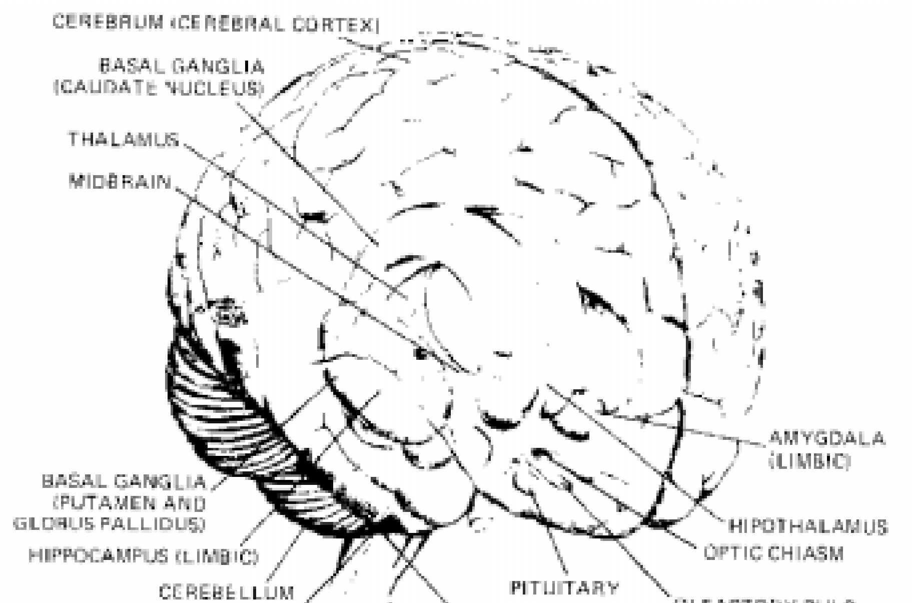
Lecture 10

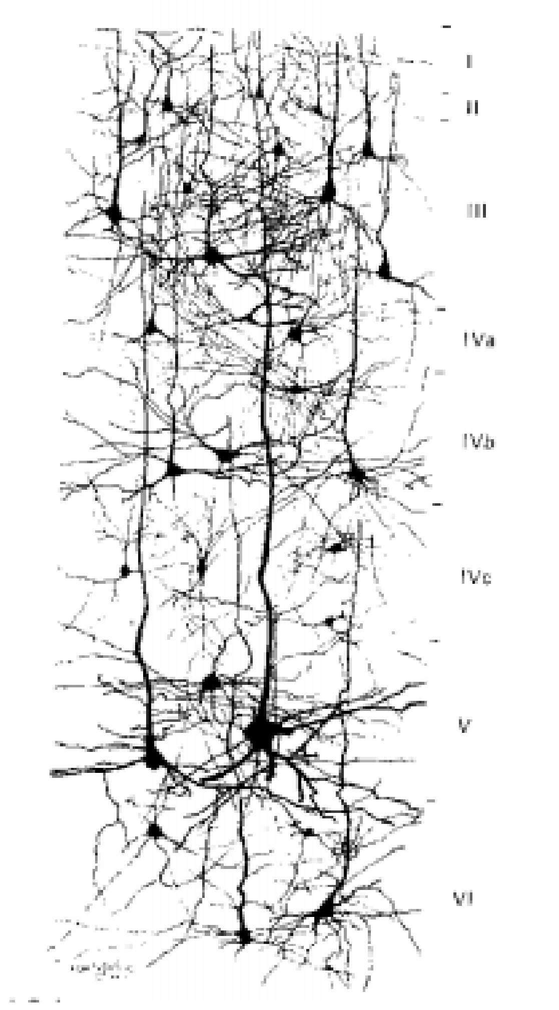
Neural networks

Some stimulating concepts from the neurobiology

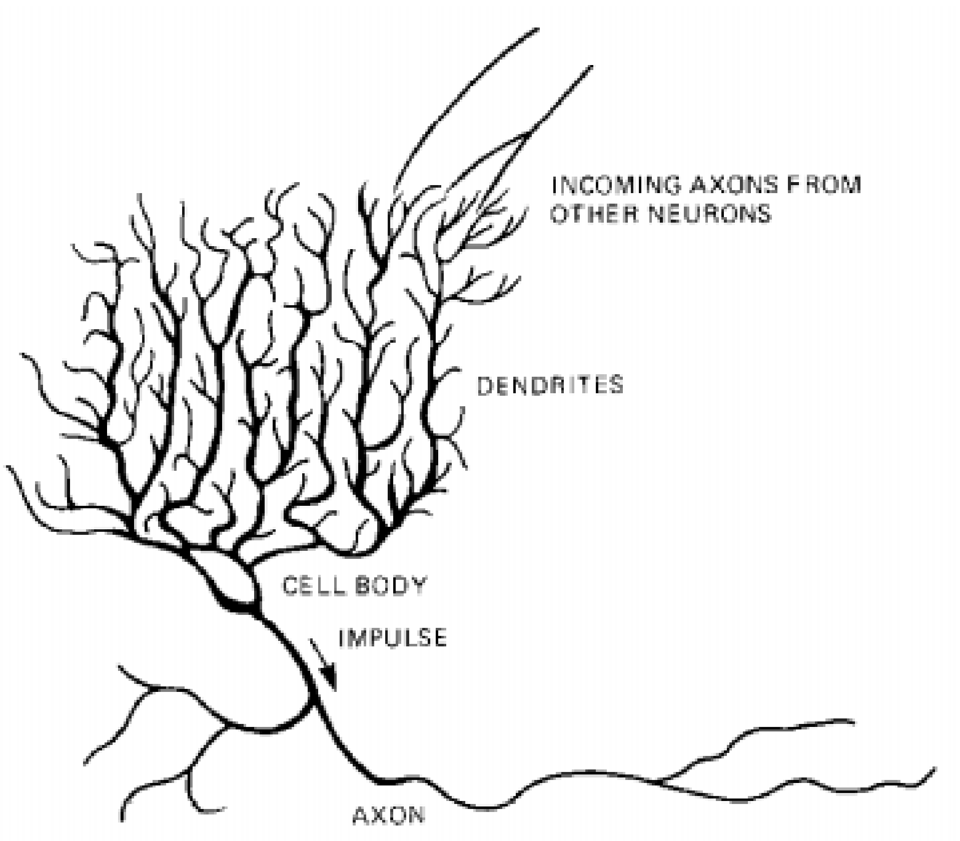
1. **Human brain** - a top result (the most complex system that we know) of evolutionary process since the big bang.



2. The human brain is composed of 10^{11} neural cells - **neurons**, that are highly interconnected.

