WARSAW UNIVERSITY OF TECHNOLOGY
FACULTY OF MATHEMATICS
AND INFORMATION SCIENCE

Neural Networks

Lecture 5

The Perceptron

The Perceptron

In 1962 Frank Rosenblatt introduced the new idea of the perceptron.

General idea: a neuron learns on its mistakes!!

If the element output signal is wrong – the changes are to minimize the possibilities that such the mistake will be repeated.

If the element output signal is correct – there are no changes.

The Perceptron

The one layer perceptron is based on the McCulloch & Pitts threshold element. The simplest perceptron - Mark 1 – is composed from four types of elements:

- layer of input elements, (square grid of 400 receptors), *elements type* ***S*** receiving stimuli from the environment and transforming those stimuli into electrical signals
- associative elements, *elements type* ***A****,* threshold adding elements with excitatory and inhibitory inputs

The Perceptron

- output layer – *elements type **R**,* the reacting elements, randomly connected with the ***A*** elements, set of ***A*** elements correspond with to each element, ***R*** passes to state 1 when its total input signal is greater than zero
- control units

Phase 1 - learning. At the beginning, e.g presentation of the representatives of the first class.
Phase 2 – verification of the learning results

Learning of the second class etc..

The Perceptron

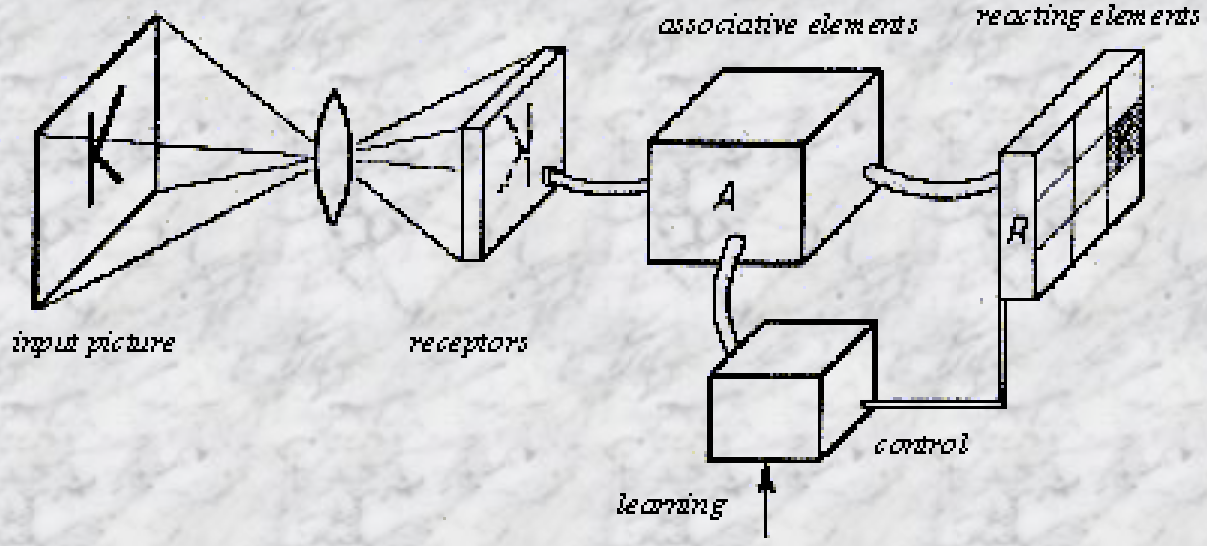
Mark 1:

400 threshold elements of the type ***S***; if they are enough excited – they produce at the output one the signal +1 and at the output two the signal -1.

The associative element ***A***, has 20 inputs, randomly (or not) connected with the ***S*** elements outputs (excitatory or inhibitory). In Mark 1 was 512 elements of type ***A***.

The ***A*** elements are randomly connected with the elements type ***R***. In Mark 1 was 8 elements of type ***R***.

The Perceptron



A block diagram of a perceptron. On the receptive layer the picture of the letter **K** is projected. As the result, in the reacting layer, the region corresponding to letter **K** (in black) is activated.

The Perceptron

Each element ***A*** obtain „weighted sum" of an input signal.
When the number of excitatory signals **>** than the number of inhibitory signals – at the output the +1 signal is generated.
When **<** there is no signal generation.
Elements ***R*** are reacting on the added input from the elements ***A***. When the input is **>** than the threshold – The +1 signal is generated, otherwise – signal 0.

Learning means changes in weights of active elements ***A***.

The Perceptron

Simplified version:

Two layers – input and output. Active is only the layer two. Input signals are equal 0 or +1. Such the structure is called **one layer perceptron**.

Elements (possibly only one) of the output layer obtain at their input the weighted signal from the input layer. If this signal is greater than the defined threshold value – the signal +1 is generated, otherwise the signal 0.

The learning method is based on the correction of weights connecting the input layer with the elements of the output layer. Only the active elements of the input layer are the subject of correction.

Weights modification rule

$$w_{iA}\,(\text{new}) = w_{iA}\,(\text{old}) - \text{input}_i$$

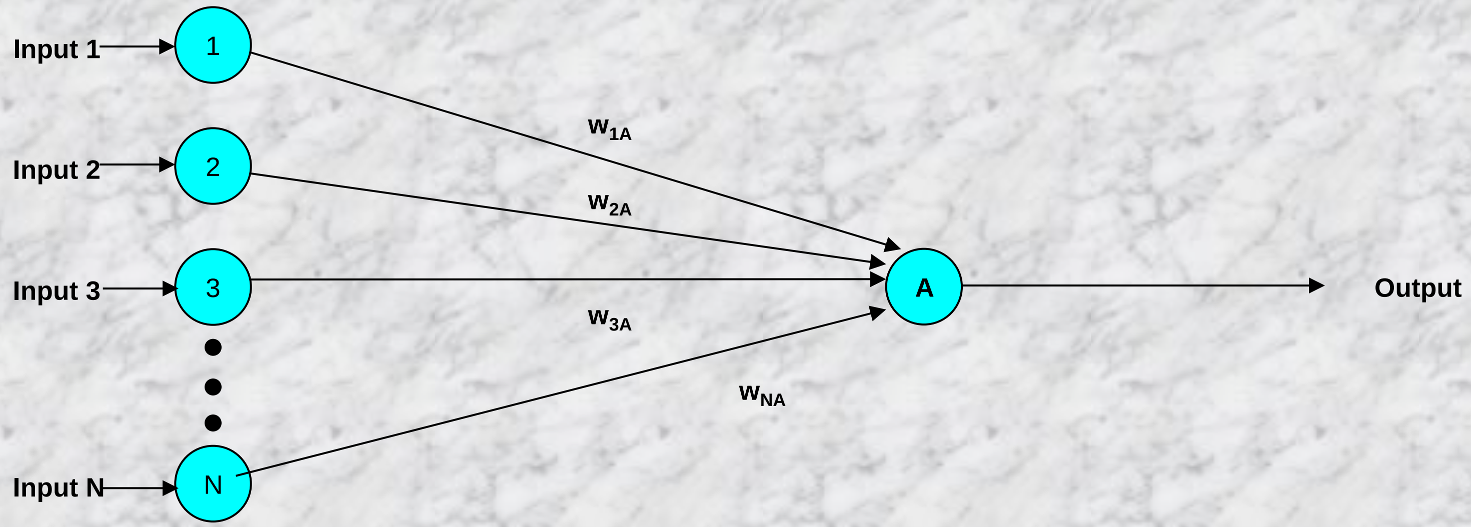
$$w_{iB}\,(\text{new}) = w_{iB}\,(\text{old}) + \text{input}_i$$

