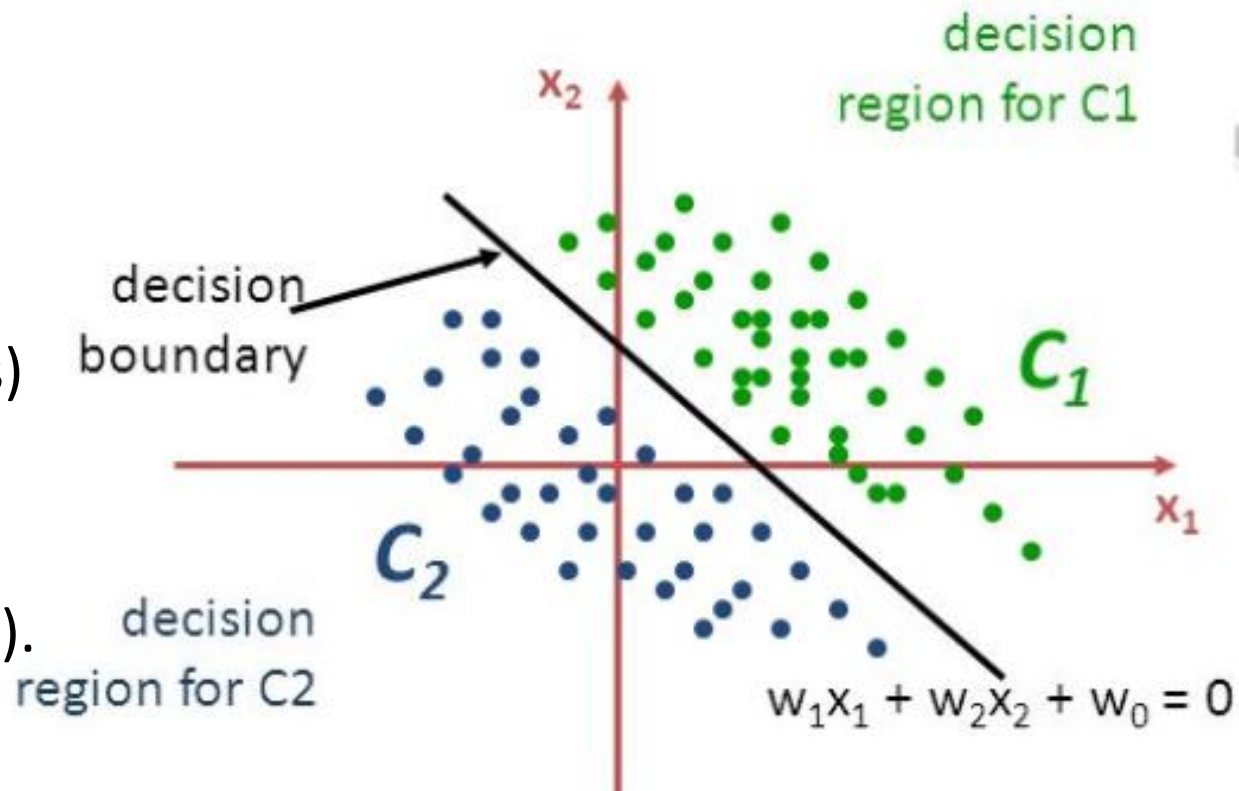
- Neuron with *m* inputs has an *m* dimensional input space
- Neuron makes a linear decision for a 2 class problem
  - Two outputs
- The decision boundary is a hyperplane defined:

$$\mathbf{w}^T\mathbf{x} = 0$$



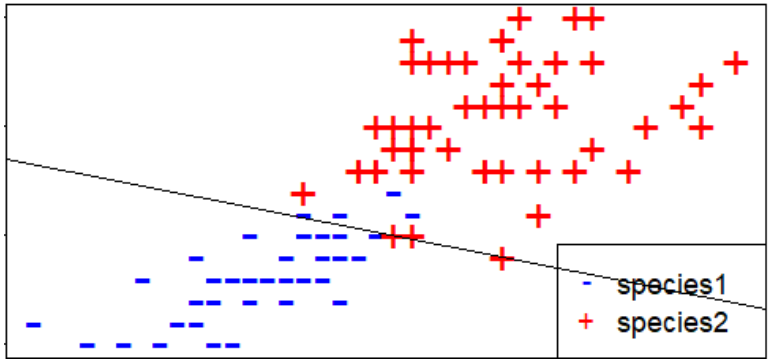
Elementary set separation by a single neuron (3)

