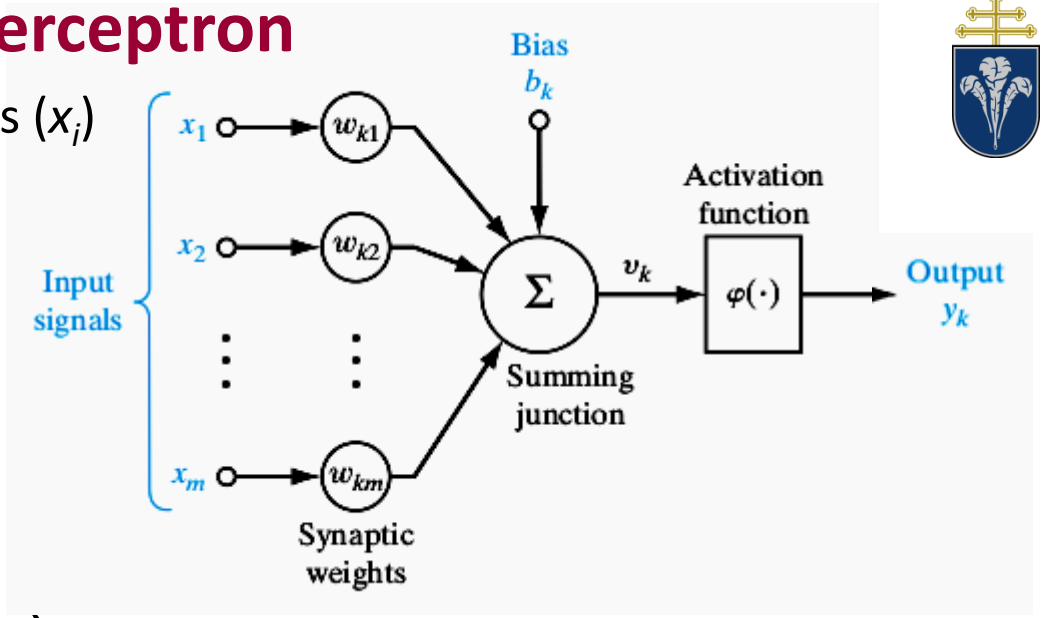
Outline

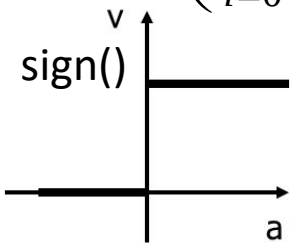
- Properties of the perceptron
- Input-output pairs
- Perceptron learning method
- Perceptron learning example
- Proof of convergence
- Good material:
  http://hagan.okstate.edu/4_Perceptron.pdf

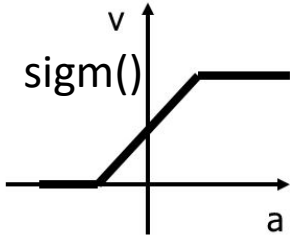
The Perceptron

- Receives input through its synapsis (x_i)
- Synapsis are weighted (w_i)
- A *b* value biases the sum to enable asymmetric behavior
- A weighted sum is calculated
- Activation function applied



