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 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
- 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...

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 $R$. 
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.

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.

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:

\[
\begin{align*}
\text{w}_a^{(\text{new})} &= \text{w}_a^{(\text{old})} - \text{input}_i \\
\text{w}_b^{(\text{new})} &= \text{w}_b^{(\text{old})} + \text{input}_i \\
\text{input}_i &= \pm 1
\end{align*}
\]

The one-layer and two-elements Perceptron

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class A</td>
<td>1</td>
</tr>
<tr>
<td>Class B</td>
<td>0</td>
</tr>
</tbody>
</table>
The one-layer and two-elements Perceptron

Weights modification rule

Assuming
- \( \zeta = \) output (library) – output (real)
  than
- \( w_B(\text{new}) = w_B(\text{old}) + \Delta w_B \)

For example:
- \( w_A(\text{new}) = w_A(\text{old}) + \zeta \) input,
- \( \begin{array}{c}
\text{input} = \begin{cases}
0 & \text{if correct output from the element A} \\
+1 & \text{if incorrect output from the element A}
\end{cases}
\end{array} \)

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 !!!

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 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.

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.”

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)**

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>output</th>
<th>point</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>A₀</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>B₁</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>B₁</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>A₁</td>
</tr>
</tbody>
</table>
Function $F$ is the simple threshold function producing at the output signal 0 (zero) when signal $s$ is just below $\Theta$, and signal 1 (one) when signal $s$ is greater (or equal) $\Theta$.

**XOR Problem**

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.

Geometrical interpretation of the equation

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

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

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

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

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).
The problem of linear separability impose limitations for 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 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 $X_i$, and the set of weights $w$ as the another vector in the same space $W$. Increasing the weights is performed by $W + X$, and decreasing by $W - X$.

We can simplify the algorithm still further, if we define $F$ to be $F^+ \cup F^-$ i.e., $F^+$ and the negatives of $F^-$. The convergence of the learning procedure:

**start:** Choose any value for $W$

**test:** Choose any $X$ from $F^+$ or $F^-$
- If $X \in F^+$ and $W \cdot X > 0 \implies$ test
  - Otherwise $\implies$ add
  - replace $W$ by $W + X \implies$ test
- If $X \in F^-$ and $W \cdot X < 0 \implies$ test
- If $X \in F^-$ and $W \cdot X > 0 \implies$ subtract
  - Replace $W$ by $W - X \implies$ test

Notice that we go to subtract when $X \in F^-$, and if we consider that going to subtract is the same as going to add $X$ replaced by $-X$.
The Perceptron

Theorem and proof

Convergence Theorem:
Program will only go to add a finite number of times.

Proof:
Assume that there is a unit vector \( \mathbf{w}^* \), which partitions up the space, and a small positive fixed number \( \delta \), such that
\[
\mathbf{w}^* \cdot \mathbf{x} > \delta \quad \text{for every } \mathbf{x} \in F
\]
Define
\[
G(\mathbf{w}) = \frac{\mathbf{w}^* \cdot \mathbf{w}}{||\mathbf{w}||}
\]
and note that \( G(\mathbf{w}) \) is the cosine of the angle between \( \mathbf{w} \) and \( \mathbf{w}^* \).

Since \( ||\mathbf{w}^*|| = 1 \), we can say that \( G(\mathbf{w}) \leq 1 \).
Consider the behavior of \( G(\mathbf{w}) \) through add.
The numerator:
\[
\mathbf{w}^* \cdot \mathbf{w}_{n+1} = \mathbf{w}^* (\mathbf{w}_n + \mathbf{x}) = \mathbf{w}^* \mathbf{w}_n + \mathbf{w}^* \mathbf{x} + \delta
\]
since \( \mathbf{w}^* \mathbf{x} > \delta \).
Hence, after the \( m \)th application of add we have
\[
\mathbf{w}^* \cdot \mathbf{w}_m \geq \delta m
\]
(1)

The denominator:
Since \( \mathbf{w} \cdot \mathbf{x} \) must be negative (add operation is performed), then
\[
| \mathbf{w}_{n+1} |^2 = | \mathbf{w}_{n+2} |^2 = (\mathbf{w}_n + \mathbf{x}) \cdot (\mathbf{w}_n + \mathbf{x}) = | \mathbf{w}_n |^2 + 2 \mathbf{w}_n \cdot \mathbf{x} + | \mathbf{x} |^2
\]
Moreover \( | \mathbf{x} | = 1 \), so
\[
| \mathbf{w}_{n+1} |^2 < | \mathbf{w}_n |^2 + 1,
\]
and after \( m \)th application of add
\[
| \mathbf{w}_m |^2 < m.
\]
(2)