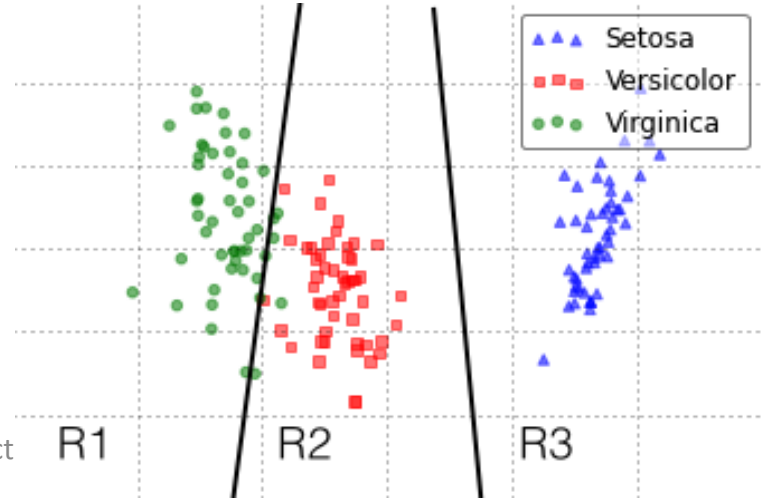
- in a 2-D input space, the hyper plane is a straight line.
- Above the line is classified: +1 (C1: yes)
- Below the line is classified : −1 (C2: no).



Why it is so important to use set separation by hyper plane? (1)

- Most logic functions has this complexity (OR, AND)
- There are plenty of mathematical and computational task which can be derived to a set separation problem by a linear hyper plane
- Application of multiple hyper plane provides complex decision boundary

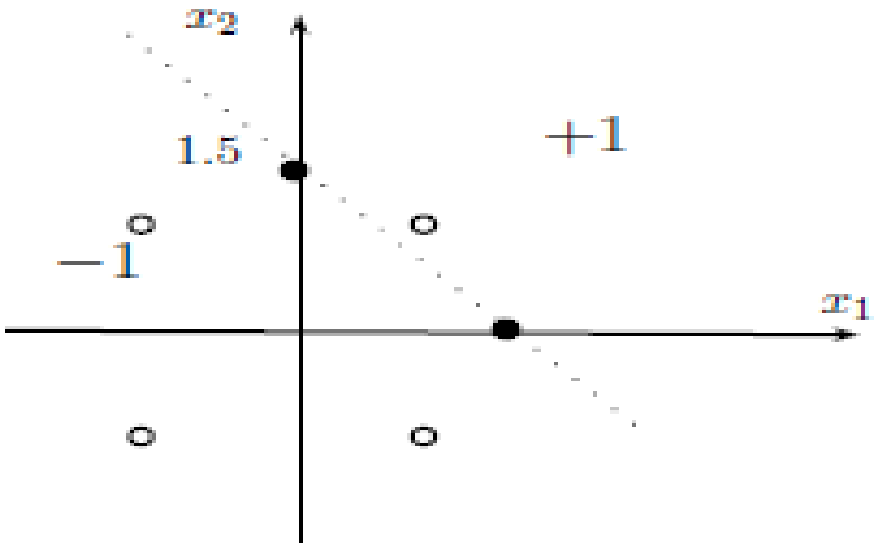




Implementation of a single logical function by a single neuron (1)

x_1	x_2	y
-1	-1	-1
-1	$+1$	-1
$+1$	-1	-1
$+1$	$+1$	$+1$

- The truth table of the logical AND function.