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

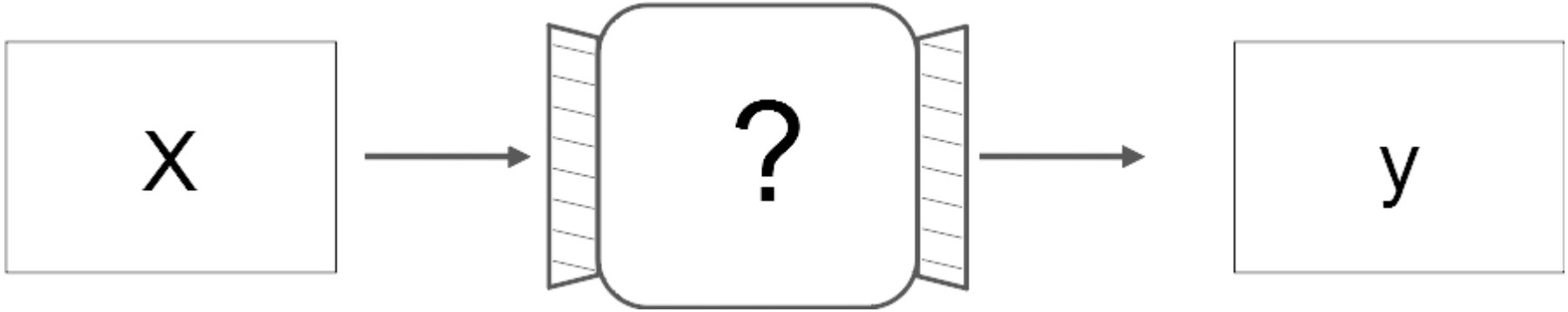




x_i : input vector
w_{ki} : weight coefficient vector of neuron k
b_k : bias value of neuron k
o_k : output value of neuron k

Neural Networks

Perceptron is an Input → Output device



As opposed to **Traditional Computers** where
- the math of the functionality is known
- the known math should be programmed

At **Neural Networks**
- the math behind the functionality is unknown
- the functionality is “illustrated” with examples

Function illustrated by examples

- Given a set of input-output pairs

  $x_j \rightarrow d_j$ (x_j: input vector; d_j: desired output)

	X		d
#	X_1	X_2	
1	0	0	0
2	0	1	0
3	1	0	0
4	1	1	1

- Number of input vectors
  - Finite/limited set (*e.g.* AND function)
    - Equivalent with a look-up-table (LUT), math known
    - Mathematically it is correct to define a function by listing all the IO pairs
    - Goal: generate a simpler than LUT decision making device through learning
  - Infinite/open set (customers of a bank asking for a loan)
    - Math behind is unknown, cannot be coded directly
    - Goal: generate the function through learning
    - It should predict well the output of a previously unknown/untested input (**GENERALIZATION**)

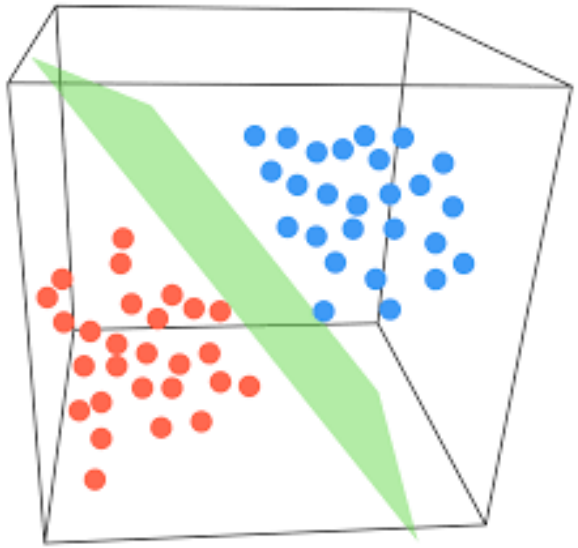
	X				d
#	age	gender	debt	salary	
1	25	M(1)	25	100	Y(1)
2	22	F(2)	18	80	Y(1)
3	65	M(1)	3000	200	N(0)
.	.	.	.	.	.
.	.	.	.	.	.

Good news: we can use the same learning/training method!!!

Linear separability

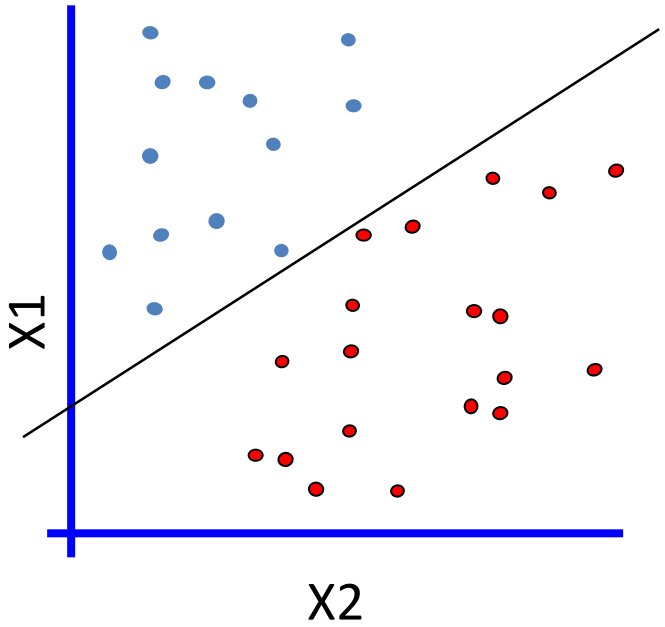
- Today, we assume that the IO sets are linearly separable
- The decision boundary is a hyperplane defined:

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



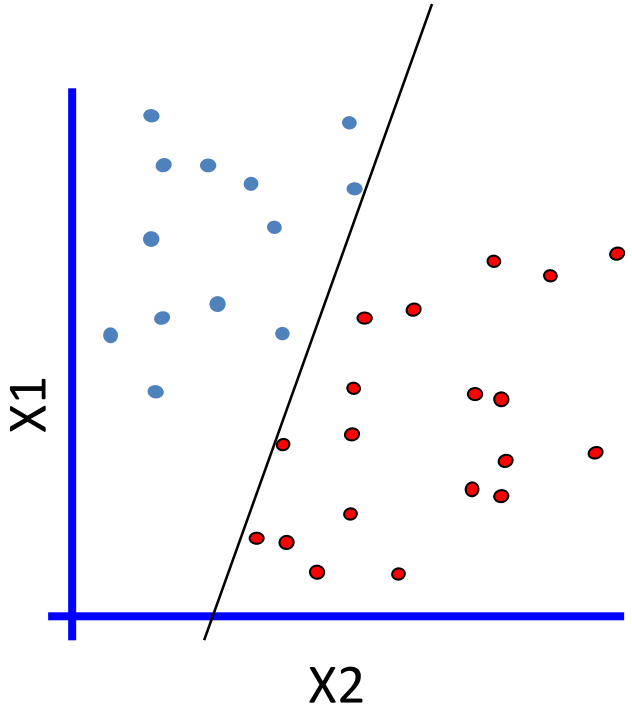
- Positive side of the hyperplane is classified: +1 (yes)
- Negative side of the hyperplane is classified : 0 (no).

Which boundary surface to use, if there are many?



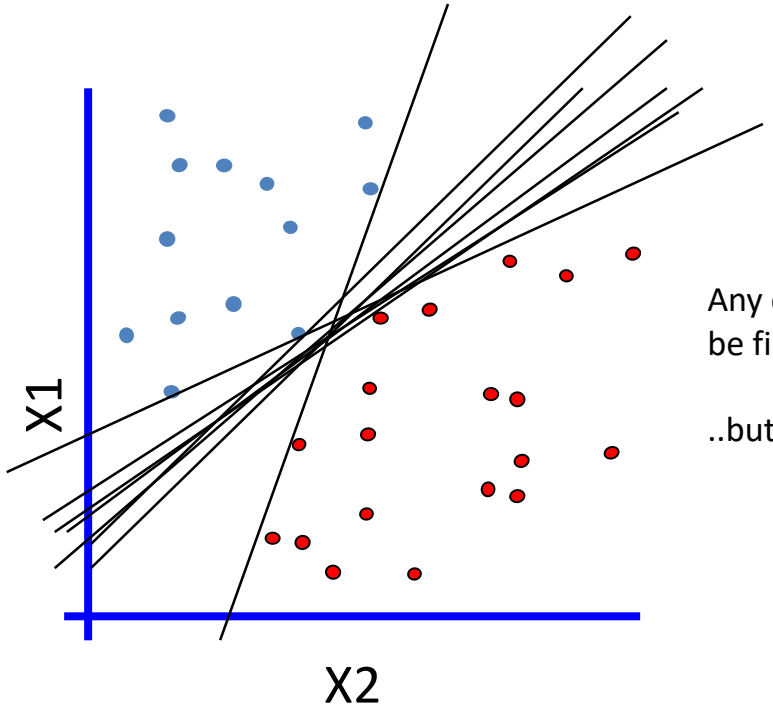
How would you classify this data?

Which boundary surface to use, if there are many?



How would you classify this data?