$$\text{input}_i = \pm 1$$

The Perceptron

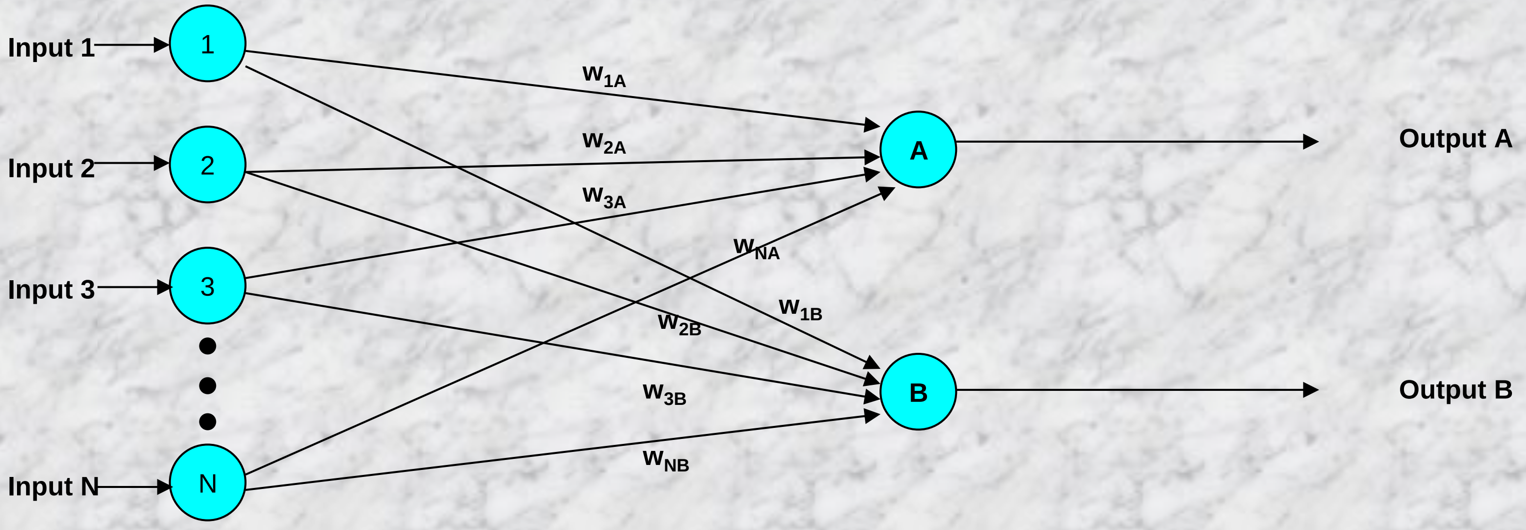
The example

The one-layer and two-elements Perceptron



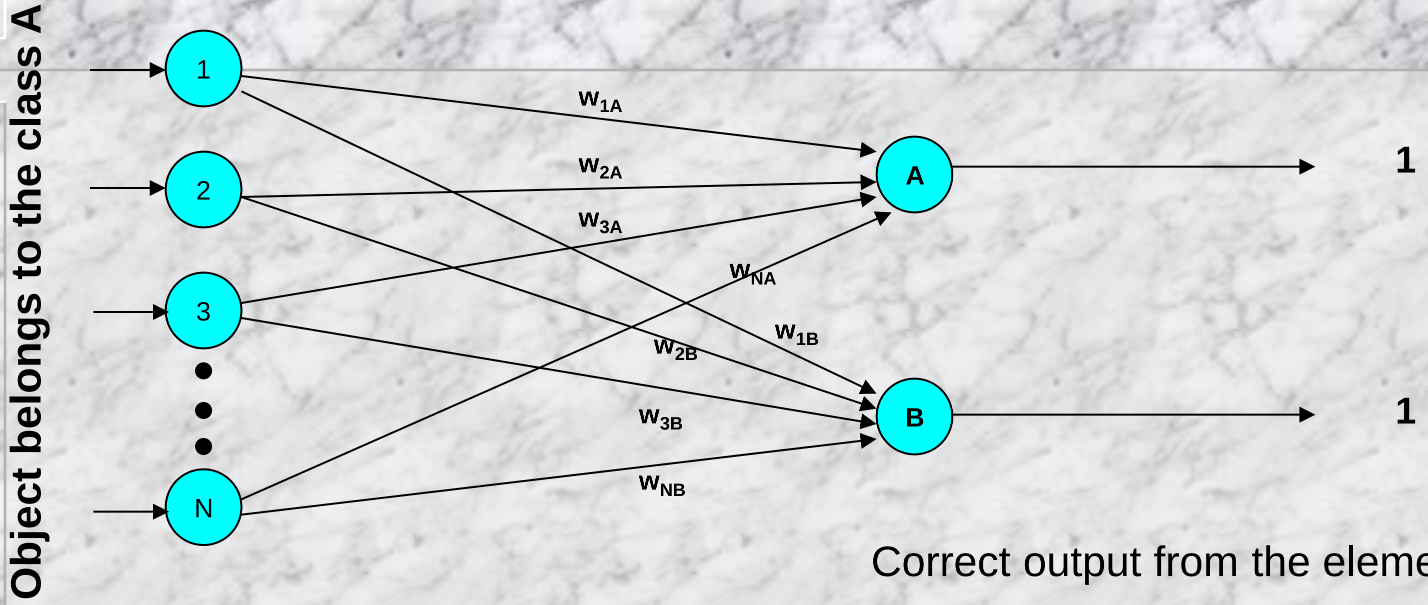
Input / Object	Output
Class A	1
Class B	0

The one-layer and two-elements Perceptron



Output / Object	Output A	Output B
Class A	1	0
Class B	0	1

Perceptron's learning



Output / Object	Output A	Output B
Class A	1	0
Class B	0	1

Correct output from the element A

We do not change the weights incoming to the element A, w_{iA}

Incorrect output from the element B (1 instead of 0)

Input signal to B ≥ threshold value

It is necessary to decrease the weights incoming to the element B w_{iB}

Weights modification rule

Assuming

- ζ = output (library) – output (real)

than

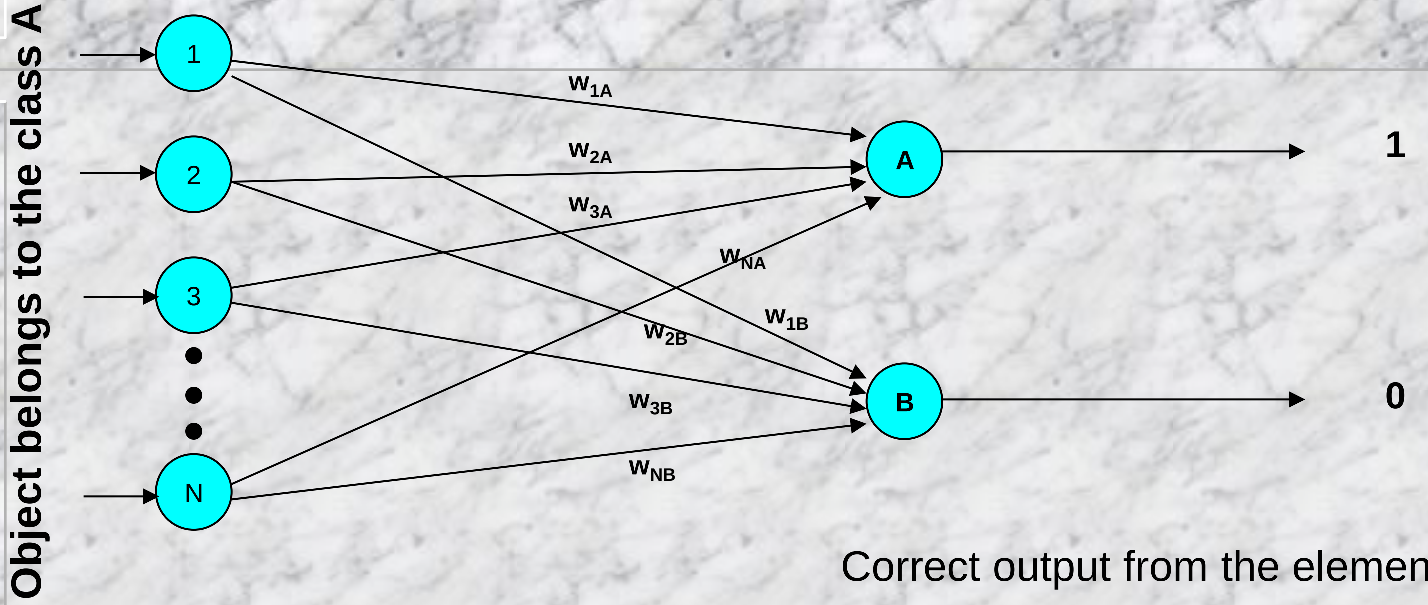
- $w_{iB}\,(\text{new}) = w_{iB}\,(\text{old}) + \Delta\, w_{iB}$