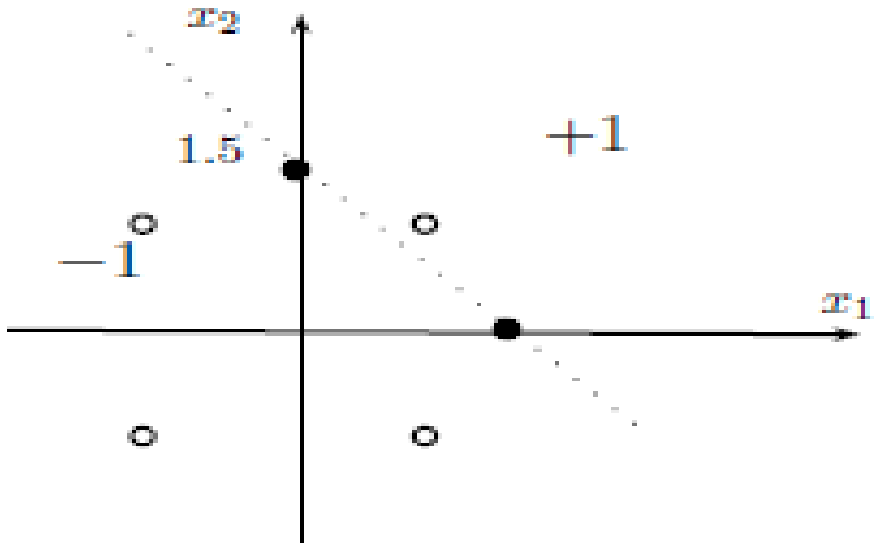
- 2-D AND input space and decision boundary

Implementation of a single logical function by a single neuron (2)

- We need to figure out the separation surface!
- Mathematically is the following equation:

$$-1.5 + x_1 + x_2 = 0$$

w_0=-1.5; w_1=1; w_2=1;



- The weight vector is:

w = (−1.5, 1, 1).

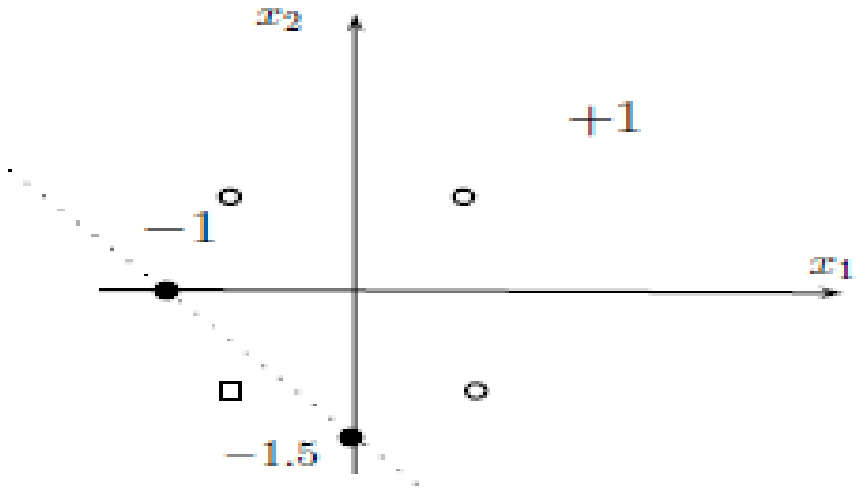
Implementation of a single logical function by a single neuron (3)

- Furthermore instead of 2D, we can actually come up with the *R* dimensional AND function.
- The weights corresponding to the inputs are all 1 and threshold should be $R - 0.5$. As a result the actual weights of the neuron are the following:

$$\mathbf{w}^{\mathrm{T}} = \left(-(R-0.5), 1, \ldots, 1\right)$$

Implementation of a single logical function by a single neuron (4)

x_1	x_2	y
-1	-1	-1
-1	$+1$	$+1$
$+1$	-1	$+1$
$+1$	$+1$	$+1$



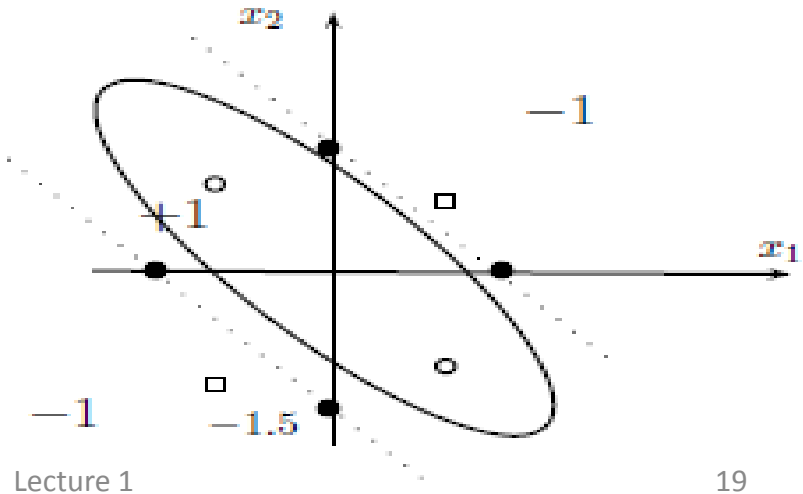
- The truth table of the logical OR function.
- 2-D OR input space and decision boundary

w =(−0.5, 1, 1).

