*Neural Networks*

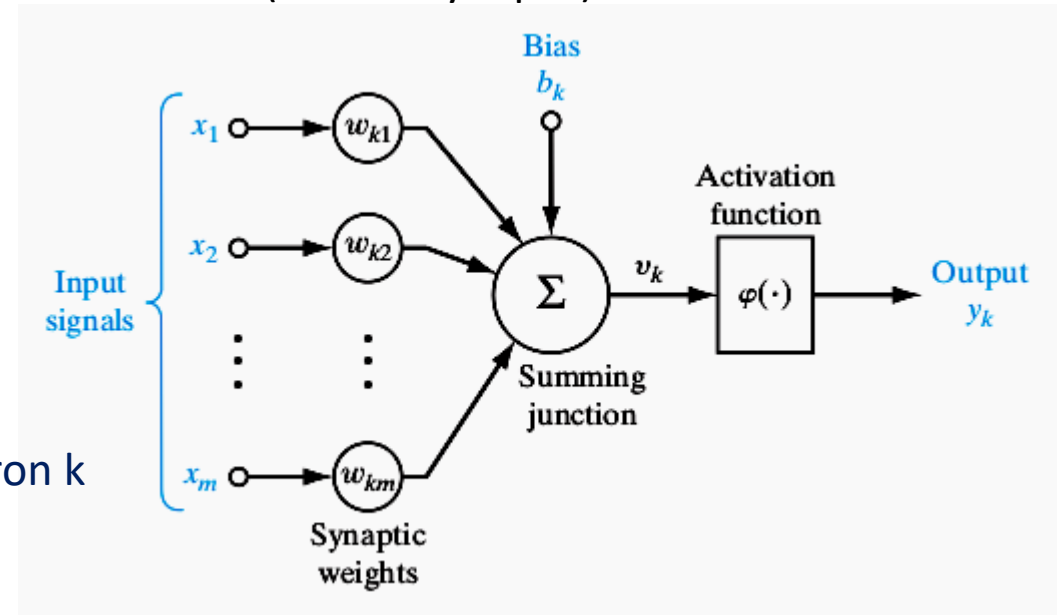
*(P-ITEEA-0011)*

# Perceptron learning

Akos Zarandy
Lecture 2
September 17, 2018

# 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:

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

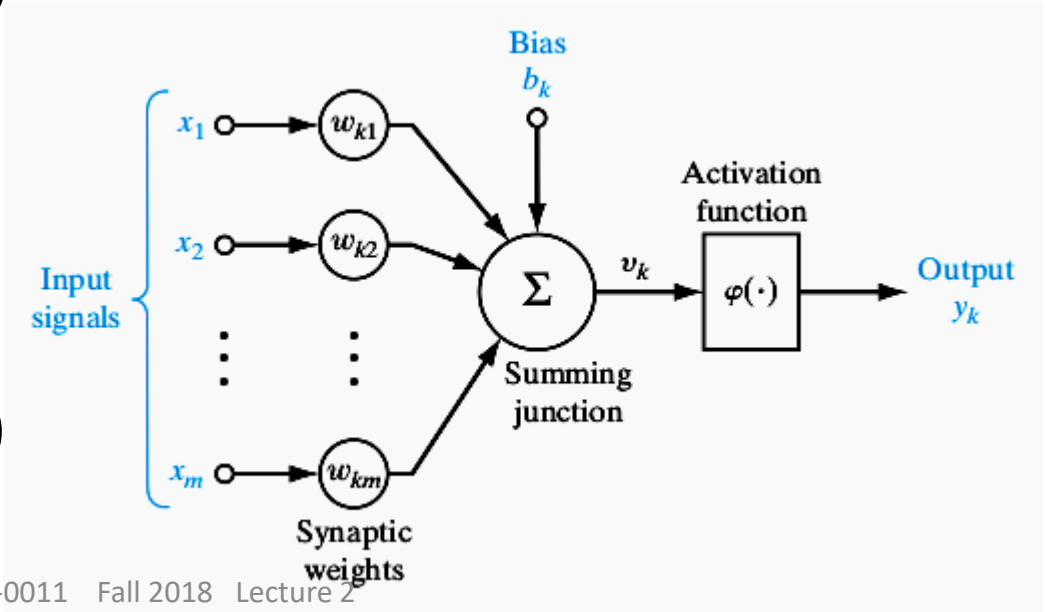
$x_i$ : input vector (*i: 1….m*)
$w_{ki}$ : weight coefficient vector of neuron k
$b_k$ : bias value of neuron k
$o_k$ : output value of neuron k

- Bias can be included as:

$w_0 = b$

$x_0 = 1$

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



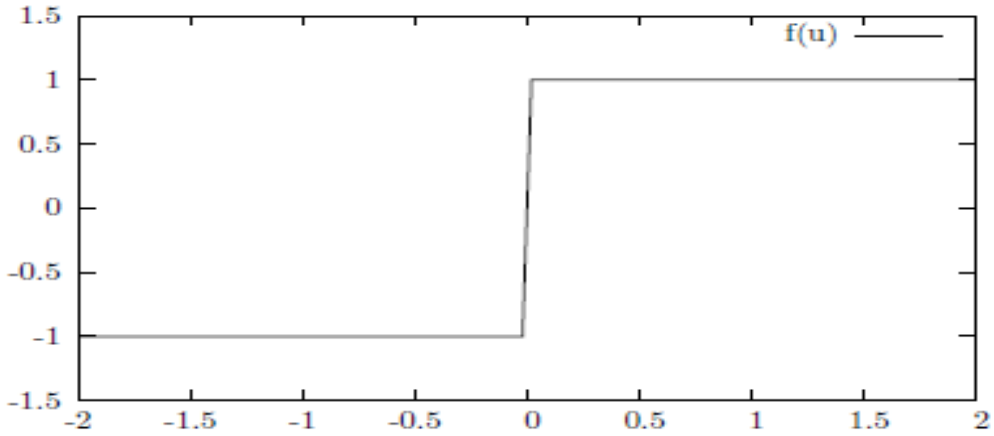
# Activation function

- Hard nonlinearity (hard threshold) type activation function is the signum function for the perceptron
- The output is:

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

- where

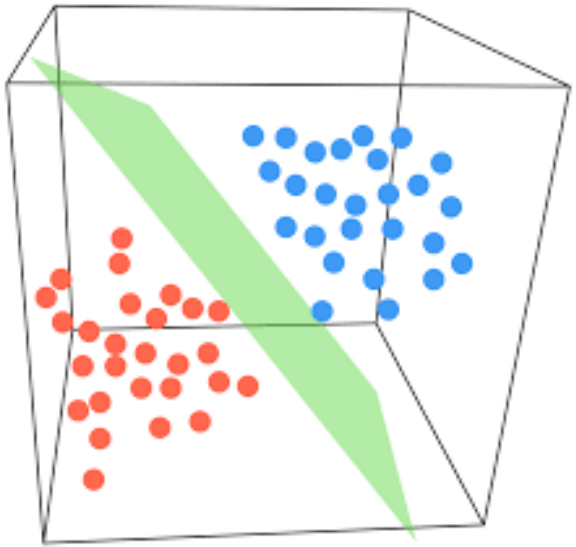
$$u = \sum_{i=0}^{m} w_i x_i = \mathbf{w}^T \mathbf{x}$$



# Elementary set separation by a single neuron (2)

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

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



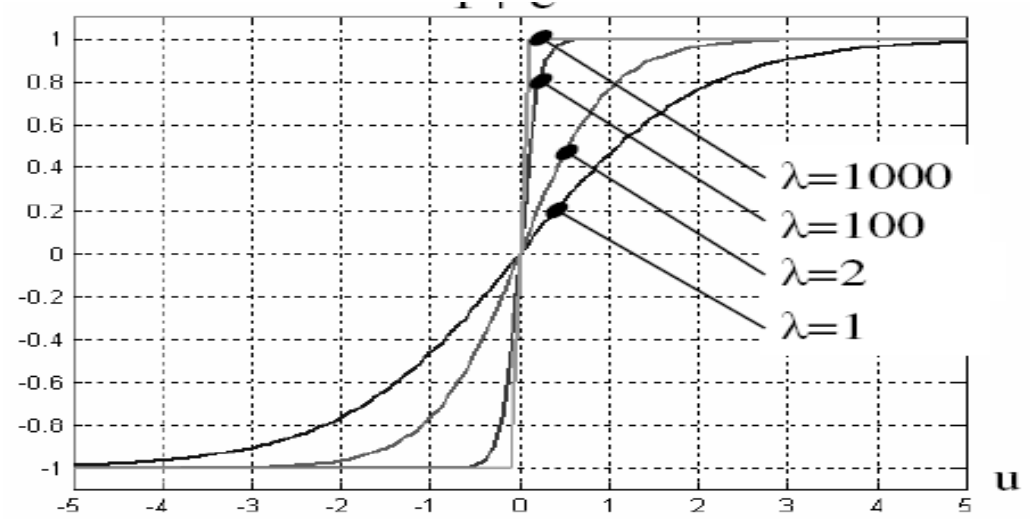
- Above the line is classified: +1 (yes)
- Below the line is classified : −1 (no).

# Sigmoid function

- Activation or threshold function: φ(.)
- φ(.): monotonic increasing function
- Typically sigmoid function is used as activation function
  - where

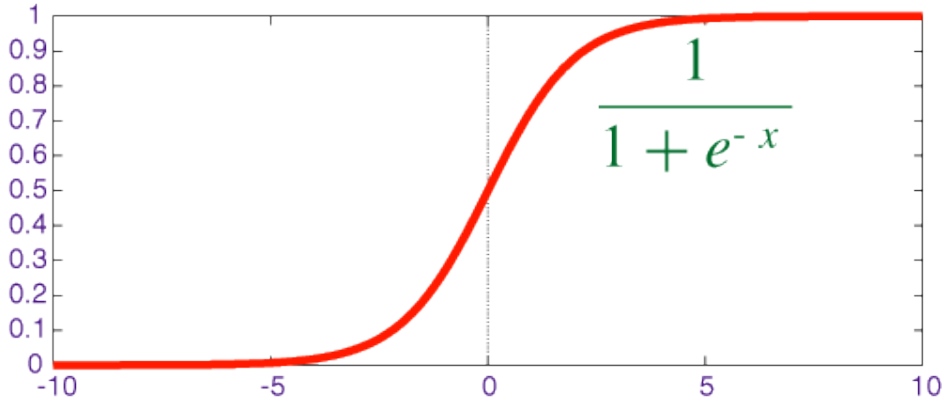
$$y = \varphi(u) = \frac{2}{1+e^{-\lambda u}} - 1$$