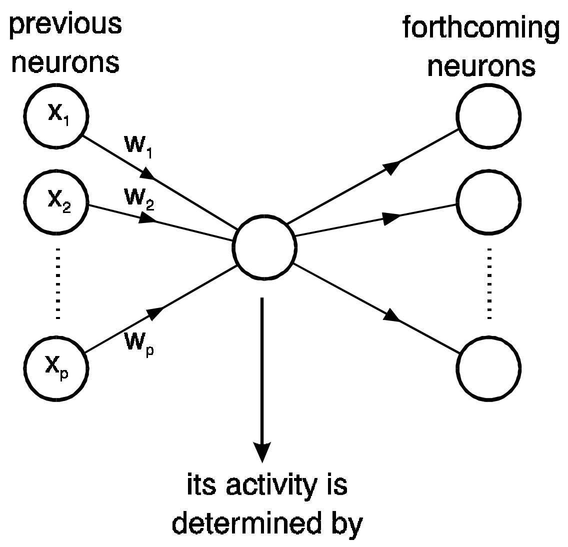


3. The neuron contains many (few hundreds) input as well as output **connections**



4. Each neurons has an **activity** (roughly, zero or one). Its inner potential is determined as a **weighted summation of input activities** of other neurons that are in-connected with the neuron.