Implementation of a single logical function by a single neuron (5)

- However we cannot implement every logical function by a linear hyper plane.
- Exclusive OR (XOR) cannot be implemented by a single neuron (linearly not separable)

x_1	x_2	y
-1	-1	-1
-1	$+1$	$+1$
$+1$	-1	$+1$
$+1$	$+1$	-1



The learning algorithm (1)

- What happens in a complex case?
- We have a learning set only
  - d: desired output
- How come we dot know?
  - A human expert can tell the decision

$$X^+ = \{\mathbf{x} : \ d = +1\}$$
$$X^- = \{\mathbf{x} : \ d = -1\}$$

- An artificial neuron can function properly, if the two classes X^+ and X^- must be linearly separable
- We are looking for an optimal parameter set:

$$X^+ = \{\mathbf{x} : \ \mathbf{w}_{\text{opt}}^{\text{T}}\mathbf{x} \geq 0\},$$
$$X^- = \{\mathbf{x} : \ \mathbf{w}_{\text{opt}}^{\text{T}}\mathbf{x} < 0\}.$$

The learning algorithm (2)

- We have to develop a recursive algorithm called learning, which can learn step by step, based on observing the previous weight vector, the desired output and the actual output of the system.
- On these specific examples it is going to recursively adopt the weight vector in order to converge $\mathbf{w}_{opt}$. This can be described formally as follows:

$$\mathbf{w}(k+1) = \Psi\left(\mathbf{w}(k), \mathbf{d}(k), \mathbf{y}(k), \mathbf{x}(k)\right) \rightarrow \mathbf{w}_{opt}$$

The learning algorithm (3)

- In a more ambitious way it can be called intelligent
- Rosenblatt introduced perceptron learning algorithm (1958)
- Given the sets of vectors X^+ and X^- and an initial weight vector $\mathbf{w}(0)$, this algorithm can set an optimal weights vector $\mathbf{w}_{opt}$ to the perceptron.

The learning algorithm (4)

1. *Initialization*. Set **w**(0)=**0**. Then perform the following computations for time step n=1,2,…
2. *Activation*. At time step k, activate the perceptron by applying continuous-valued input vector **x**(k) and desired *d*(*k*).
3. *Computation of Actual response*. Compute the actual response of the perceptron:

$$y(k) = \mathrm{sgn}\left\{\mathbf{w}^T(k)\mathbf{x}(k)\right\}.$$

The learning algorithm (5)

4. *Adaptation of the weight vector*. Update the weight vector of the perceptron according to rule:

- where $$\mathbf{w}(k+1) = \mathbf{w}(k) + \left[d(k) - y(k) \right] \mathbf{x}(k),$$
- $$d(k) = \begin{cases} 1 & \text{if } \mathbf{x}(k) \text{ belongs to class } \mathrm{X}^{+} \\ -1 & \text{if } \mathbf{x}(k) \text{ belongs to class } \mathrm{X}^{-} \end{cases},$$
- and
- $$\varepsilon(k) = d(k) - y(k)$$
- is the error function.

The learning algorithm (6)