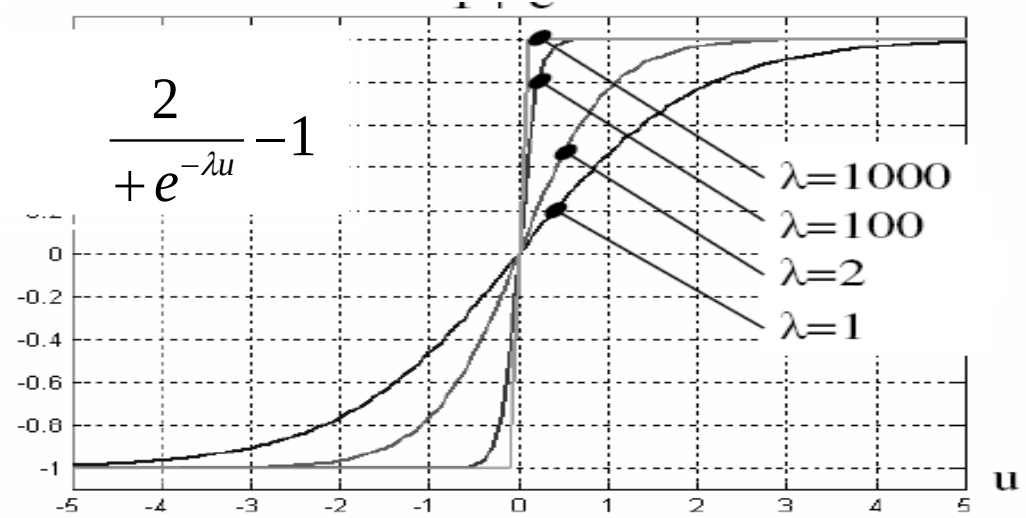
$$u = \sum_{i=0}^{m} w_i x_i = \mathbf{w}^T \mathbf{x}$$



# Sigmoid: unipolar or bipolar?

- Sigmoid is unipolar by definition
- Sigmoid type is can be bipolar

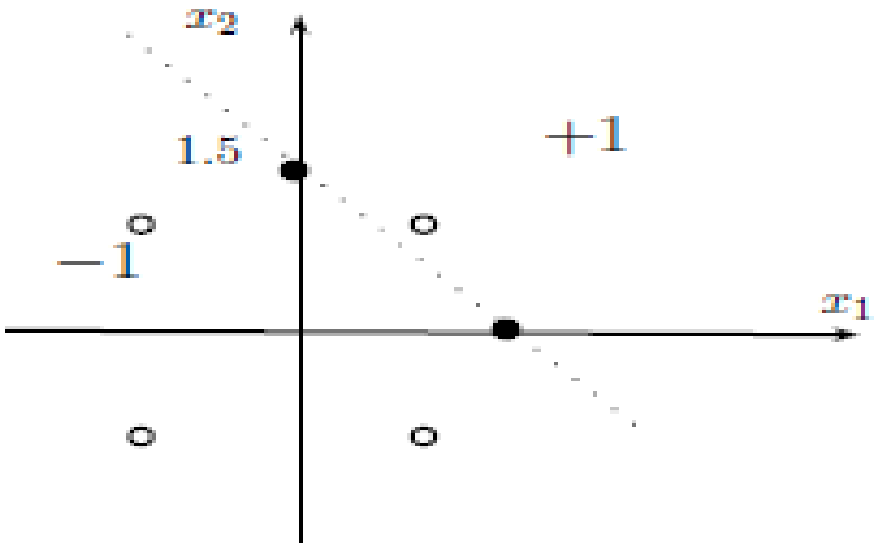


# Implementation of a single logical function by a single neuron (1)

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

- The truth table of the logical AND function.




- 2-D AND input space and decision boundary

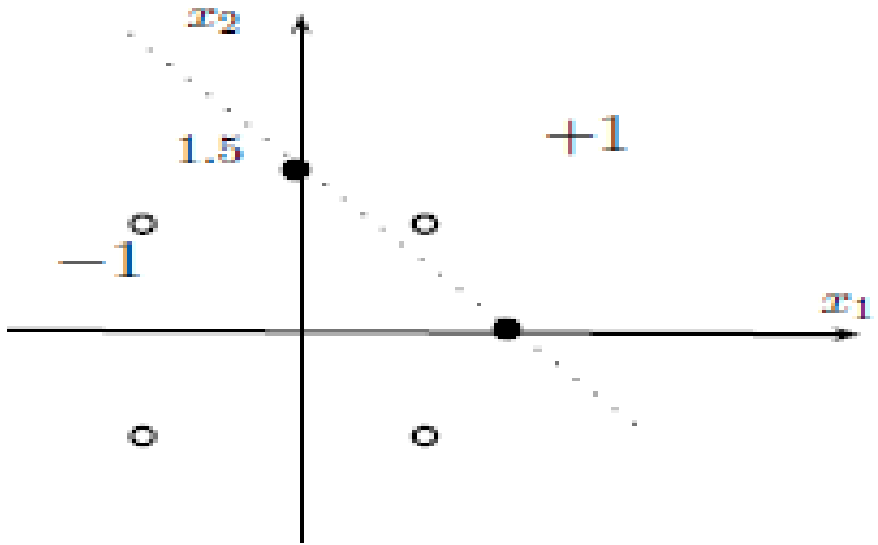
# Implementation of a single logical function by a single neuron (2)