If the potential is greater than a **threshold value**, then the neuron starts to fire (it has a nonzero activity) signals to other neurons that are out-connected with the neuron.



$$x = t\left(\sum_{i=1}^{p} w_i x_i + \vartheta\right)$$

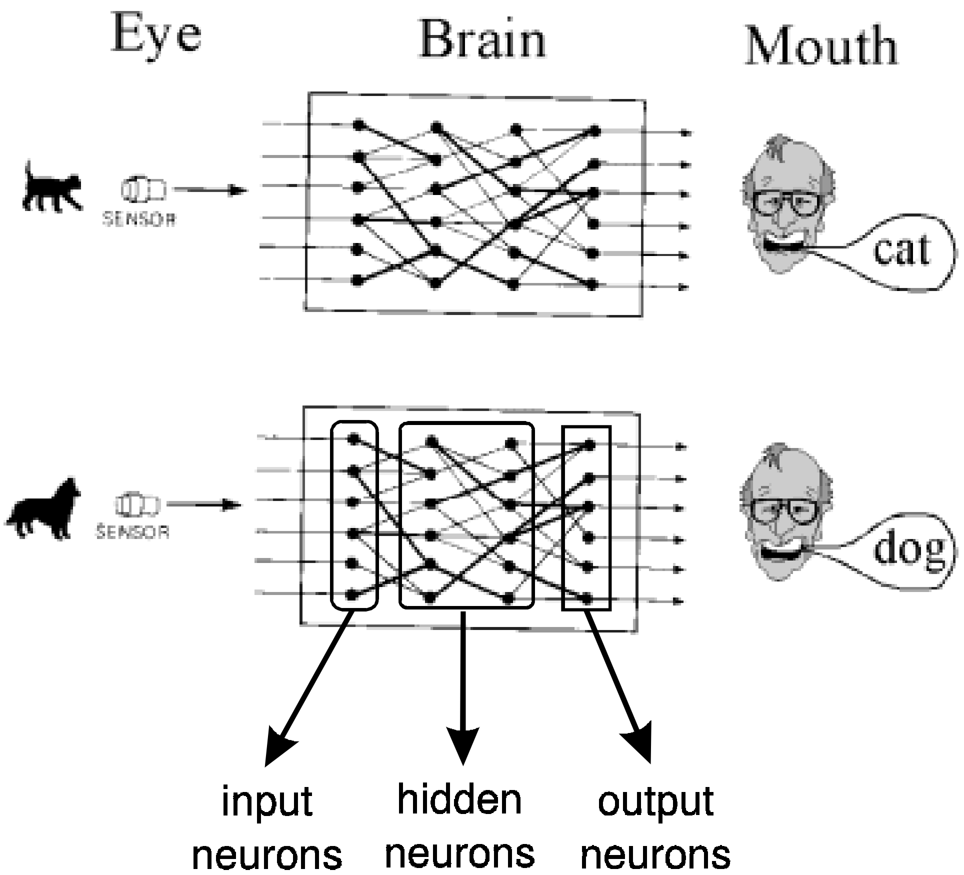
$$t(\xi) = \begin{cases} 1 & (\xi > 0) \\ 0 & (\xi \le 0) \end{cases}$$ transfer function

5. Neurons are classified as follows:

(1) **Input neurons**, they get input information for the brain.

(2) **Hidden neurons**, they process input information onto output information

(3) **Output neurons**, they send output information to motor centers.



Artificial neural networks (NN)

Basic stimulus for their constructions are gained from the neurobiology (see previous transparencies).

Many types of NNs are suggested. We will discuss here only the so-called **feed-forward NNs** that are trained by the **back-propagation method** (Rumelhart et al. 1986). This type of NNs, widely used in computer science and many branches of science and technology, belongs to the most popular paradigms of NN theory.