For example:

$$w_{iB}\,(\text{new}) = w_{iB}\,(\text{old}) + \zeta\ \text{Input}_i$$

$$\text{Input}_i = \begin{cases} 0 \\ +1 \end{cases}$$

Perceptron's learning



Output / Object	Output A	Output B
Class A	1	0
Class B	0	1

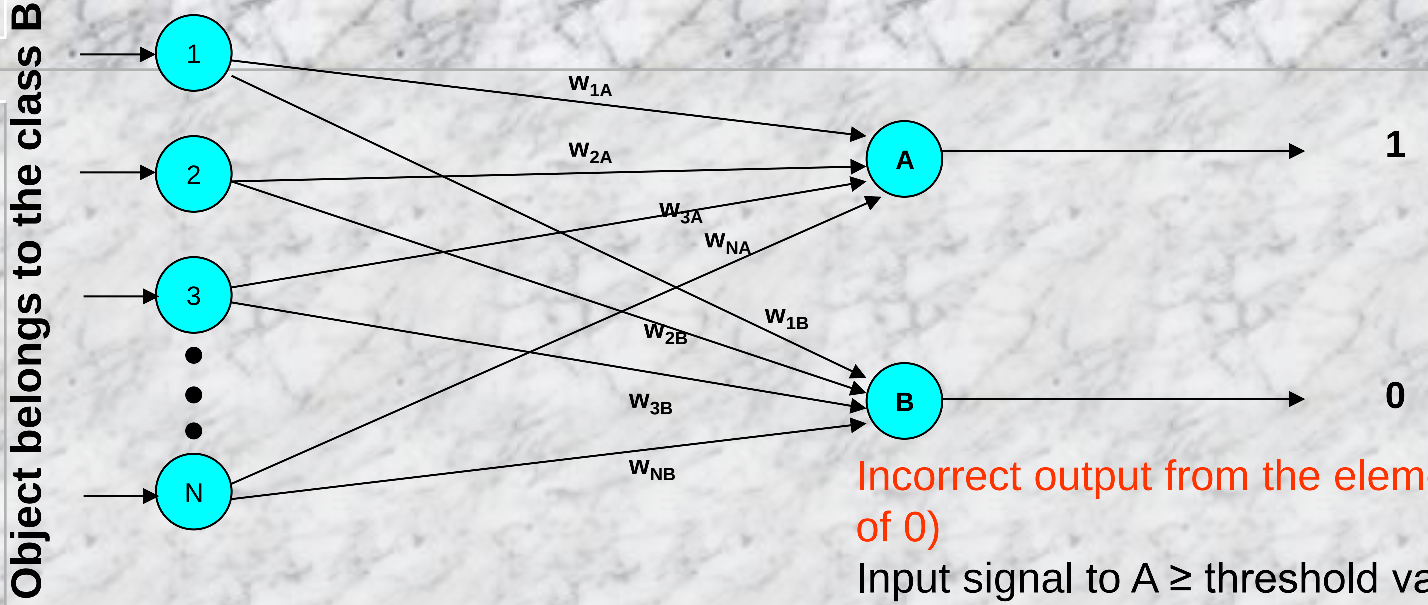
Correct output from the element A

We do not change the weights incoming to the element A, w_{iA}

Correct output from the element B

We do not change the weights incoming to the element B, w_{iB}

Perceptron's learning



Output / Object	Output A	Output B
Class A	1	0
Class B	0	1

Incorrect output from the element A (1 instead of 0)

Input signal to A ≥ threshold value

It is necessary to decrease the weights incoming to the element A, w_{iA}

Incorrect output from the element B (0 instead of 1)

Input signal to B < threshold value

It is necessary to increase the weights incoming to the element B, w_{iB}

The Perceptron learning algorithm

The perceptron learning algorithm

It can be proved that:

„ ... given it is possible to classify a series of inputs, ... then a perceptron network will find this classification”.

another words

„a perceptron will learn the solution, if there is a solution to be found”

Unfortunately, such the solution not always exists !!!

The perceptron learning algorithm

It is important to distinguish between the representation and learning.

- **Representation** refers to the ability of a perceptron (or any other network) to simulate a specified function.
- **Learning** requires the existence of a systematic procedure for adjusting the network weights to produce that function.

The perceptron learning algorithm

This problem was used to illustrate the weakness of the perceptron by Minsky and Papert in 1969:

They showed that some perceptrons were impractical or inadequate to solve many problems and stated there was no underlying mathematical theory to perceptrons.

The perceptron learning algorithm

Bernard Widrow recalls: *„..my impression was that Minsky and Papert defined the perceptron narrowly enough that it couldn't do anything interesting. You can easily design something to overcome many of the things that they proved' couldn't be done. It looked like an attempt to show that the perceptron was no good. It wasn't fair."*

XOR Problem

One of Minsky's and Papert more discouraging results shows that a single-layer perceptron cannot simulate a simple but very important function

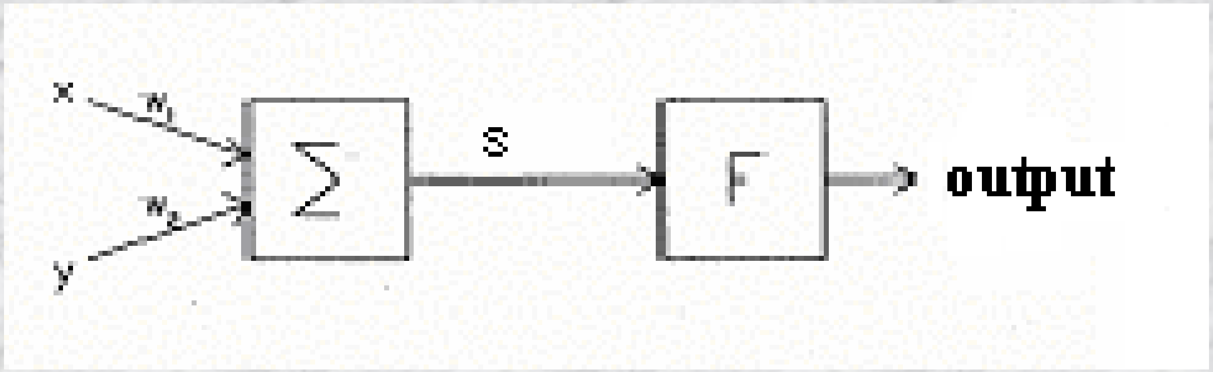
the exclusive-or (XOR)

XOR Problem

XOR truth table

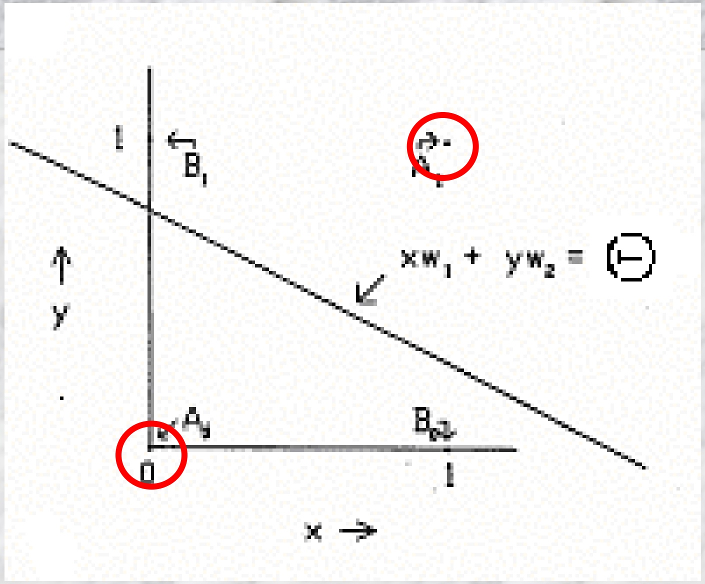
x	y	output	point
0	0	0	A_0
1	0	1	B_0
0	1	1	B_1
1	1	0	A_1

XOR Problem



Function **F** is the simple threshold function producing at the output **signal 0 (zero)** when signal **s** jest below Θ , and signal **1 (one)** when signal **s** is greater (or equal) Θ .