- 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)

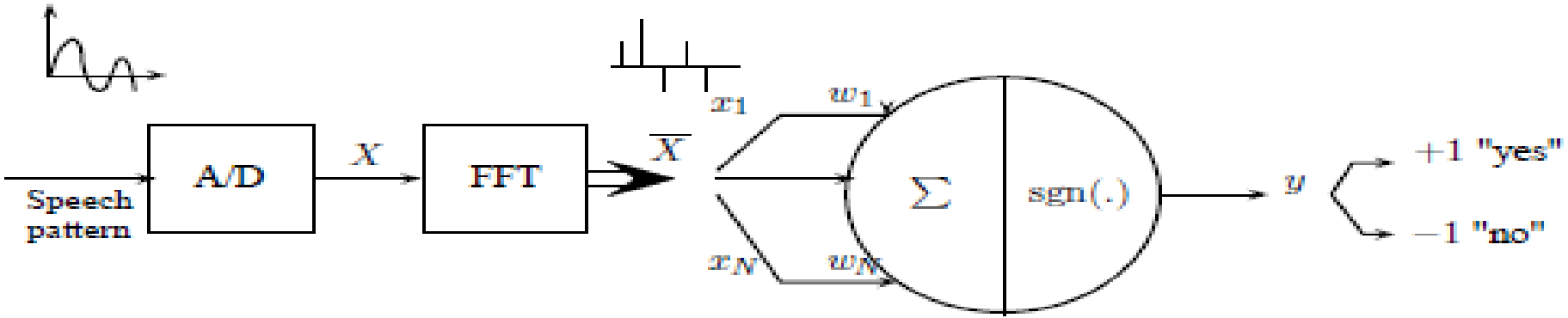
- Furthermore instead of 2D, we can actually come up with the *R* dimensional AND function.
- The weights corresponding to the inputs are all 1 and threshold should be *R* − 0.5. As a result the actual weights of the neuron are the following:

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

# Finding parameter set in a complex case

- Let assume that the input is a speech pattern in a form of continuous signal which represents a “yes” or a “no”.
- This continuous signal is going in an A/D converter where it is transformed to a simple digital signal.
- After that, the next step is to carry out the features with an FFT, which is represented by **X** vector.
- Assume that the **X** vectors are linearly separable according to “yes” or a “no”.
- Then, all of the components are plugged into an artificial neuron, which computes the weighted sum of the input, compares with a threshold, and if the state is greater or equal than the threshold the output is going to be *+1*, otherwise it is going to be *−1*.

# Finding parameter set in a complex case



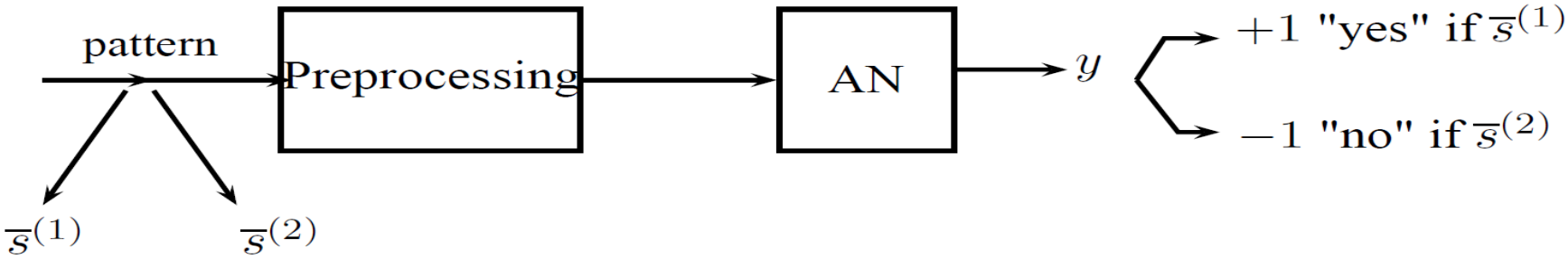

- The block diagram of the speech pattern recognition by an artificial neuron.

# Finding parameter set in a complex case

- This is how we would like to solve the speech pattern recognition task, with a separation by a linear hyper plane. The model is now correct, the next question is that what the weights are of this neuron, so what should be the program of the perceptron to get correct recognition.

- In a more general view every special pattern can arrive which has two possible value $\mathbf{s}^{(1)}$ or $\mathbf{s}^{(2)}$,which are going to represent the Fourier transformation of “yes” and the “no”.

# Finding parameter set in a complex case




- Generalization of a pattern recognition task by an artificial neuron.