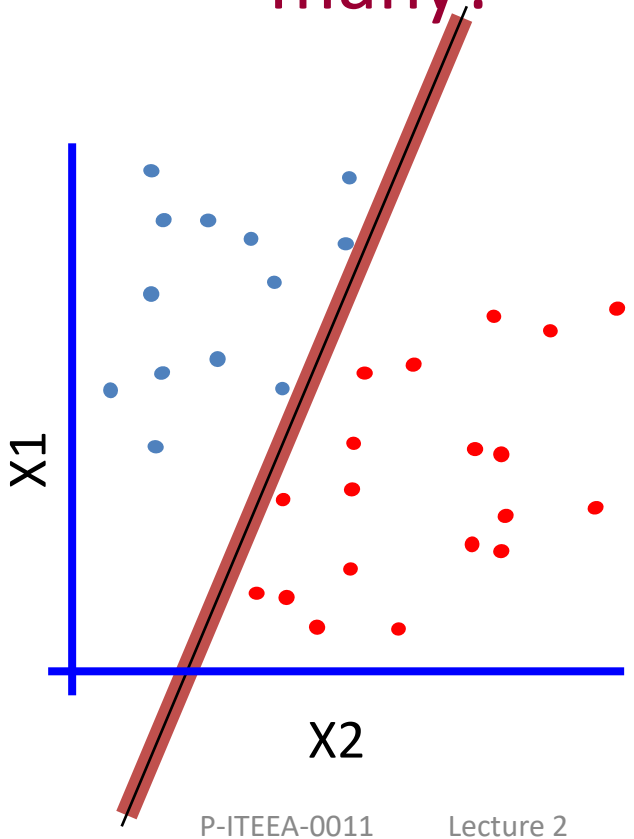
Which boundary surface to use, if there are many?



Any of these would be fine..

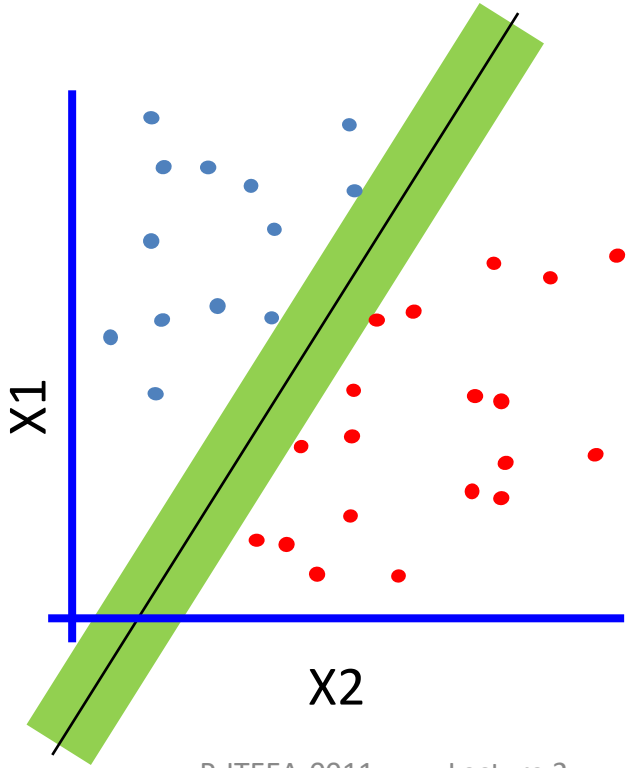
..but which is best?

Which boundary surface to use, if there are many?



Maximum Margin:
Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a data point.

Which boundary surface to use, if there are many?



Maximum Margin:
Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a data point.

What does learning mean?