XOR Problem



$$xw_1 + yw_2 = \Theta$$

Does not exist the system of values of w_1 i w_2, that points A_0 i A_1 will be located on one side, and B_0 i B_1 on the other side of this straight line.

Finally, what the perceptron really is ??

- **Question**, is it possible to realize every logical function by means of a single neuronal element with properly selected parameters??
- Is it possible to built every digital system by means of the neuronal elements??
- Unfortunately, there exist functions where it is necessary to use two or more elements.
- It is easy to demonstrate, that it is impossible to realize any function of N variables by means of single neuronal element.

Finally, what the perceptron really is??

Geometrical interpretation of the equation

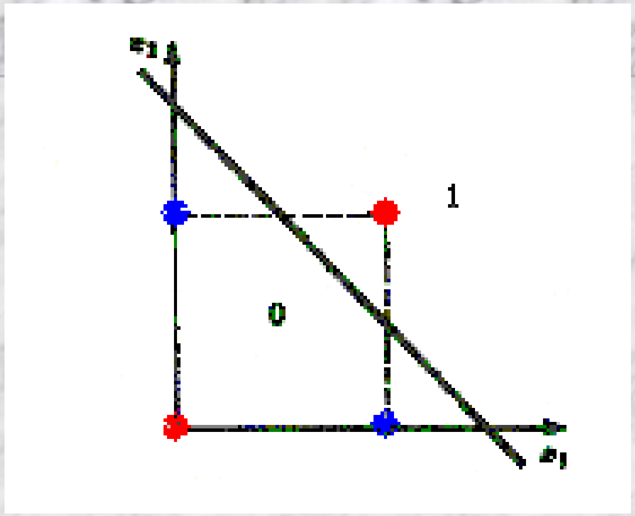
$$\sum_i w_i(t)x_i(t) = \Theta$$

is a plane (surface), which orientation depends from the weights..

The plane should be orientated in such the way all vertices, where **output = 1** where located on the same side, i.e. the inequality will be fulfilled

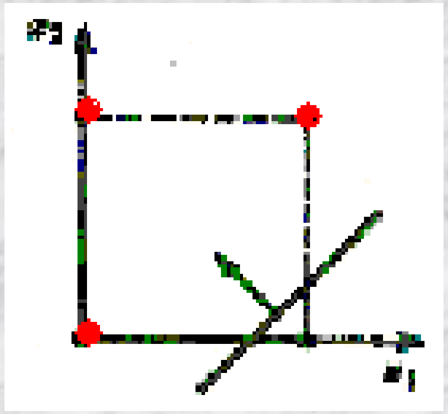
$$\sum_i w_i(t)x_i(t) \geq \Theta$$

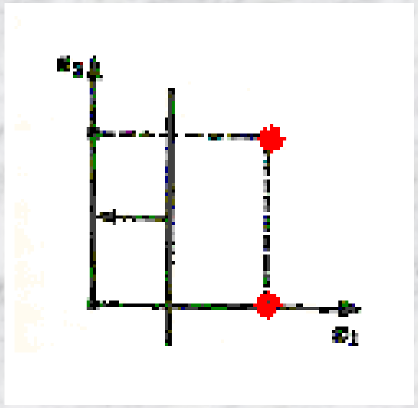
Finally, what the perceptron really is??

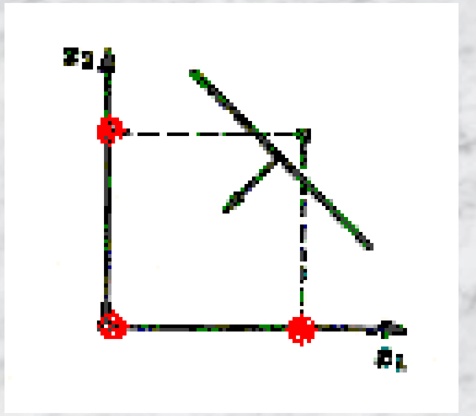


From the figure above it is easy to understand why realization of the **XOR** is impossible. Does not exist the single plane (for N=2 – straight line) separating points of different color.

Finally, what the perceptron really is??







On these figures is the difficulties with the realization demanding the negative threshold values (n.b. the biological interpretation is sometime doubtful).

Linear separability

The problem of linear separability impose limitations fro the use of one layer neural nets. So, it is very important to knowledge about this property.

The problem of linear separability can be solved by the increase of the number of the network layers.

The convergence of the learning procedure

The convergence of the learning procedure

- The input patterns are assumed to come from a space which has two classes: **F^+** and **F^-**.
- We want the perceptron to respond with **1** if the input comes from **F^+**, and **-1** if it comes from **F^-**.
- The set of input values x_i as a vector in n-dimensional space **X**, and the set of weights w_i as the another vector in the same space **W**.
- Increasing the weights is performed by **W + X**, and decreasing by **W - X**.

The convergence of the learning procedure

start: Choose any value for **W**

test: Choose an **X** from $\mathbf{F}^+$ or $\mathbf{F}^-$

If $\mathbf{X} \in \mathbf{F}^+$ i $\mathbf{W} \cdot \mathbf{X} > 0 \Rightarrow$ *test*

If $\mathbf{X} \in \mathbf{F}^+$ i $\mathbf{W} \cdot \mathbf{X} \leq 0 \Rightarrow$ *add*

If $\mathbf{X} \in \mathbf{F}^-$ i $\mathbf{W} \cdot \mathbf{X} < 0 \Rightarrow$ *test*

If $\mathbf{X} \in \mathbf{F}^-$ i $\mathbf{W} \cdot \mathbf{X} \geq 0 \Rightarrow$ *subtract*

add: Replace **W** by $\mathbf{W} + \mathbf{X} \Rightarrow$ *test*

subtract: Replace **W** by $\mathbf{W} - \mathbf{X} \Rightarrow$ *test*

Notice that we go to ***subtract*** when $\mathbf{X} \in \mathbf{F}^-$, and if we consider that going to ***subtract*** is the same as going to ***add*** **X** replaced by **–X.**

The convergence of the learning procedure

```
start:   Choose any value for $\mathbf{W}$

test:    Choose a $\mathbf{X}$ from $\mathbf{F}^+$ or $\mathbf{F}^-$
           If $\mathbf{X} \in \mathbf{F}^-$ change the sign of $\mathbf{X}$
           If $\mathbf{W} \cdot \mathbf{X} > 0 \Rightarrow$ test
              otherwise $\Rightarrow$ add

add:     Replace $\mathbf{W}$ by $\mathbf{W} + \mathbf{X} \Rightarrow$ test
```

We can simplify the algorithm still further, if we define **F** to be $\mathbf{F}^+ \cup -\mathbf{F}^-$ i.e. $\mathbf{F}^+$ and the negatives of $\mathbf{F}^-$.

The convergence of the learning procedure

```
start:  Choose any value for $\mathbf{W}$
test:   Choose any $\mathbf{X}$ from $\mathbf{F}$
            If $\mathbf{W} \cdot \mathbf{X} > 0 \Rightarrow$ test
            otherwise $\Rightarrow$ add
add:    Replace $\mathbf{W}$ by $\mathbf{W} + \mathbf{X} \Rightarrow$ test
```

The Perceptron

Theorem and proof

Theorem and proof

Convergence Theorem:

Program will only go to *add* a finite number of times.