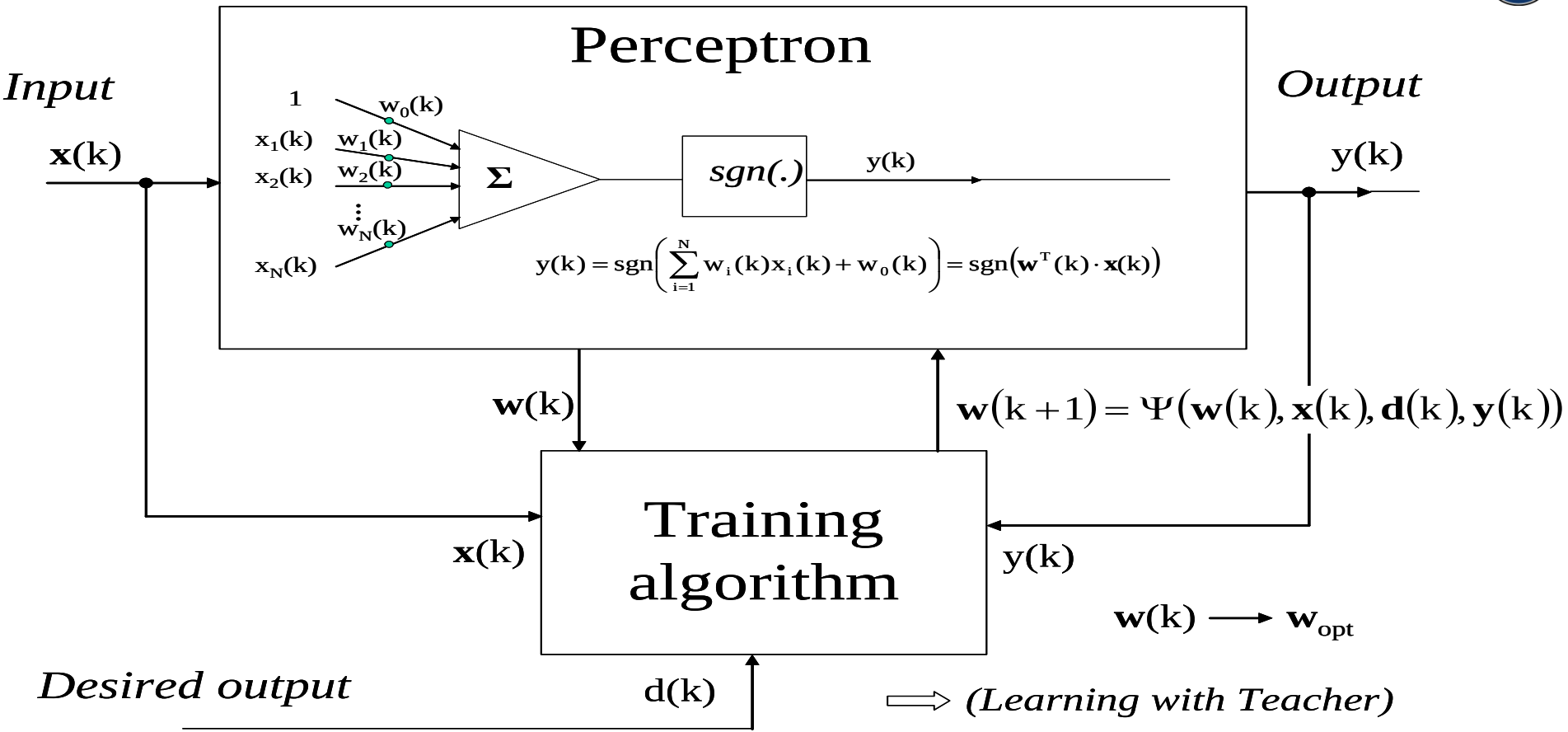
5. *Continuation*. Increment time step n-by-one and go back to step 2.

- Basically we feedback the error signal to adopt the weights more efficiently.

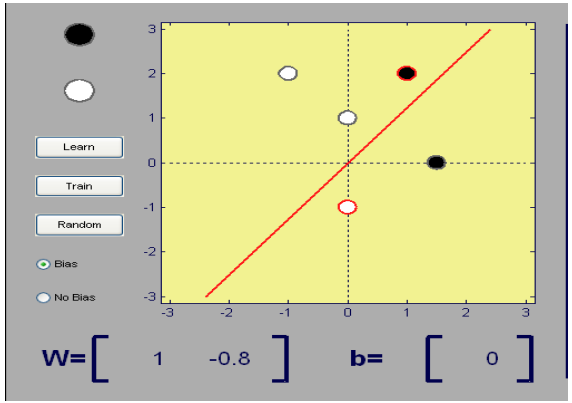
- One can come with the following questions:
  - if the algorithm converges to any fix point?
  - if there is a fix point, what is the speed of convergence?