Definition of NN

Let G=(V,E) be an acyclic oriented graph

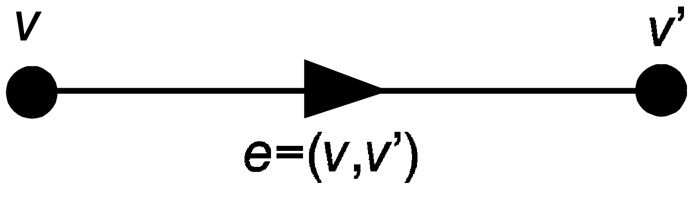
Vertex set V is composed of N vertices (neurons)

$$V = \{v_1, v_2, \ldots, v_N\}$$

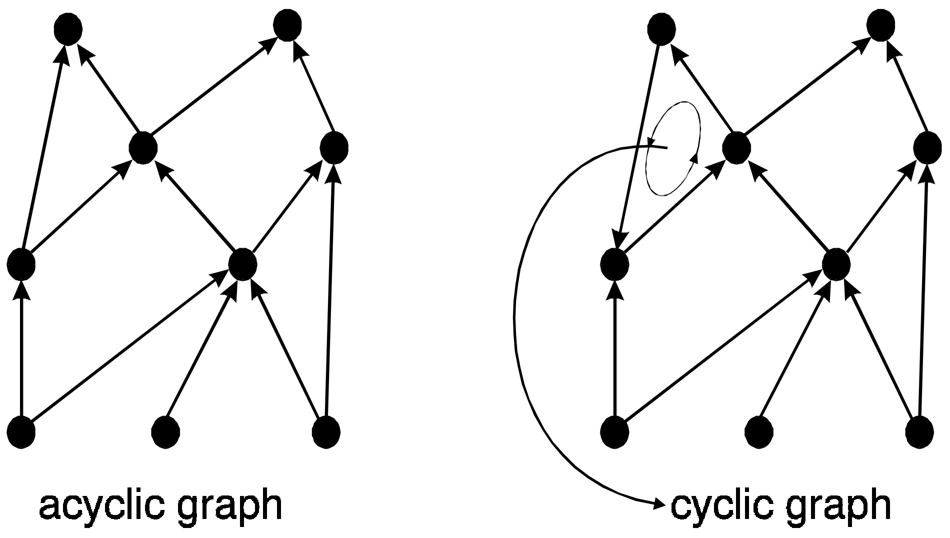
Edge set E is composed of M oriented edges (connections)

$$E = \{e_1, e_2, \ldots, e_M\}$$

Each connection $e \in E$ is interpreted as an ordered pair (v,v') of two neurons from V



Examples of oriented graphs



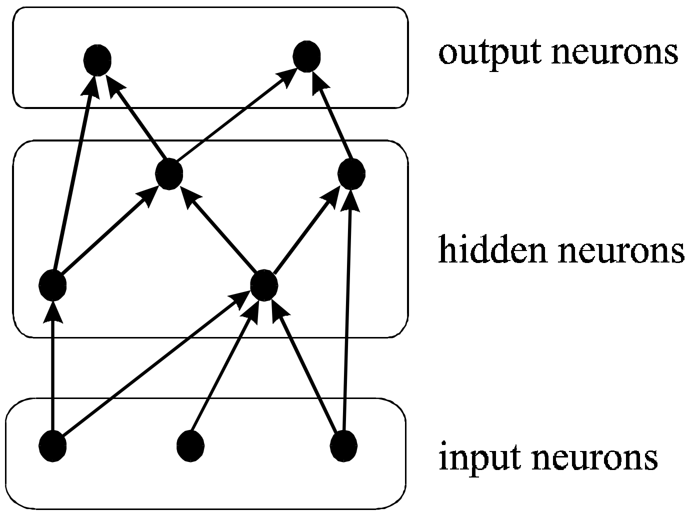
Neurons of acyclic graph are uniquely classified as follows:

$$V = V_I \cup V_H \cup V_O$$

V_I input neurons that are incident only with outgoing connections

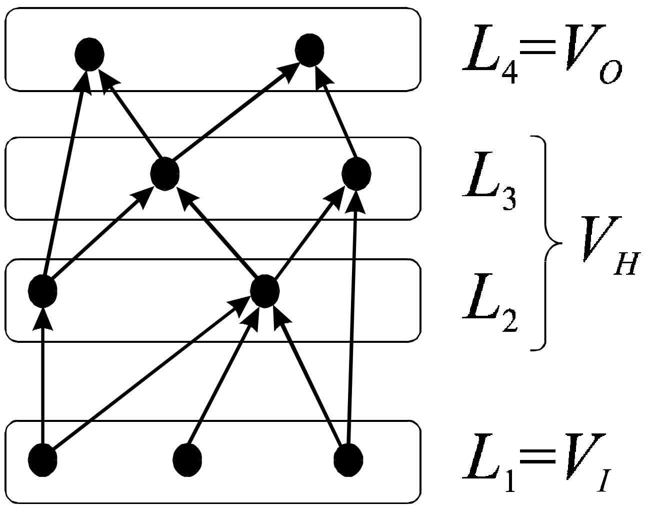
V_H hidden neurons that are simultaneously incident with outgoing and incoming connections

V_O output neurons that are incident only with incoming connections



Neurons of acyclic graphs are divided onto the so-called **layers**

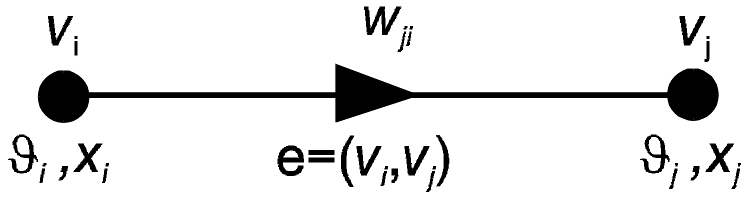
$$V = L_1 \cup L_2 \cup \ldots \cup L_{t-1} \cup L_t$$
$$V_I = L_1$$
$$V_H = L_2 \cup \ldots \cup L_{t-1}$$
$$V_O = L_t$$



Neurons and connections of the graph are **evaluated** by real numbers:

(1) Each connection $e=(v_i,v_j)$ is evaluated by the **weight coefficient** w_{ji} .

(2) Each hidden or output neuron v_i is evaluated by the **threshold coefficient** ϑ_i and the **activity** x_i