Combining (1) i (2) gives
\[
G(\mathbf{w}_m) = \frac{(\mathbf{w}^* \cdot \mathbf{w}_m)}{|\mathbf{w}_m|} \geq \frac{m \delta}{\sqrt{m}}
\]
Because \( G(\mathbf{w}) \leq 1 \), so we can write
\[
\sqrt{m} \cdot \delta \leq 1 \quad \text{i.e.} \quad m \leq \frac{1}{\delta^2}
\]

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 \( \delta \), such that \( \mathbf{w} \cdot \mathbf{x} > \delta \). Inequality (3) then says that we can make \( \delta \) 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 \( \mathbf{w} \), 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

1 step – initialize weight and threshold:
Define \( w_i(t) \), (\( i = 0,1,...,n \)) to be the weight from input \( i \) at time \( t \), and \( \theta \) to be a threshold value if the output node. Set \( w_i(0) \) to small random numbers.

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

3 step:
Calculate actual output
\[
\hat{y}(t) = f\left( \sum_i w_i(t)x_i(t) \right)
\]

4 step:
Adapt weights

4 step (cont.):
if \( \hat{y}(t) = d(t) \) \( \Rightarrow \)
\[
w_{i}(t+1) = w_{i}(t)
\]
if \( \hat{y}(t) = 0 \) and \( d(t) = 1 \) \( \Rightarrow \)
\[
w_{i}(t+1) = w_{i}(t) + \eta \cdot x_{i}(t)
\]
if \( \hat{y}(t) = 1 \) and \( d(t) = 0 \) \( \Rightarrow \)
\[
w_{i}(t+1) = w_{i}(t) - \eta \cdot x_{i}(t)
\]
0 \( \leq \eta \leq 1 \), a positive gain term that controls the adaptation rate.

Algorithm modifications

Widrow and Hoff modification

4 step (cont.):
if \( \hat{y}(t) = d(t) \) \( \Rightarrow \)
\[
w_{i}(t+1) = w_{i}(t)
\]
if \( \hat{y}(t) \neq d(t) \) \( \Rightarrow \)
\[
w_{i}(t+1) = w_{i}(t) + \eta \cdot \Delta \cdot x_{i}(t)
\]
0 \( \leq \eta \leq 1 \) a positive gain term that controls the adaptation rate.
\[
\Delta = d(t) - \hat{y}(t)
\]
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 ADALINEs (ADAptive Linear NEurons), who are also connected into a many ADALINE, or MADALINE structure.

The perceptron learning algorithm

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

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.
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.

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

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

\[ y = W^T \cdot X \]

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

From the inner product properties, we know that the output signal will be bigger when the direction of the vector \( x_i \) in the \( n \)-dimensional space of input signals \( X \) will coincide with the direction of the vector \( w_i \) 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 \( x_i \) \( w_i \) are normalized (i.e. \( |w_i| = 1 \) \( |x_i| = 1 \)), one get

\[ y = \cos \Phi \]

where \( \Phi \) is the angle between the vectors \( x_i \) \( w_i \).

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

\[ Y = WX \]

where rows in the matrix \( W \) \( (1, 2, ..., m) \) correspond to the weights coming to particular processing elements from input nods, and
\[ Y = [y_1, y_2, ..., y_m] \]

The net is mapping the input space \( X \) into \( \mathbb{R}^m \), \( X \rightarrow \mathbb{R}^m \).

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.
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 \( \mathbf{X} \) is associated with the signal \( d \), the correct output signal.

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 \( \mathbf{X} \) the weight vector \( \mathbf{W} \) is corrected.

The new weight vector \( \mathbf{W}' \) is calculated by the formula

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

where \( \eta \) is the learning speed coefficient.

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 \( \mathbf{X} \) and \( \mathbf{W} \) – was too big. To the vector \( \mathbf{W} \) it is necessary to add the vector \( \mathbf{X} \) multiplied by the constant \( 0 < \eta \cdot e < 1 \).

\( \{ \)This condition prevents too fast “rotations” of vector \( \mathbf{W} \).\( \} \)

The vector \( \mathbf{W} \) correction is bigger when the error is bigger – the correction should be stronger with the big error and more precise with the small one.

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 \( x^k_i \in \{-1,1\} \), \( y^k \in \{-1,1\} \) and \( d^k \in \{-1,1\} \).

The error for the \( k \)-th input signal is equal

\[
e^k = d^k - \sum_{j=1}^{n} w_j x^k_j
\]

where \( \eta \) coefficient is constant and equal to \( 1/n \).

This procedure is repeated for all \( m \) input signals \( \mathbf{X}^m \) and the output signal \( y^m \) should be equal to \( d^m \) for each \( \mathbf{X}^m \).