The learning algorithm (7)

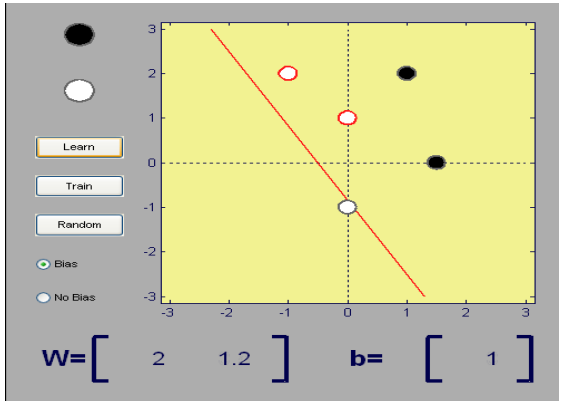
- The Rosenblatt learning algorithm:



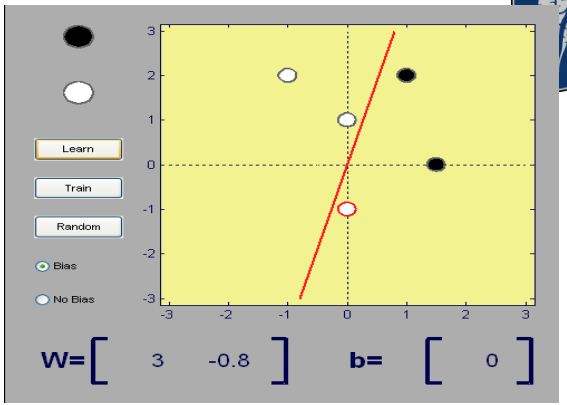
Perceptron learning rule – an example



1st step :initial state



2nd step

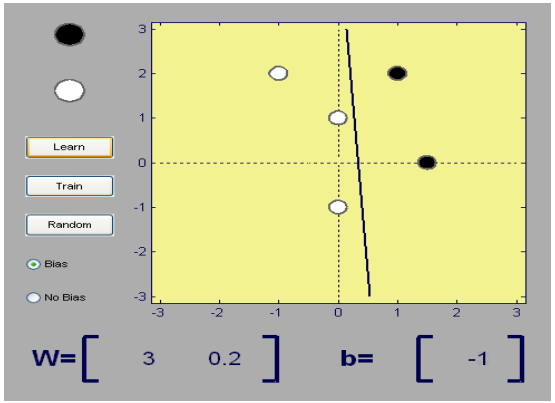


3rd step

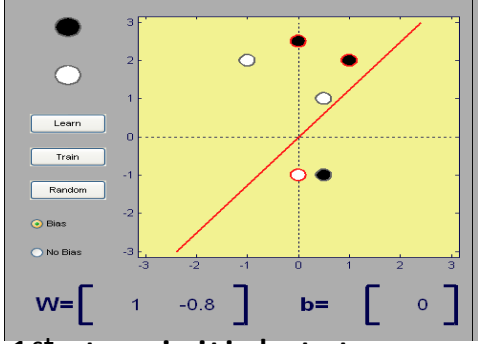
2D algorithm

$$w_i(k+1) = w_i(k) + \varepsilon(k)x_i(k)$$

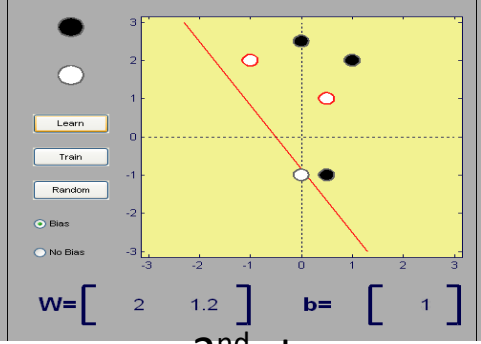
$$i = 1,2,3$$



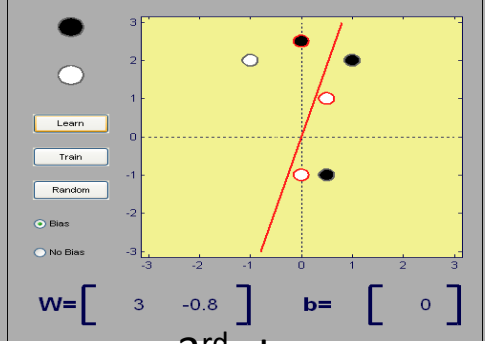