- Given an annotated dataset

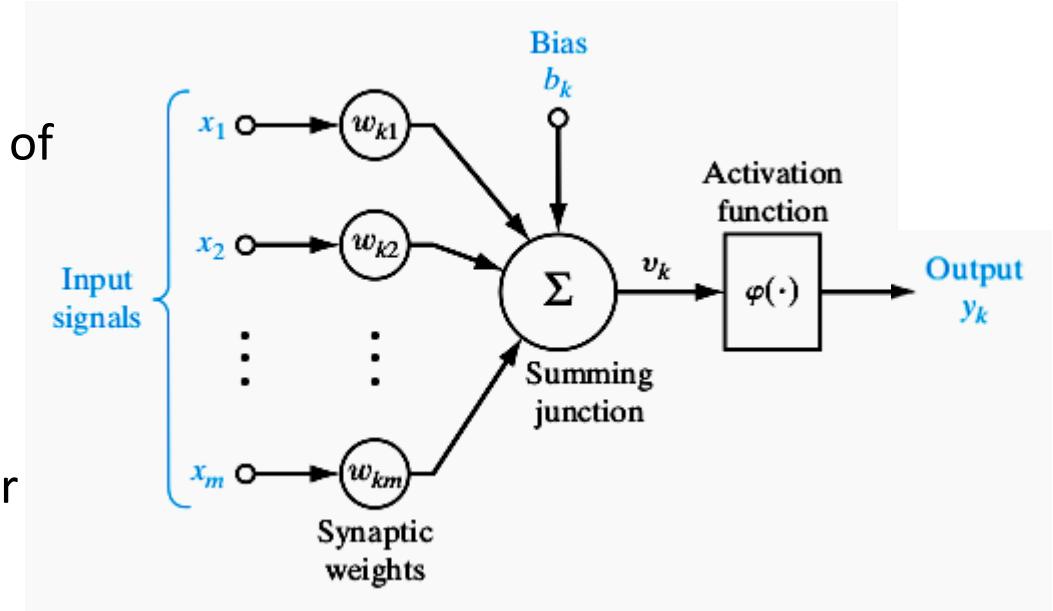
  $\mathbf{x}_j \rightarrow d_j$

- Given the parametric equation of the perceptron

  $y = sign(\mathbf{w}^T\mathbf{x})$

- Goal: find the optimal $\mathbf{w}_{opt}$ weights (parameters), where for each j

  $d_j = sign(\mathbf{w}_{opt}^T\mathbf{x}_j)$



The learning algorithm: Datasets

- Training set
  - *Set of input – desired output pairs*
  - *Will be used for training*

$$X^+ = \{\mathbf{x} : \ d = +1\}$$
$$X^- = \{\mathbf{x} : \ d = 0\}$$

- Test set
  - Used, when we have large set of input vectors (not used today)
  - *Set of input – desired output pairs*
  - *Will be used for testing and scoring the result*

- We assumed that X^+ and X^- must be linearly separable
- We are looking for an optimal parameter set:

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

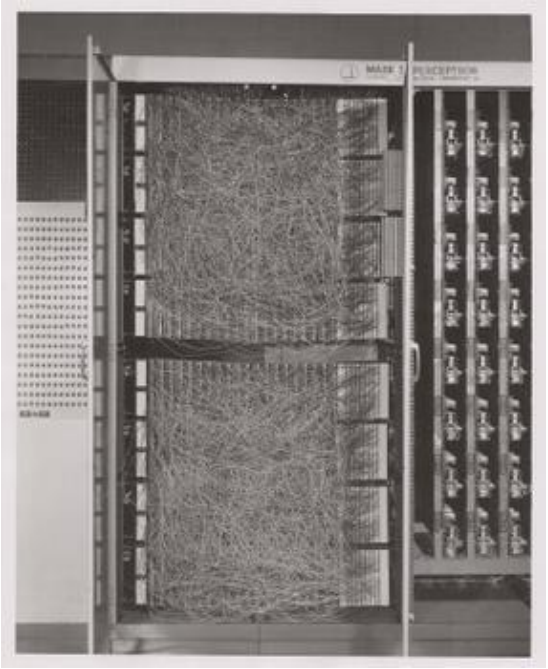
The learning algorithm: Recursive algorithm

- We have to develop a recursive algorithm called learning, which can learn the weight step by step, based on observing
  - the **(i) input,**
  - the **(ii) weight vector**,
  - the **(iii) desired output,** and
  - the **(iv) actual output** of the system.
- This can be described formally as follows:

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

The learning algorithm: Perceptron Learning Algorithm

- In a more ambitious way it can be called intelligent, because
  - perceptron can learn through examples (adapt),
  - even the function parameters are fully hidden.
- Perceptron learning was introduced by Frank Rosenblatt 1958
  - Built a 20x20 image sensor
  - With analog perceptron
  - 400 weights controlled by electromotors