# The learning algorithm (1)

- What happens in a complex case?
- We have a learning set only
  - d: desired output
- How come we don't know?
  - A human expert can tell the decision

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

- An artificial neuron (Perceptron) can function properly, if the two classes $X^+$ and $X^-$ must be linearly separable (assume it for now!!!)
- 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:

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

# The learning algorithm (3)

- In a more ambitious way it can be called intelligent, because artificial intelligence is a kind of philosophy that we can learn from example, even the parameters are fully hidden.

- Rosenblatt  learning algorithm (1969)

- 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)

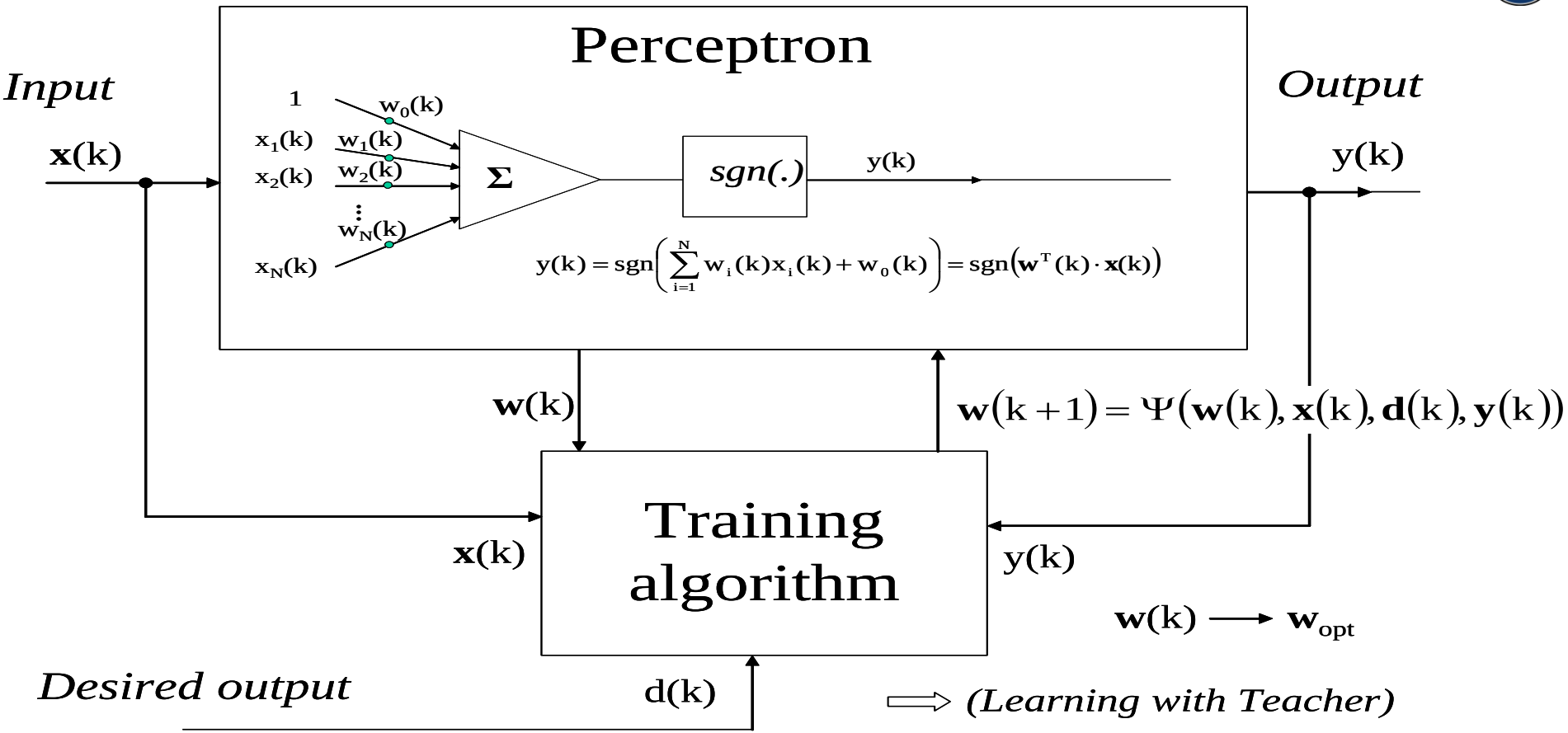
5. *Continuation*. Increment time step n-by-one and go back to step 2.

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

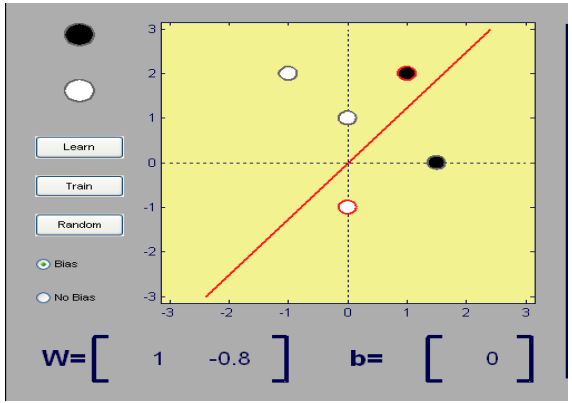
- One can come with the following questions:
  - if the algorithm converges to any fix point?
  - if there is a fix point, what is the speed of convergence?