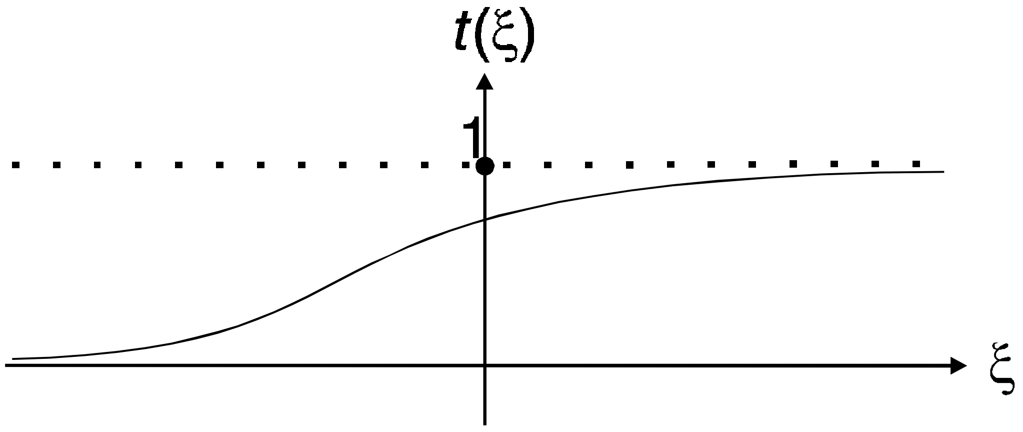
Activities are determined by

$$\boxed{x_i = t\left(\sum_j w_{ij} x_j + \vartheta_i\right)}$$

where $t(\xi)$ is the so-called **transfer function** specified, in general, by

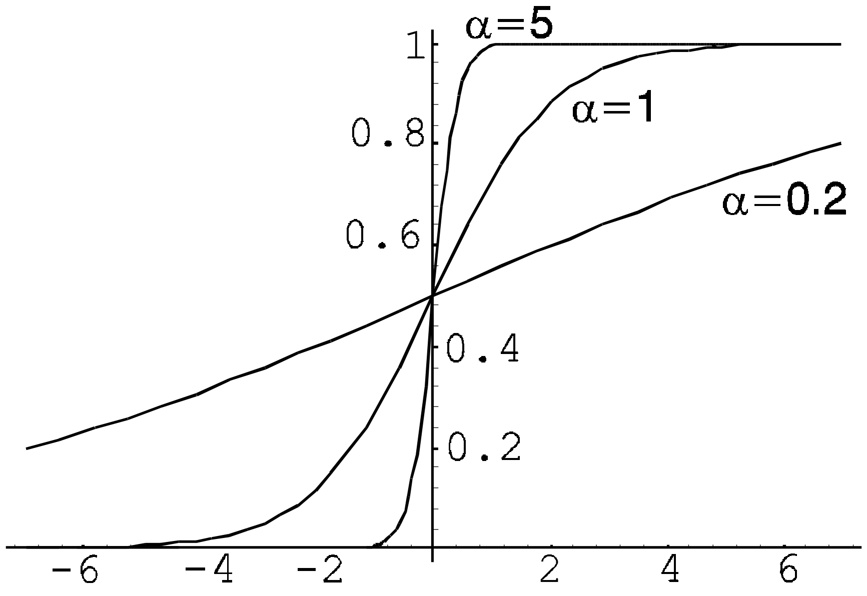
(1) $t : R \rightarrow (0,1)$ is continuos and monotonously increasing function, satisfying

(2) asymptotic conditions $t(-\infty)=0$ and $t(\infty)=1$.

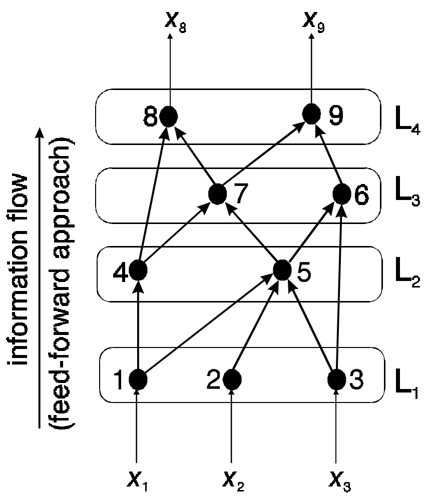


Sigmoid realization of the transfer function

$$t_\alpha(\xi) = \frac{1}{1 + e^{-\alpha\xi}}$$



$$\lim_{\alpha\to\infty} t_\alpha(\xi) = \begin{cases} 1\,(\xi > 0) \\ 1/2\,(\xi = 0) \\ 0\,(\xi < 0) \end{cases}$$



Why feed-forward neural networks? The activities of hidden and output activities are evaluated recurrently going bottom-up through all layers.