The learning algorithm: Recursive steps

1. *Initialization.*

   Set **w**(0)=**0** or **w**(0)=**rand**

2. *Activation.*

   Select a $\mathbf{x}_k$ → d_k pair

3. *Computation of actual response*

   $$y(k) = sign\left(w^T(k)x(k)\right)$$

4. *Adaptation of the weight vector*

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

5. *Continuation*

   Until all responses of the perceptron are OK

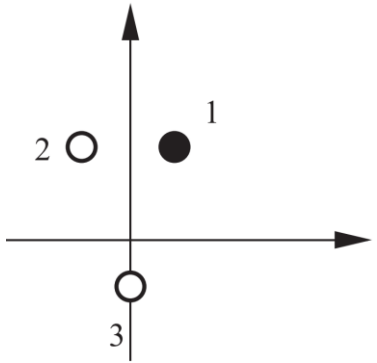
Weight update: very simple example

- Given a 3 input vector example
- Assume that bias is zero
  (decision boundary will cross the origo)
- Random initialization

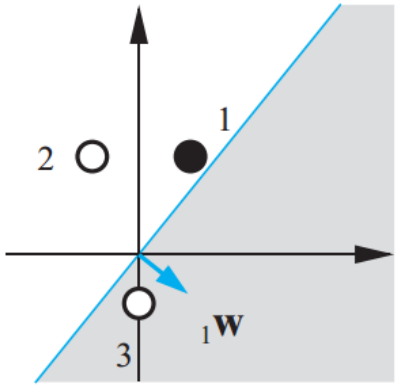
$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, d_1 = 1;$$

$$\mathbf{x}_2 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, d_2 = 0;$$

$$\mathbf{x}_3 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, d_3 = 0;$$



$$\mathbf{w}^T(1) = [1 \quad -0.8];$$



Remember: the weight vector is orthogonal to the decision boundary!!!
Decision boundary: $x_1 - 0.8x_2 = 0$
Its orthogonal vector is: (1, -0.8)

Weight update: very simple example

- Test with the first input vector $\mathbf{x}_1 = \begin{bmatrix}1\\2\end{bmatrix}, d_1 = 1;$

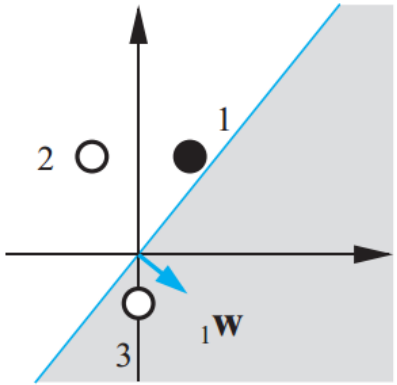
$\mathbf{w}^T(1) = [1 \quad -0.8];$

d_j-y_j > 0

$$y_1(1) = sign(w^T(1)x_1) = sign\left([1 \quad -0.8] \quad \begin{bmatrix}1\\2\end{bmatrix}\right) = sign(1 - 1.6) = 0$$

The result is not OK! Positive misclassification: Instead of 1, the result is 0!!

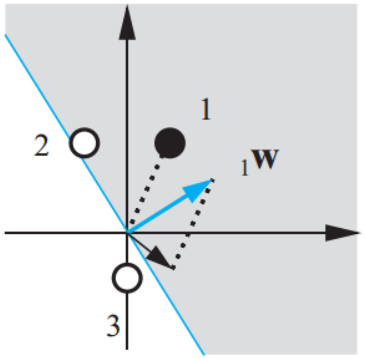
(The normal vector points to the positive side of the decision boundary.)



Idea: **add** the vector pointing to the positively misclassified point to the orthogonal vector of the decision boundary, to rotate it **towards** the point!