# The learning algorithm (7)

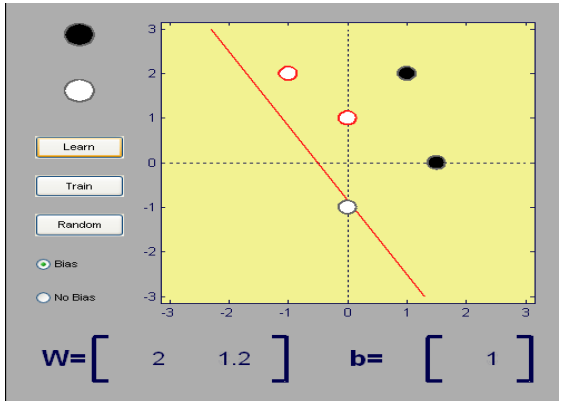
- The Rosenblatt learning algorithm:



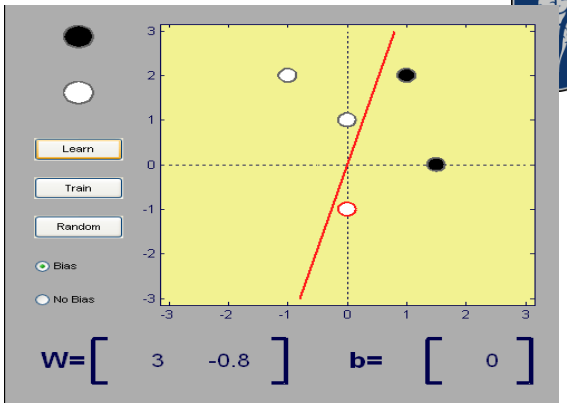
# Perceptron learning rule – an example




1st step :initial state




2nd step

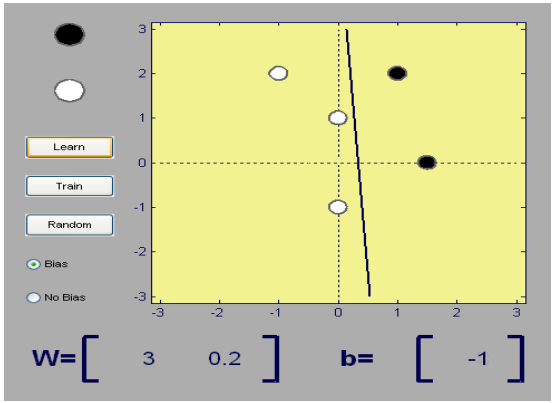



3rd step

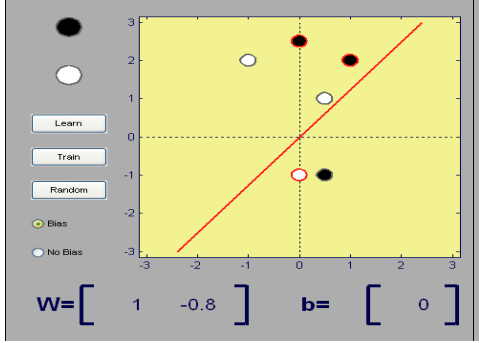
2D algorithm

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

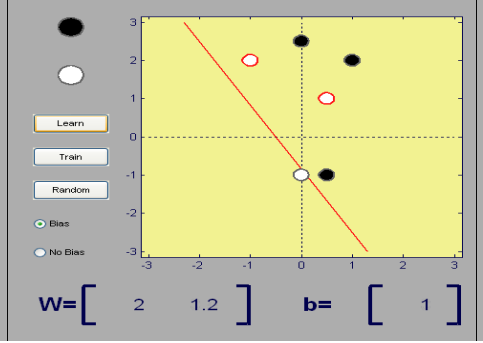
$$i = 1,2,3$$



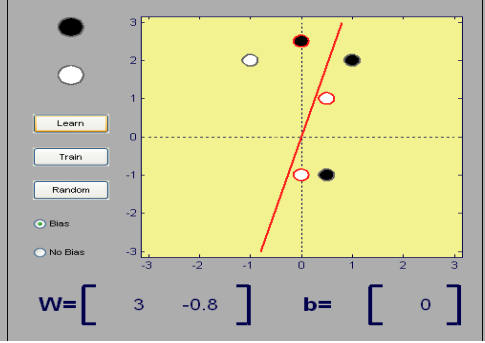

4th step:correct separation



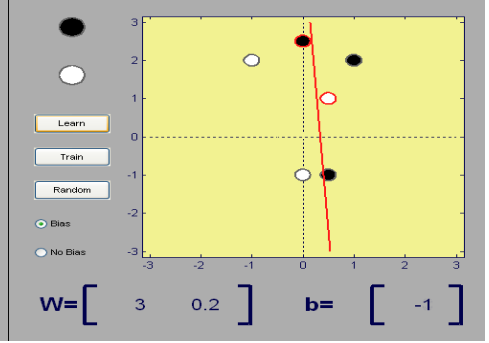

1st step:initial state



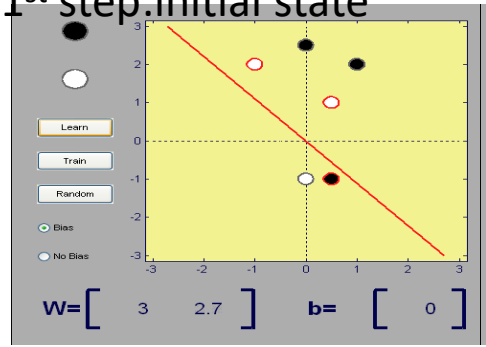

2nd step



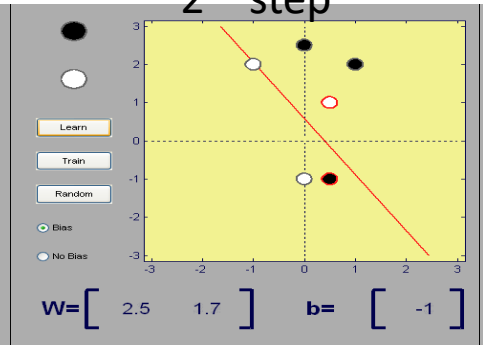

3rd step



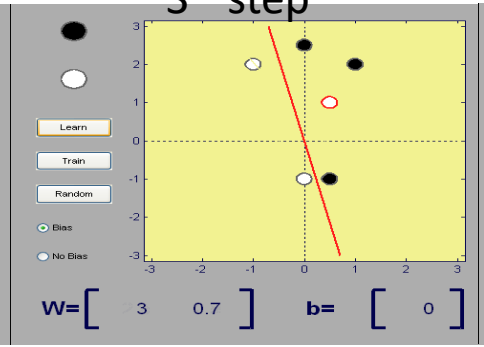

4th step



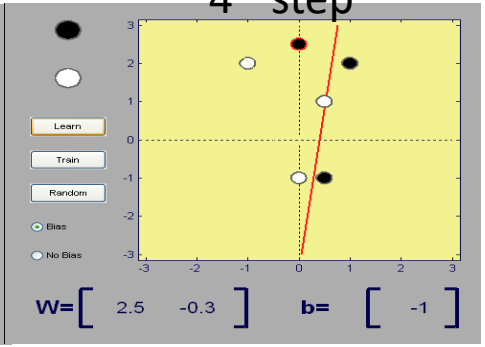

5th step



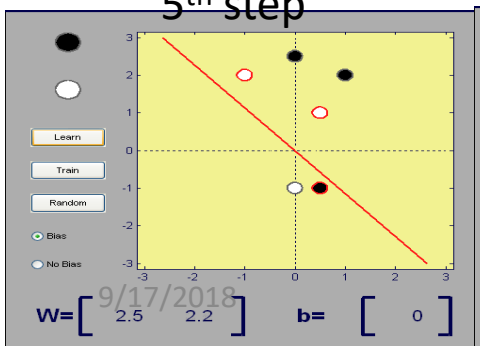

6th step

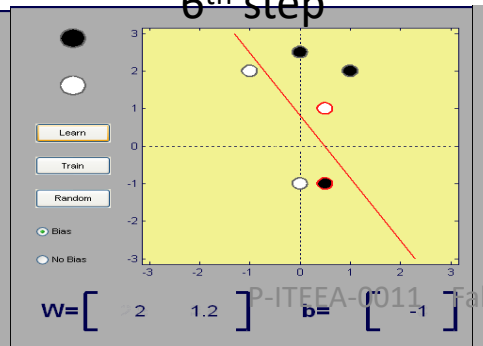