Step 1 (L_2)

$$x_4=t(w_{41}x_1+\vartheta_4)$$

$$x_5=t(w_{51}x_1+w_{52}x_2+w_{53}x_3+\vartheta_5)$$

Step 2 (L_3)

$$x_6=t(w_{65}x_5+w_{63}x_3+\vartheta_6)$$

$$x_7=t(w_{74}x_4+w_{75}x_5+\vartheta_7)$$

Step 3 (L_4)

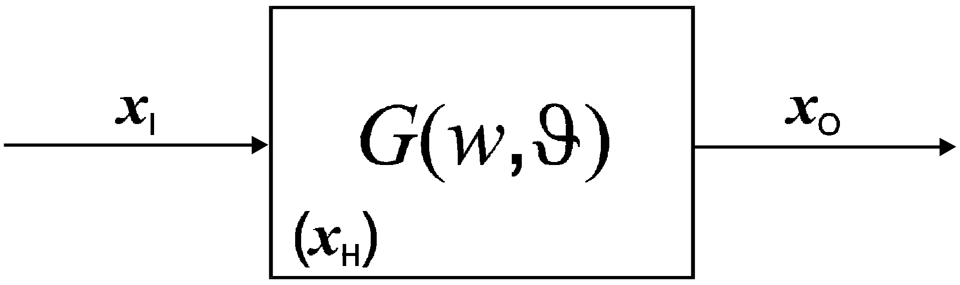
$$x_8=t(w_{84}x_4+w_{87}x_7+\vartheta_8)$$

$$x_9=t(w_{97}x_7+w_{96}x_6+\vartheta_9)$$

Important note: This recurrent feed-forward approach is applicable only for NNs that are represented by **acyclic** oriented graphs

Formal generalization

NN may be understood as a **mapping of input activities onto output activities**, hidden activities are considered as auxiliary results that should not be explicitly displayed



$\boldsymbol{x}_\text{I}$
$G(w,\vartheta)$
$(\boldsymbol{x}_\text{H})$
$\boldsymbol{x}_\text{O}$

$$\boldsymbol{x} = \boldsymbol{x}_I \oplus \boldsymbol{x}_H \oplus \boldsymbol{x}_O$$
$$\boldsymbol{x}_O = G\left(\boldsymbol{x}_I;\boldsymbol{w},\vartheta\right)$$
$$\boldsymbol{x}_I \in R^N, \boldsymbol{x}_O \in \left(0,1\right)^M$$

$$\boxed{G\left(\boldsymbol{w},\vartheta\right): R^N \to \left(0,1\right)^M}$$

Classification of objects and regression analysis (general comment)

Let us introduce a set of **objects** $O=\{o_1,o_2,....\}$.

Each object of O is determined by the so-called **descriptor** $\mathbf{x}(o)$ and **property** $\mathbf{y}(o)$.

The descriptor and property entities are mutually related by a **hypothetical function**

$$\mathbf{y}=F(\mathbf{x})$$

In most cases of interest, an analytical form of this function F is **unknown.**

Regression analysis offers a numerical tool how to model the function F. We introduce the so-called **training set** (regression table)

$$A_{\mathrm{train}}=\{\mathbf{x}/\mathbf{y},\ \mathbf{x}'/\mathbf{y}',...\}$$

composed of pairs of the descriptor and the required property

Goal: To find optimal parameters of the so-called **model function** $G(\boldsymbol{x},\boldsymbol{w})$ (where $\boldsymbol{w}$ denotes its parameters) such that it provides properties that are closely related to the required ones (determined by $\boldsymbol{y}=F(\boldsymbol{x})$)

$$E(\boldsymbol{w})=\frac{1}{2}\sum_{\boldsymbol{x}/\boldsymbol{y}\in A_{train}}\left(G(\boldsymbol{x},\boldsymbol{w})-\boldsymbol{y}\right)^2$$

Optimal values of the parameters $\boldsymbol{w}$ are determined by

$$\boxed{\bar{\boldsymbol{w}}=\arg\min_{\boldsymbol{w}\in W} E(\boldsymbol{w})}$$

Objective function $E(\boldsymbol{w})$ has the global minimum at $\bar{\boldsymbol{w}}$.