$w(k+1)=w(k)+x_1$

$$\mathbf{w}^T(2) = [1+1 \quad -0.8+2] = [2 \quad 1.2];$$



Weight update: very simple example

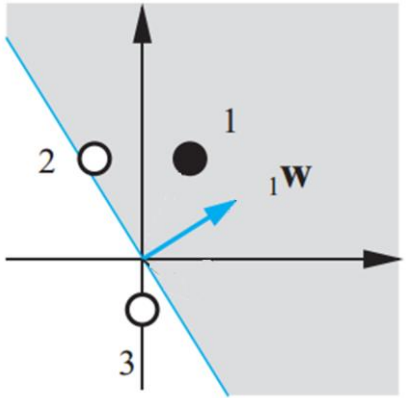
- Test with the second input vector $\mathbf{x}_2 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, d_1 = 0;$

$\mathbf{w}^T(2) = [2 \quad 1.2];$

d_j-y_j < 0

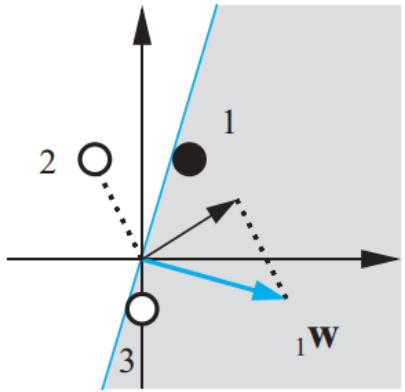
$$y_2(2) = sign(w^T(2)x_2) = sign\left([2 \quad 1.2] \quad \begin{bmatrix} -1 \\ 2 \end{bmatrix}\right) = sign(-2 + 2.4) = 1$$

The result is not OK! Negative misclassification: Instead of 0, the result is 1!!



Idea: **subtract** the vector pointing to the negatively misclassified point to the orthogonal vector of the decision boundary, to rotate it **away** the point!
$w(k+2)=w(k+1)-x_2$

$$\mathbf{w}^T(3) = [2 - (-1) \quad 1.2 - 2] = [3 \quad -0.8]$$



Weight update: very simple example

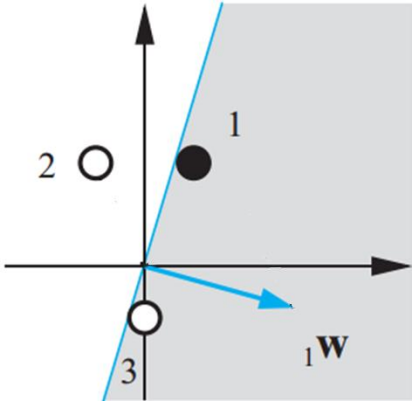
- Test with the third input vector $\mathbf{x}_3 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, d_3 = 0;$

d_j-y_j < 0

$\mathbf{w}^T(3) = [3 \quad -0.8];$

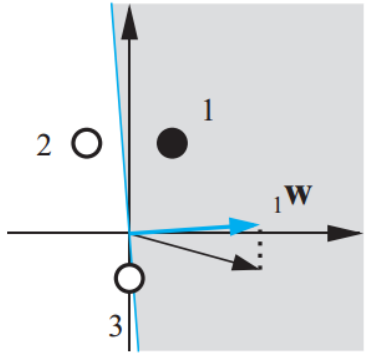
$$y_3(3) = sign(w^T(3)x_3) = sign\left([3 \quad -0.8] \quad \begin{bmatrix} 0 \\ -1 \end{bmatrix}\right) = sign(0 + 0.8) = 1$$

The result is not OK! Negative misclassification: Instead of 0, the result is 1!!



Again: **subtract** the vector pointing to the negatively misclassified point to the orthogonal vector of the decision boundary, to rotate it **away** the point!
$w(k+3)=w(k+2)-x_3$

$$\mathbf{w}^T(4) = [3 - 0 \quad -0.8 - (-1)] = [3 \quad 0.2];$$



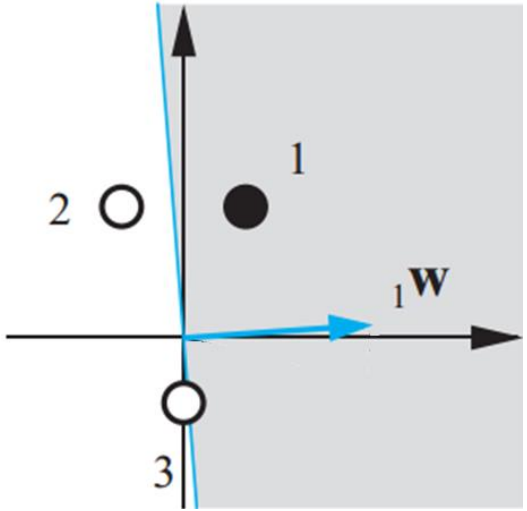
Weight update: very simple example

- Start again:
  - Test with the again with the first vector

    The result is OK!
    - Do not modify!!!
  - Test with the again with the second vector