Proof:

Assume that there is a unit vector **W*,** which partitions up the space, and a small positive fixed number δ, such that

$$\mathbf{W}^*\cdot\mathbf{X} > \delta \text{ for every } \mathbf{X} \in \mathbf{F}$$

Define $G(\mathbf{W}) = \mathbf{W}^*\cdot\mathbf{W}/|\mathbf{W}|$

and note that $G(\mathbf{W})$ is the cosine of the angle between **W** and **W*.**

Theorem and proof

Since $|\mathbf{W^*}| = 1$, we can say that $G(\mathbf{W}) \leq 1$.

Consider the behavior of $G(\mathbf{W})$ through *add.*

The numerator: $G(\mathbf{W}) = \dfrac{\mathbf{W^* W}}{|\mathbf{W}|}$

$\mathbf{W^*} \cdot \mathbf{W}_{t+1} = \mathbf{W^*} \cdot (\mathbf{W}_t + \mathbf{X}) = \mathbf{W^*} \cdot \mathbf{W}_t + \mathbf{W^*} \cdot \mathbf{X} \geq \mathbf{W^*} \cdot \mathbf{W}_t + \delta$

since $\mathbf{W^*} \cdot \mathbf{X} > \delta$.

Hence, after the m*th* application of *add* we have

$$\mathbf{W^*} \cdot \mathbf{W}_m \geq \delta \cdot m \qquad (1)$$

Theorem and proof

The denominator:

Since $\mathbf{W} \cdot \mathbf{X}$ must be negative (*add* operation is performed), then

$$| \mathbf{W}_{t+1} |^2 = \mathbf{W}_{t+1} \cdot \mathbf{W}_{t+1} = (\mathbf{W}_t + \mathbf{X}) \cdot (\mathbf{W}_t + \mathbf{X}) =$$

$$= | \mathbf{W}_t |^2 + 2\mathbf{W}_t \cdot \mathbf{X} + | \mathbf{X} |^2$$

Moreover $| \mathbf{X} | = 1$, so

$$| \mathbf{W}_{t+1} |^2 < | \mathbf{W}_t |^2 + 1,$$

and after m*th* application of *add*

$$| \mathbf{W}_m |^2 < m. \qquad (2)$$

Theorem and proof

Combining (1) i (2) gives

$$\mathbf{G(W_m) = \frac{(W * W_m)}{|W_m|} > \frac{m\delta}{\sqrt{m}}}$$

Because G(**W**) $\leq$ 1, so we can write

$$\mathbf{\sqrt{m} \cdot \delta \leq 1 \qquad i.e. \qquad m \leq \frac{1}{\delta^2}}$$

Theorem and proof

What does it mean?? Inequality (3) is our proof.

In the perceptron algorithm, we only go to *test* if $\mathbf{W}\cdot\mathbf{X} > 0$. We have chosen a small fixed number δ, such that $\mathbf{W}\cdot\mathbf{X} > \delta$. Inequality (3) then says that we can make δ as small as we like, but the number of times, m, that we go to *add* will still be ***finite,*** and will be $\leq 1/\delta^2$.

In other words, perceptron will learn a weight vector **W**, that partitions the space successfully, so that patterns from F^+ are responded to with a positive output and patterns from F^- produce a negative output.

The perceptron learning algorithm

The perceptron learning algorithm

<u>**1 step – initialize weight and threshold:**</u>

Define $w_i(t)$, $(i=0,1,\ldots,n)$ to be the weight from input *i* at time *t,* and Θ to be a threshold value ij the output node. Set $w_i(0)$ to small random numbers.

<u>**2 step – present input and desired output:**</u>

Present input $X = [x_1, x_2, \ldots, x_n]$, $x_i \in \{0,1\}$, and to the comparison block desired output d(t).

The perceptron learning algorithm

3 step:

Calculate actual output

$$y(t) = f\left[\sum_i w_i(t)x_i(t)\right]$$

4 step:

Adapt weights

The perceptron learning algorithm

4 step (cont):

if $y(t) = d(t)$ $\Rightarrow w_i(t+1) = w_i(t)$

if $y(t) = 0$ and $d(t) = 1 \Rightarrow w_i(t+1) = w_i(t) + x_i(t)$

if $y(t) = 1$ and $d(t) = 0 \Rightarrow w_i(t+1) = w_i(t) - x_i(t)$

The perceptron learning algorithm

Algorithm modifications

4 step (cont.):

if $y(t) = d(t)$ $\Rightarrow$ $w_i(t+1) = w_i(t)$

if $y(t) = 0$ and $d(t) = 1$ $\Rightarrow$ $w_i(t+1) = w_i(t) + \eta \cdot x_i(t)$

if $y(t) = 1$ and $d(t) = 0$ $\Rightarrow$ $w_i(t+1) = w_i(t) - \eta \cdot x_i(t)$

$0 \le \eta \le 1$, a positive gain term that controls the adaptation rate.

The perceptron learning algorithm

<u>Widrow and Hoff modification</u>

<u>4 step (cont.):</u>

if $y(t) = d(t)$ $\Rightarrow$ $w_i(t+1) = w_i(t)$

if $y(t) \neq d(t)$ $\Rightarrow$ $w_i(t+1) = w_i(t) + \eta \cdot \Delta \cdot x_i(t)$

$0 \leq \eta \leq 1$ a positive gain term that controls the adaptation rate.

$\Delta = d(t) - y(t)$

The perceptron learning algorithm

The Widrow-Hoff delta rule calculates the difference between the weighted sum and the required output, and calls that the **error**.

This means that during the learning process, the output from the unit is not passed through the step function – however, actual classification is effected by using the step function to produce the +1 or 0.

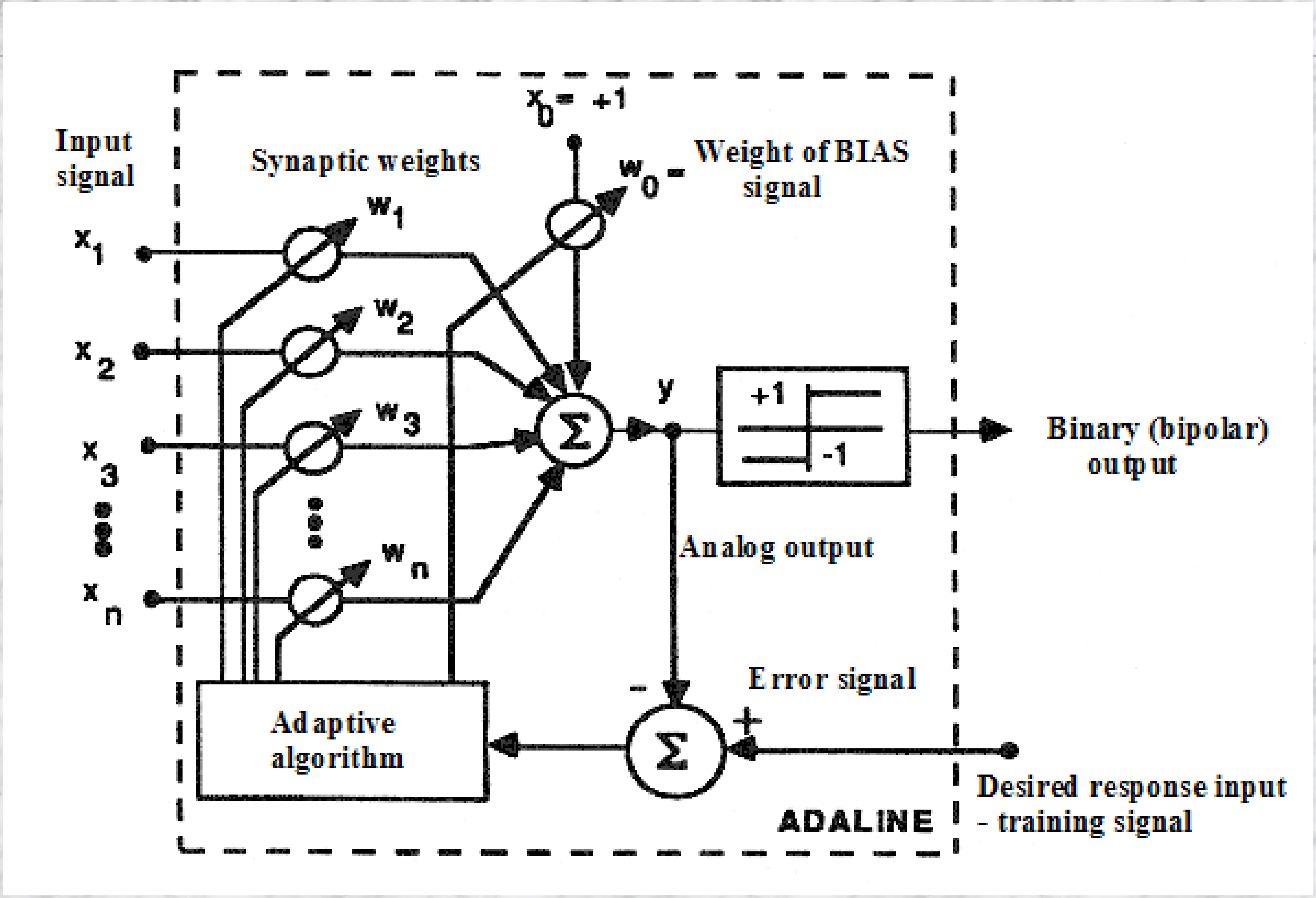
Neuron units using this learning algorithm were called ADALINE*s* (ADAptive LInear NEurons), who are also connected into a many ADALINE, or MADALINE structure.