4th step:correct separation



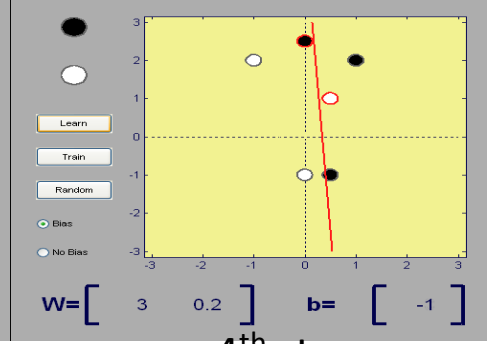
1st step:initial state



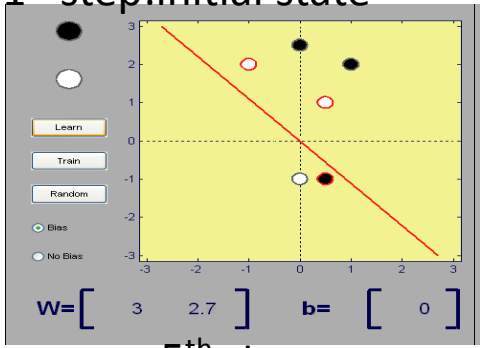
2nd step



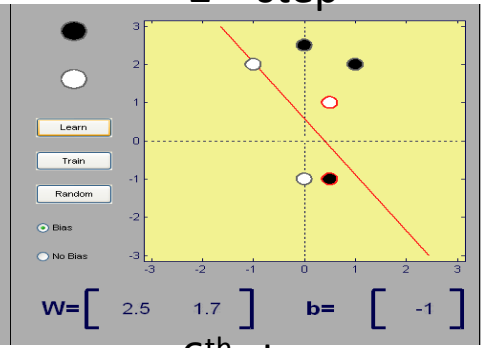
3rd step



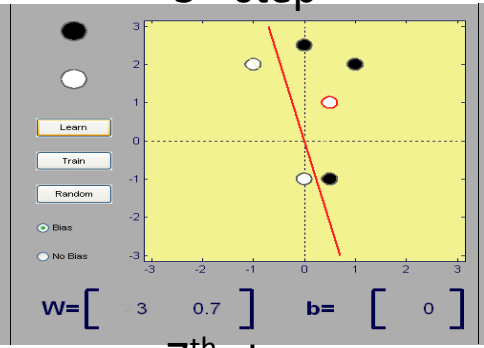
4th step



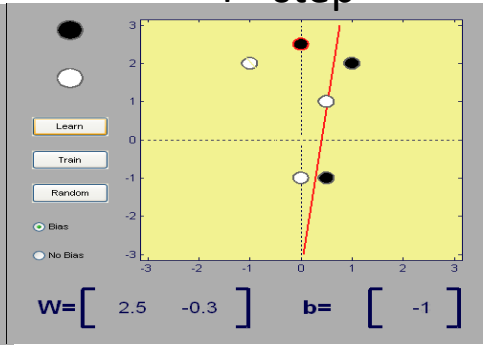
5th step



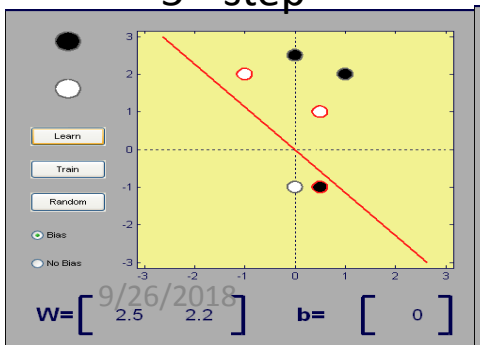
6th step

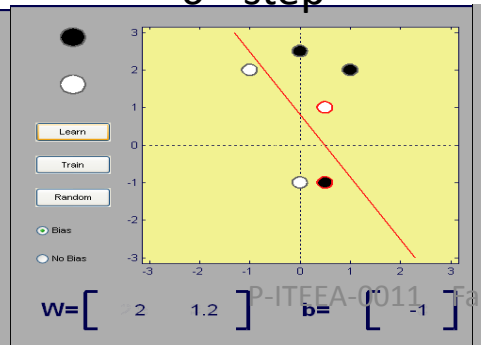


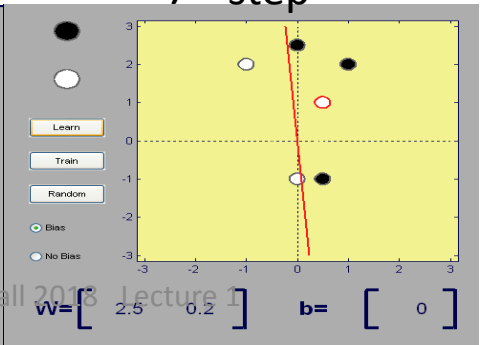
7th step



8th step







It cannot separate correctly