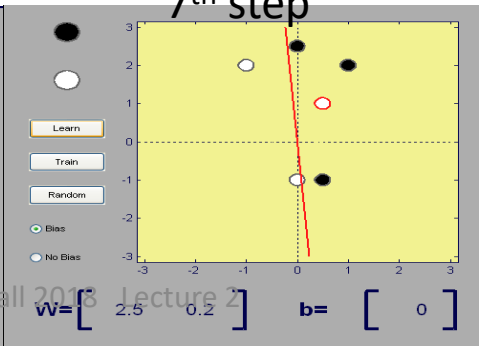

7th step




8th step




It cannot separate correctly

# Perceptron Convergence theorem (1)

*Assumptions:*

- ***w**(0)=0*
- *the input space is linearly separable, therefore* $\mathbf{w}_o$ *(stands for* $\mathbf{w}_{optimal}$*) exists:*

$$x \in X^+: \quad w_o^T x > 0: \; d = 1$$

$$x \in X^-: \quad w_o^T x < 0: \; d = -1$$

$$\tilde{x} \in \tilde{X}^-: \quad w_o^T \tilde{x} > 0: \; d = 1$$

# Perceptron Convergence theorem (2)

- Idea:
  - During the training, the network will be activated with those input vectors (one after the other), where the decision is wrong, hence non zero adaptation is needed:

$$x(j) \in X^{+}: \quad w^{T}(j)x(j) < 0, \quad y = -1, \; d = 1$$

$$x(j) \in \tilde{X}^{-}: \quad w^{T}(j)x(j) < 0, \quad y = -1, \; d = 1$$

  - Note: The error function is always positive ( $\varepsilon = 2$ )