Model ADALINE

Widrow and Hoff model

The structure ADALINE, and the way how it performs a weighted sum of inputs is similar to the single perceptron unit and has similar limitations. (for example the XOR problem).

Widrow and Hoff model

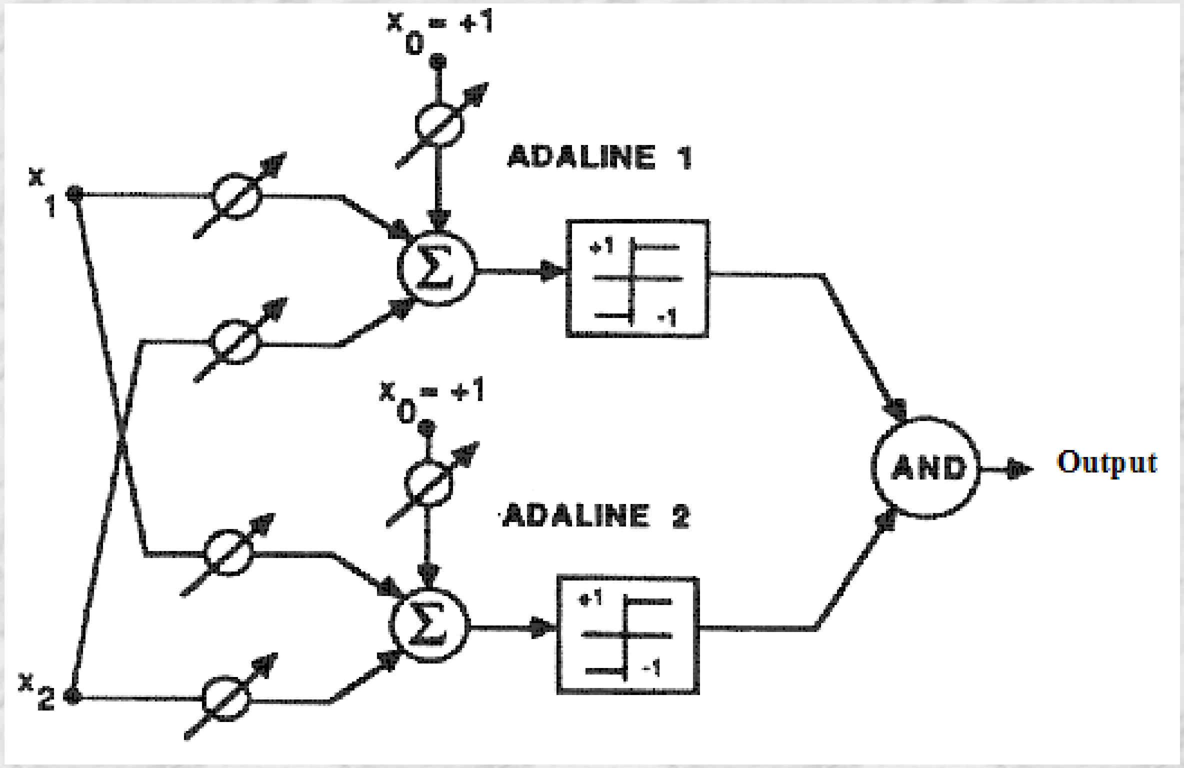


Widrow and Hoff model

When in a perceptron decision concerning change of weights is taken on the base of the output signal ADALINE uses the signal from the sum unit (marked Σ).

Widrow and Hoff model

The system of two ADALINE type elements can realize the logical AND function.



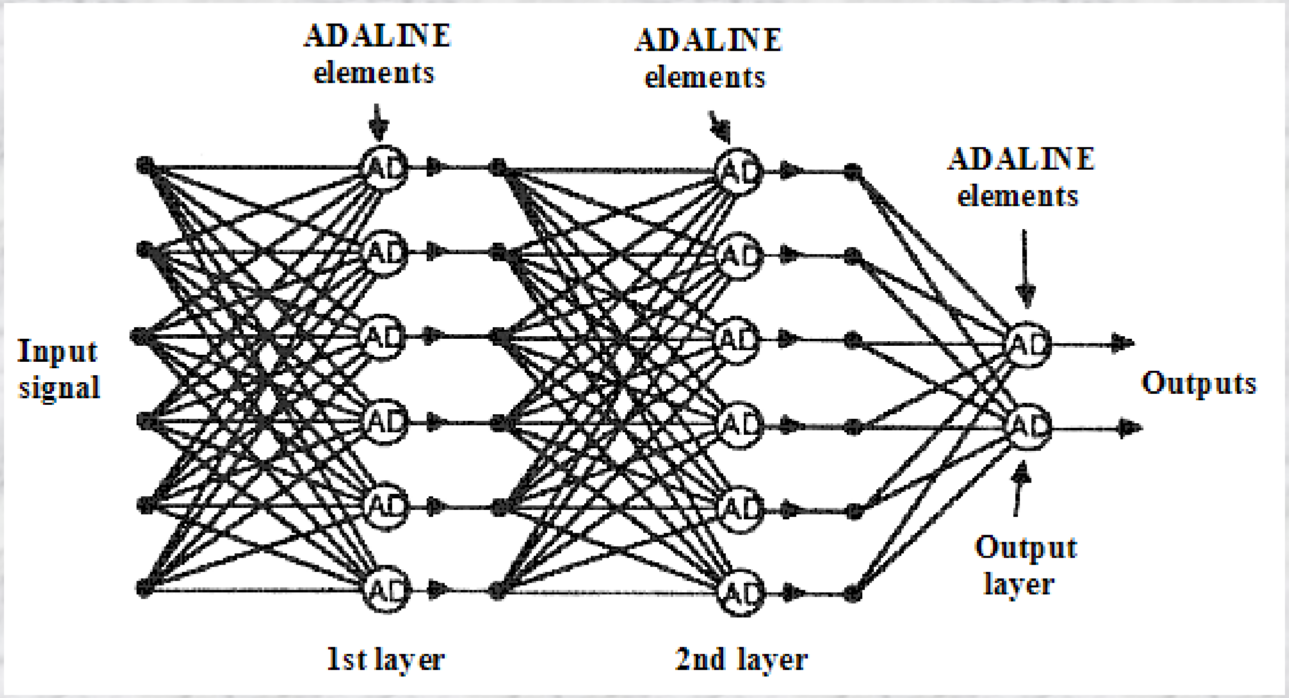
Widrow and Hoff model

Similarly to another multilayer nets (e.g. perceptron), from basic ADALINE elements one can create the whole network called ADALINE or MADALINE.