    The result is OK!
    - Do not modify!!!
  - Test with the again with the third vector

    The result is OK!
    - Do not modify!!!
- Since all input vectors are correctly classified: we are ready



Formalization of the update rules

- Positive misclassification : ADD

  $\varepsilon = d_j - y_j = 1$ $\quad w(k+1)=w(k)+x_j$

- Negative misclassification : SUBTRACT

  $\varepsilon = d_j - y_j = -1$ $\quad w(k+1)=w(k)-x_j$

- Correct classification : DO NOTHING

  $\varepsilon = d_j - y_j = 0$ $\quad w(k+1)=w(k)$

- In general:

$$w(k+1)=w(k)+\varepsilon\, x_j$$

The learning algorithm: Adaptation

We were looking for a recursive function:

$$\mathbf{w}(k+1) = \Psi\big(\mathbf{x}(k), \mathbf{w}(k), d(k), y(k)\big)$$

In general:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \varepsilon\eta\boldsymbol{x}_j$$

where is the error function

$$\varepsilon(k) = d(k) - y(k)$$

and

$$d(k) = \begin{cases} 0 & \text{if } \mathbf{x}(k) \text{ belongs to class } \mathrm{X}^{+} \\ -1 & \text{if } \mathbf{x}(k) \text{ belongs to class } \mathrm{X}^{-} \end{cases},$$

η is the learning rate
(η controls the learning speed and should be positive)

AND

Weight update strategy

- Apply all the input vectors in one after the others, selecting them randomly
- Instance update
  - Update the weights after each input
- Batch update
  - Add up the modifications
  - Update the weights with the sum of the modifications, after all the inputs were applied
- Mini batch
  - Select a smaller batch of input vectors, and do with that as in the batch mode

Perceptron Convergence theorem (1)

Assumptions:

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

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

- *Let us denote* $\tilde{x} = -x$

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

For the proof, see also: Simon Haykins: Neural Networks and Learning Machines, *Section 1.3:* http://dai.fmph.uniba.sk/courses/NN/haykin.neural-networks.3ed.2009.pdf

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:

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

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

  - Note: The error function is always positive ($\varepsilon = 1$)

Perceptron Convergence theorem (3)

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

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

which means that η is a scaling factor, therefore it can be choosen for any positive number.

Let us use $\eta=1$, *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)

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

Summing up the upper term:

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

Telescoping sum: $\sum_{i=1}^{n}(a_{i+1}-a_i) = a_{n+1} - a_1$

Example: $\sum_{i=1}^{4}(a_{i+1}-a_i) = \cancel{a_2} - a_1 +$
$+\cancel{a_3} - \cancel{a_2} +$
$+\cancel{a_4} - \cancel{a_3} +$
$+a_5 - \cancel{a_4} =$
$= \boldsymbol{a_5 - a_1}$

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

$\|w(n+1)\|^2 - \|w(0)\|^2$:

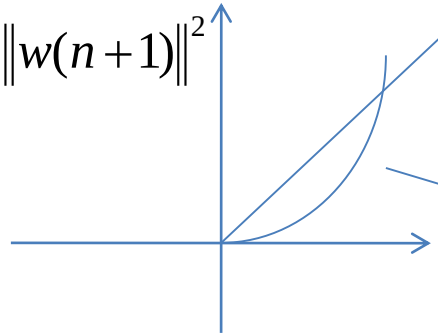
$$0 < \beta = \max_{x(k)\in\{X^+,\tilde{X}^-\}}\|x(k)\|^2$$

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

$\|w(0)\|^2 = 0$

Upper limit linearly proportional with n

Perceptron Convergence theorem (9) comparing upper and lower limits



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

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

Linear upper limit and squared lower limit cannot grow unlimitedly

n_{max} should exist

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