# Perceptron Convergence theorem (3)

- According to the learning method:
- $w(n+1)=w(0)+2\eta x(0)+2\eta x(1)+2\eta x(2)+2\eta x(3)+\ldots+2\eta x(n)$
  - where

$$x(j)\in X^{+}:\quad w^{T}(j)x(j)<0,\quad y=-1,\ d=1$$

or

$$x(j)\in \tilde{X}^{-}:\quad w^{T}(j)x(j)<0,\quad y=-1,\ d=1$$

  - The decision boundary will be:

$$2\eta w^{T}x=0$$

*which means that $\eta$ is a scaling factor, therefore it can be choosen for any positive number.*

*Let us use* $\eta=1/2$, *therefore* $\eta\varepsilon=1$

# Perceptron Convergence theorem (4)

- We will calculate $\|w(n+1)\|^2$ in two ways, and give an upper and a lower boundary, and it will turn out that an $n_{max}$ exists, and beyond that the lower boundary is higher than the upper boundary (squeeze theorem, sandwitch lemma (*közrefogási elv, rendőr elv*))

# Perceptron Convergence theorem (5)
# lower limit (1)

According to the learning method, the presented input vectors are added up:

$$w(n+1) = w(0) + x(0) + x(1) + \ldots + x(n)$$

w(0)=0

Multiply it with $w_o^T$ from the left:

$$w_o^T w(n+1) = w_o^T x(0) + w_o^T x(1) + \ldots + w_o^T x(n)$$

$$0 < \alpha \le w_o^T x(j)$$

Because each input vector (or its opposite) were selected that way.

$$0 < \alpha = \min_{x(n) \in \{X^+, \tilde{X}^-\}} w_o^T x(n)$$

$$w_o^T w(n+1) \ge n\alpha$$

# Perceptron Convergence theorem (6)
# lower limit (2)

$$w_o^T w(n+1) \geq n\alpha$$

We apply Cauchy Schwarty inequality $\|a\|^2 \|b\|^2 \geq \|a^T b\|^2$

$$\|w_0^T\|^2 \|w(n+1)\|^2 \geq \|w_o^T w(n+1)\|^2 \geq n^2\alpha^2$$

Lower limit:

$$\|w(n+1)\|^2 \geq \frac{n^2\alpha^2}{\|w_0^T\|^2}$$

Lower limit proportional with $n^2$

# Perceptron Convergence theorem (7) upper limit (1)

Let us have a different synthetization approach of *w(n+1)*:

$$w(k+1) = w(k) + x(k)$$

for k= 0 … n

Squared Euclidian norm:

$$\|w(k+1)\|^2 = \|w(k)\|^2 + \|x(k)\|^2 + 2w(k)^T x(k)$$

$$w(k)^T x(k) < 0$$

Because each input vector (or its opposite) were selected that way.

$$\|w(k+1)\|^2 \leq \|w(k)\|^2 + \|x(k)\|^2$$

$$\|w(k+1)\|^2 - \|w(k)\|^2 \leq \|x(k)\|^2$$

for k= 0 … n

# Perceptron Convergence theorem (8)
# upper limit (2)

$$\|w(k+1)\|^2 - \|w(k)\|^2 \le \|x(k)\|^2 \qquad \|w(k+1)\|^2 = \|w(k)\|^2 + \|x(k)\|^2 + 2w(k)^T x(k)$$

Calculating w(n+1) by adding up:

$$\sum_{k=0}^{n} \left( \|w(k+1)\|^2 - \|w(k)\|^2 \right) \le \sum_{k=0}^{n} \|x(k)\|^2$$

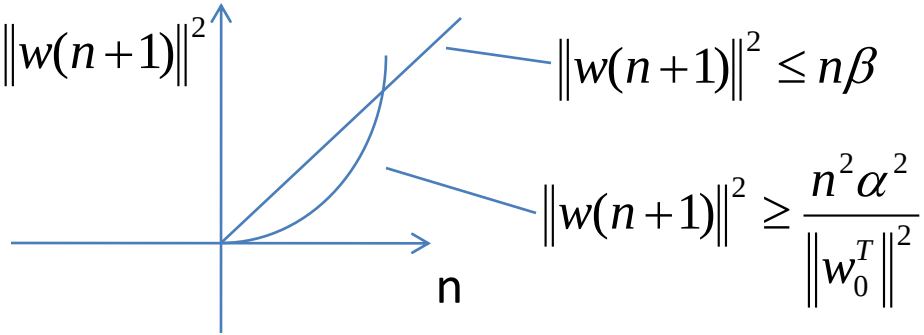
Note that there is a telescopic sum in the left hand side.

$$w(n+1) - w(0) = w(n+1) \le \sum_{k=0}^{n} \|x(k)\|^2 \qquad 0 < \beta = \max_{x(k) \in \{X^+, \tilde{X}^-\}} \|w_o^T x(k)\|^2$$

$$\|w(n+1)\|^2 \le n\beta$$

Upper limit linearly proportional with n

# Perceptron Convergence theorem (9) comparing upper and lower limits




Linear upper limit and squared lower limit cannot grow unlimitedly

$n_{max}$ should exist

$$n_{\max} = \frac{\beta \|w_0\|^2}{\alpha^2}$$