Complicated net’s structure makes difficult definition of an effective learning algorithm. The most in use is the LMS algorithm (Least-Mean-Square). But for the LMS method it is necessary to know the input and output values of every hidden layer. Unfortunately these information are not accessible.

Widrow and Hoff model

Three layer net composed of ADALINE elements create the MADALINE net.



Widrow and Hoff model

The neuron operation can be described by the formula (assuming the threshold = 0)

$$y = W^{T} \cdot X$$

where $W = [w_1, w_2, \ldots, w_n]$ is the weight vector
$X = [x_1, x_2, \ldots x_n]$ is the input signal (vector)

Widrow and Hoff model

From the inner product properties, we know that the out put signal will be bigger when the direction of the vector $\mathbf{x}_i$ in the *n*-dimensional space of input signals **X** will coincide with the direction of the vector $\mathbf{w}_i$ in the *n*-dimensional space of the weights **W**. The neuron will react stronger for the input signals more „similar" to the weight vector.

Assuming that vectors $\mathbf{x}_i$ i $\mathbf{w}_i$ are normalized (i.e. $|\mathbf{w}_i| = 1$ i $|\mathbf{x}_i| = 1$), one get

$$y = \cos\Phi$$

where Φ is the angle between the vectors $\mathbf{x}_i$ i $\mathbf{w}_i$.

Widrow and Hoff model

For the *m*-elements layer of the neurons (processing elements), we get

$$\mathbf{Y} = \mathbf{W} \cdot \mathbf{X}$$

where rows in the matrix **W** (1,2,...,m) correspond to the weights coming to particular processing elements from input nods, and

$$\mathbf{Y} = [y_1, y_2, \ldots, y_m]$$

Widrow and Hoff model

The net is mapping the input space X into R^{m}, $X \rightarrow R^{m}$. Of course this mapping is absolutely free. One can say that the net is performing the *filtering*.

A net operation is defined by the elements of a matrix W – i. e. the weights are an equivalent of the program in numerical calculation.

The a priori definition of weights is difficult, and in the multilayer nets – practically impossible.

Widrow and Hoff model

The one-step process of the weights determining can be replaced by the multi-step process – ***the learning process***.
It is necessary to expand a system adding the element able to define the output signal error and the element able to control the weights adaptation.
The method of operating the ADALINE is based on the algorithm called **DELTA** introduced by Widrow and Hoff. General idea: each input signal **X** is associated with the signal *d,* the correct output signal.

Widrow and Hoff model

The actual output signal *y* is compared with *d* and the error is calculated. On the base of this error signal and the input signal **X** the weight vector **W** is corrected.

The new weight vector **W'** is calculated by the formula

$$\mathbf{W'} = \mathbf{W} + \eta \cdot e \cdot \mathbf{X}$$

where η is the learning speed coefficient

Widrow and Hoff model

The idea is identical with the perceptron learning rule. When $d > y$ it means, that the output signal was too small, the angle between vectors **X** and **W** – was too big. To the vector **W** it is necessary to add the vector **X** multiplied by the constant

$$(0< \eta \cdot e < 1).$$

{This condition prevents too fast „rotations" of vector **W**}.

The vector **W** correction is bigger when the error is bigger – the correction should be stronger with the big error and m0ore precise with the small one.

Widrow and Hoff model

The rule assures, that *i-th* component of the vector **W** is changed more the bigger appropriate component of learned **X** was .

When the components of **X** can be both positive and negative – the sign of the error *e* defines the increase or decrease of **W.**

Widrow and Hoff model

Let us assume that the element has to learn many different input signals.

For simplicity, for *k*-th input signal $\mathbf{X}^k$ we have

$\boldsymbol{x_i^k} \in \{-1,1\}$, $y^k \in \{-1,1\}$ and $d^k \in \{-1,1\}$.

The error for the *k*-th input signal is equal

$$e^k = d^k - \sum_{i=1}^{n} w_i x_i^k$$

where η coefficient is constant and equal to 1/n.

Widrow and Hoff model

This procedure is repeated for all *m* input signals $\boldsymbol{X}^k$ and the output signal y^k should be equal to d^k for each $\boldsymbol{X}^k$.

Of course, usually such the weight vector **W**, fulfilling such solution does not exists, then we are looking for the vector **W*** able to minimize the error e^k

$$\left|\overline{e}^k\right|^2 = \frac{1}{m}\sum_{k=1}^{m}(e^k)^2 = \frac{1}{m}\sum_{k=1}^{m}\left\{d^k - \sum_{i=1}^{n} w_i x_i^k\right\}^2$$

Widrow and Hoff model

Lets denote the weights minimizing $\overline{\left|\overset{-k}{e}\right|^2}$
by $\bar{\boldsymbol{w}}_1$, $\bar{\boldsymbol{w}}_2$, … , $\bar{\boldsymbol{w}}_n$ and by $\overline{\boldsymbol{e}^2(\delta)}$ the value of
$\left|\overset{-k}{e}\right|^2$ where δ is the vector determining the difference between the weight vector **W** from optimal vector **W***

$$\delta = \mathbf{W}^* - \mathbf{W}$$

Widrow and Hoff model

The necessary condition to minimize $\overline{e^2(\delta)}$ is

$$\left.\frac{\partial \overline{\mathbf{e}^2(\delta)}}{\partial \delta_j}\right|_{\delta_J=0} = 0$$

for j = 1,2,3...,n

hence

$$\frac{\partial \overline{e^2(\delta)}}{\partial \delta_j} = \frac{2}{m}\sum_{k=1}^{m}[d^k - \sum_{i=1}^{n}\overline{w}_i x_i^k]x_j^k = 0$$

Widrow and Hoff model

yield
$$\sum_{i=1}^{n} \overline{w}_i \overline{x_i x_j} = \overline{x_j d}$$

where

$$\overline{x_i x_j} = \frac{1}{m} \sum_{k=1i}^{m} x_i^k x_j^k$$

$$\overline{x_i d} = \frac{1}{m} \sum_{k=1i}^{m} d^k x_j^k$$

Widrow and Hoff model

the optimal weight values $\overline{w_i}$ can be obtained by solving the above mentioned system of equation

It can be shown, this condition are sufficient to minimize the mean square error also.

Widrow and Hoff model

Learning process convergence

The difference equation

$$\boldsymbol{w_t(t+1) = w_t(t) + \delta(t+1) \cdot x_i(t+1)} \qquad (1)$$

describes the learning process in the *(t+1)* iteration, and

$$\boldsymbol{\delta(t+1)} = \boldsymbol{e(t+1)}/n$$

Widrow and Hoff model

$$e(t+1) = d(t+1) - \sum w_i(t)x_i(t+1) \qquad (2)$$

the set of *n* weights represents the point in the *n*-dimensional space. The second power of a distance ***L(t)*** between the optimal point in this space and the point defined by the weights $w_i(t+1)$

$$L(t) = \sum_i [\overline{w_i} - w_i(t)]^2$$

Widrow and Hoff model

In the learning process, the value of ***L(t)*** changes to ***L(t+1)***, hence

$$\Delta L(t) = L(t) - L(t+1) = \sum\{[\overline{w_i} - w_i(t)]^2 - [\overline{w_i} - w_i(t+1)]^2\} =$$

$$\sum[w_i(t+1) - w_i(t)]\,[2\overline{w_i} - 2w_i(t) - \delta(t+1)\cdot x_i(t+1)]$$

$$= \sum \delta(t+1)\cdot x_i(t+1)][2\overline{w_i} - 2\,w_i(t) - \delta(t+1)\cdot x_i(t+1)]$$

Widrow and Hoff model

where

$$\delta(t+1) = e(t+1)/n = [d(t+1) - \sum w_i(t)x_i(t+1)]/\mathrm{n}$$

there are two possible cases:

1) the optimal weights $\overline{w_i}$ give the proper output for every input signal; then

$$\Delta L(t)= \delta(t+1) \{\underbrace{2\sum x_i (t+1)\ \overline{w_i} - 2\sum w_i(t)\, x_i(t+1)}_{2\sum x_i(t+1)[\underbrace{\overline{w_i} - w_i(t)}_{\delta(t+1)x_i(t+1)}]= 2n\delta(t+1)} - n\delta(t+1)\} = \; = n\,\delta^2(t+1)$$

$$\overline{w_i} = w_i(t+1) \longrightarrow$$

Widrow and Hoff model

Because the right side is positive, then ***L(t)*** is decreasing function of *t*. Because ***L(t)*** is nonnegative, then the infimum exists, the function is convergent and of course ***δ(t+1)*** has the limit zero – thus the error ***e(t+1)*** has the limit zero also.