Of course, usually such the weight vector \( \mathbf{W} \), fulfilling such solution does not exists, then we are looking for the vector \( \mathbf{W}^* \) able to minimize the error \( e^m \)

\[
\left| e^m \right| = \frac{1}{m} \sum_{k=1}^{m} e^k = \frac{1}{m} \sum_{k=1}^{m} \left( d^k - \sum_{j=1}^{n} w_j x^k_j \right)
\]
Let's denote the weights minimizing $\sum_{i} w_i$ by $\bar{w}_1, \bar{w}_2, ..., \bar{w}_n$ and by $\bar{e}(\delta)$ the value of $\sum_{i} e_i$ where $\delta$ is the vector determining the difference between the weight vector $W$ from optimal vector $W^*$

$$\delta = W^* - W$$

The necessary condition to minimize $\bar{e}(\delta)$ is

$$\frac{\partial \bar{e}(\delta)}{\partial \delta} = 0$$

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

hence

$$\frac{\partial e(t+1)}{\partial \delta} = -\frac{2}{m} \sum_{i=1}^{m} (d_i - \sum_{i=1}^{n} w_i x_i) x_i = 0$$

The difference equation

$$w_i(t+1) = w_i(t) + \delta(t+1) x_i(t+1)$$

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

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

the optimal weight values $\bar{w}_i$ 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.

$$e(t+1) = d(t+1) - \sum_{i=1}^{n} w_i x_i(t+1)$$

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 $w_i(t+1)$

$$L(t) = \sum_{i} [\bar{w}_i - w_i(t)]^2$$
In the learning process, the value of \( L(t) \) changes to \( L(t+1) \), hence

\[
\Delta L(t) = L(t) - L(t+1) = \sum \left( \left[ \frac{w_i}{w_i + w_i(t+1)} \right]^2 - \left[ \frac{w_i}{w_i + w_i(t+1)} \right]^2 \right) = \sum \left[ w_i(t+1) - w_i(f(t+1)) \right] \left[ 2w_i - 2w_i(t) - \delta(t+1)x_i(t+1) \right]
\]

\[
= \sum \delta(t+1)x_i(t+1) \left[ 2w_i - 2w_i(t) - \delta(t+1)x_i(t+1) \right]
\]

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

\[
\delta(t+1) = e(t+1)/n = \left[ d(t+1) - \sum w_i(t)x_i(t+1) \right]/n
\]

There are two possible cases:

1) The optimal weights \( w_i \) give the proper output for every input signal, then

\[
\Delta L(t) = \delta(t+1) \left\{ \sum x_i(t+1) w_i - 2 \sum w_i(t) x_i(t+1) - n\delta(t+1) \right\} = n \delta(t+1)
\]

\[
w_i = w_i(t+1)
\]

2) The optimal weights \( w_i \) do not produce the proper output signal for every input signal then

\[
\Delta L(t) = \delta(t+1) \left\{ \sum x_i(t+1) w_i - 2 \sum w_i(t) x_i(t+1) - n\delta(t+1) \right\} = n \delta(t+1)
\]

\[
\delta(t+1) = e(t+1)/n \cdot \left[ e(t+1) - 2\delta(t+1) \right]
\]

\[
\Delta d(t+1) = d(t+1) - 2\sum x_i(t+1) w_i
\]

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 than

\[
2 \max |\Delta d(k)|.
\]
The Delta Rule

The one-layer network

The Delta learning rule

The perceptron learning rule is also the delta rule if \( y(t) = d(t) \)

\[ w_i(t+1) = w_i(t) \]

if \( y(t) \neq d(t) \)

\[ w_i(t+1) = w_i(t) + \eta \Delta \cdot x_i(t) \]

where

\[ 0 \leq \eta \leq 1 \]

is the learning coefficient

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

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

Let \( \delta_k \), the error term, defines the difference between the \( d_k \) desired response of the \( k \)-th element of the output layer, and \( y_k \) is the actual response (real) \( y_k \).

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:

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

Because

\[ y_k = \sum_{i=1}^{N} w_i x_i \]

thus

\[ E = \sum_{k=1}^{N} (d_k - \sum_{i=1}^{N} w_i x_i)^2 \]
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 descent 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.

Each weight can be fixed this way. Let's calculate the partial derivative of $E$

$$\frac{\partial E}{\partial w_k} = \frac{\partial E}{\partial \delta_k} \frac{\partial \delta_k}{\partial x_i} = 2\delta_k \frac{\partial \delta_k}{\partial \delta_k} \frac{\partial \delta_k}{\partial x_i} = 2\delta_k (-1)x_i = 2\delta_i x_i$$

Thus

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

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 $\eta$ is the learning rate

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

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

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.