Widrow and Hoff model

2) The optimal weights $\overline{w_i}$ do not produce the proper output signal for every input signal

then

$$\Delta L(t) = \delta(t+1)\left\{2\sum x_i(t+1)\,\overline{w_i} - 2\sum w_i(t)\,x_i(t+1) - \sum x_i^2(t+1)\delta(t+1)\right\}$$

$$= \delta(t+1)\left\{2[d(t+1) - \Delta d(t+1)] - 2\underbrace{\sum w_i(t)\,x_i(t+1)}_{d(t+1)\,-\,e(t+1)} - \underbrace{n\delta(t+1)}_{e(t+1)/n}\right\}$$

Widrow and Hoff model

$$= \delta(t+1)\,\{2[d(t+1)-\Delta d(t+1)] \quad - 2[d(t+1)-e(t+1)] - e(t+1)\} =$$

$$= \delta(t+1)\,[e(t+1) - 2\Delta d(t+1)] \;=\; e(t+1)/n \cdot [e(t+1) - 2\Delta d(t+1)]$$

where

$$\Delta d(t+1) = d(t+1) - 2\sum x_i(t+1)\overline{w_i}$$

Widrow and Hoff model

It can be also shown that ***L(t)*** monotonically decrease when

$$|e(t+1)| > 2\,|\Delta d(t+1)|.$$

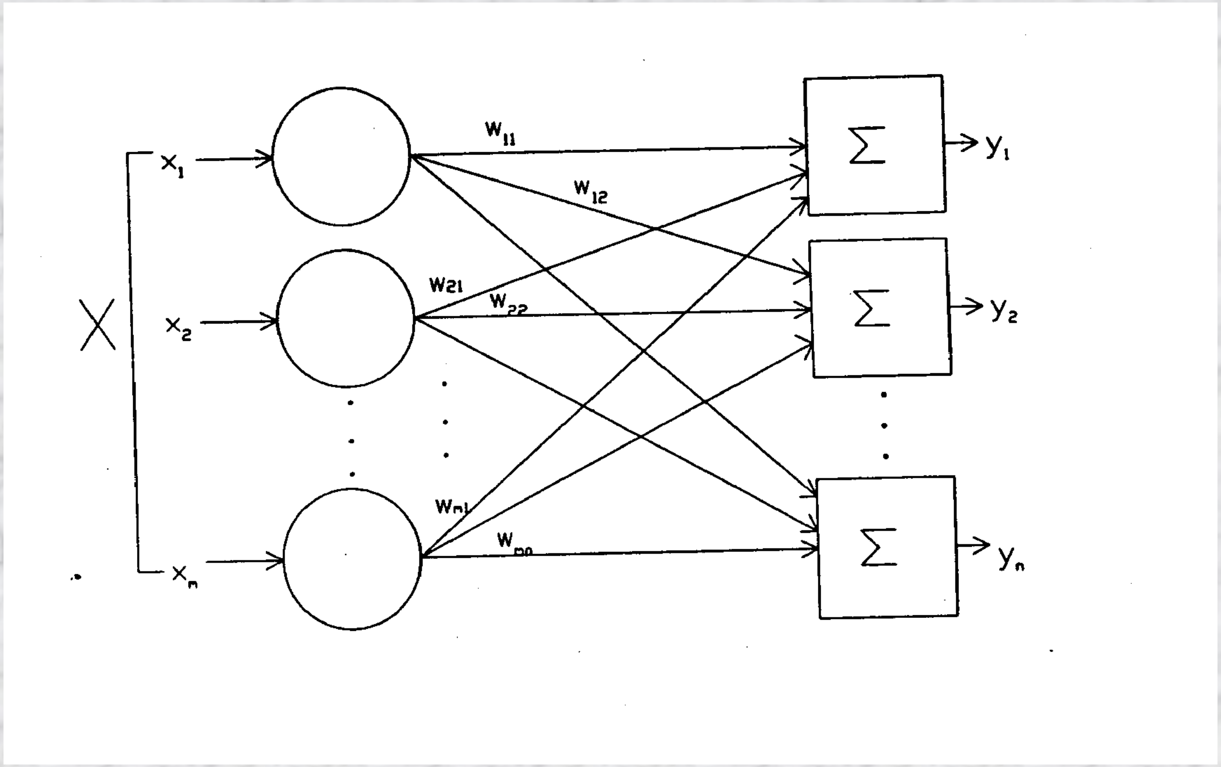
Let us assume that the learning procedure ends where the error

$$e(k) < 2 \max |\Delta d(k)|,$$

it means that error modulus for every input signal is smaller that $2 \max |\Delta d(k)|$.

The Delta Rule

The Delta learning rule



The one –layer network

The Delta learning rule

The perceptron learning rule is also the delta rule

if $y(t) = d(t) \Rightarrow \quad w_i(t+1) = w_i(t)$

if $y(t) \neq d(t) \Rightarrow \quad w_i(t+1) = w_i(t) + \eta \cdot \Delta \cdot x_i(t)$

where

$0 \leq \eta \leq 1$ is the learning coefficient

and $\Delta = d(t) - y(t)$

The Delta learning rule

The basic difference is in the error definition – discrete in the perceptron and continuous in the Adaline model.

The Delta learning rule

Let δ_k, the error term, defines the difference between the ***d***$_k$ desired response of the *k*-th element of the output layer, and is the actual response (real) $\boldsymbol{y_k}$.

Let us define the error function ***E*** to be equal to the square of the difference between the actual and desired output, for all elements in the output layer

$$E = \sum_{k=1}^{N} \delta_k^2 = \sum_{k=1}^{N} (d_k - y_k)^2$$

The Delta learning rule

Because

$$y_k = \sum_{i=1}^{n} w_{ik} x_i$$

thus

$$E = \sum_{k=1}^{N} (d_k - \sum_{i=1}^{n} w_{ik} x_i)^2$$

The Delta learning rule

The error function E is the function of all the weights. It is the square function with respect to each weight, so it has exactly one minimum with respect to each of the weights. To find this minimum we use the ***gradient descend method.***

Gradient of **E** is the vector consisting of the partial derivatives of **E** with respect to each variable. This vector gives the direction of most rapid increase in function; the opposite direction gives the direction of most rapid decrease in the function.

The Delta learning rule

So, the weight change is proportional to the partial derivative of a error function with respect to this weight with the minus sign.

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

where η is the learning rate

The Delta learning rule

Each weight can be fixed this way.
Lets calculate the partial derivative of ***E***

$$\frac{\partial E}{\partial w_{ik}} = \frac{\partial E}{\partial \delta_k}\frac{\partial \delta_k}{\partial w_{ik}} = 2\delta_k \frac{\partial \delta_k}{\partial y_k}\frac{\partial y_k}{\partial w_{ik}} = 2\delta_k(-1)x_i = -2\delta_k x_i$$

thus

$$\Delta w_{ik} = 2\eta\delta_k x_i$$

The Delta learning rule

The DELTA RULE changes weights in a net proportionally to the output error (the difference between the real and desired output signal), and the value of input signal

$$\boxed{\Delta w_{ik} = 2\eta\delta_k x_i}$$

The multilayer Perceptron

Many years the idea of multi layer perceptron was introduced . Multi – typically three

Layers: input, output and hidden.

In 1986 Rumelhart and McClelland described the new learning rule **the backpropagation learning rule**.