Common believe: It is expected that the so-called **adapted** model function $G(\boldsymbol{x},\bar{\boldsymbol{w}})$ well approximates the hypothetical function $F(\boldsymbol{x})$ **outside the training set**

$$\boxed{F(\boldsymbol{x})\approx G(\boldsymbol{x},\bar{\boldsymbol{w}})}$$

Adaptation process (learning) of neural networks

The main idea is the same as in the regression analysis. The model function G is constructed so that it corresponds to a preselected architecture of the used NN (see previous transparencies), we get $G(\boldsymbol{x};\boldsymbol{w},\vartheta)$.

Input activities - descriptors of objects that are classified by the NN

Output activities - properties of objects

The training set A_{train} is composed of pairs of the input activity and the required output activity

$$A_{train} = \left\{ \boldsymbol{x}_I / \boldsymbol{x}_{O,req}, \boldsymbol{x}'_I / \boldsymbol{x}'_{O,req}, \ldots \right\}$$

Objective function is determined by

$$E(\boldsymbol{w},\vartheta)=\frac{1}{2}\sum_{\boldsymbol{x}_I / \boldsymbol{x}_{req,O}\in A_{train}}\left(G(\boldsymbol{x}_I;\boldsymbol{w},\vartheta)-\boldsymbol{x}_{req,O}\right)^2$$

Adaptation process of NN consists in looking for the weight and threshold coefficients that minimize the objective function $E(\boldsymbol{w},\vartheta)$

$$\boxed{\left(\bar{\boldsymbol{w}},\bar{\vartheta}\right)=\arg\min_{(\mathrm{w},\vartheta)}E(\boldsymbol{w},\vartheta)}$$

How to realize this minimization problem? Simple gradient method (called the **steepest descent method**) is usually applied. The weight and threshold coefficients are recurrently updated by

$$\boxed{\begin{aligned} w_{ij}^{(k+1)} &= w_{ij}^{(k)}-\lambda\frac{\partial E}{\partial w_{ij}} \\ \vartheta_i^{(k+1)} &= \vartheta_i^{(k)}-\lambda\frac{\partial E}{\partial \vartheta_i} \end{aligned}}$$

where λ is a small positive number called the **learning rate**.

Back-propagation method
(calculation of grad *E*)

It is assumed that the objective function *E* is determined with respect to a single pair of input and required output activity $\boldsymbol{x}_I/\boldsymbol{x}_{req,O}$

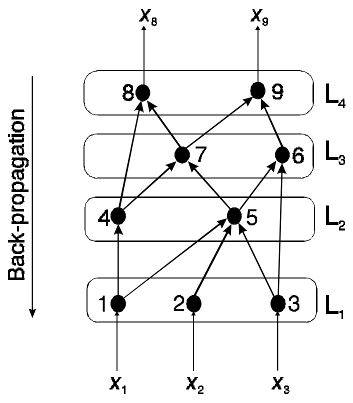
$$E(\boldsymbol{w},\vartheta)=\frac{1}{2}\left(G(\boldsymbol{x}_I;\boldsymbol{w},\vartheta)-\boldsymbol{x}_{req,O}\right)^2$$

Applying standard rules for evaluation of partial derivatives of composed functions (**chain rule**), partial derivatives of the above specified *E* are

$$\boxed{\begin{gathered}\frac{\partial E}{\partial w_{ij}}=\frac{\partial E}{\partial \vartheta_i}x_j\\ \frac{\partial E}{\partial \vartheta_i}=t'(\xi_i)\left(g_i+\sum_k\frac{\partial E}{\partial \vartheta_k}w_{ki}\right)\end{gathered}}$$

where $t'(\xi)=t(\xi)[1-t(\xi)]$ is the first derivative of the transfer function, and

$$g_i=\begin{cases}\left(x_i-x_{i,req}\right) & (\text{for}\, i\in V_O)\\ 0 & (\text{for}\, i\notin V_O)\end{cases}$$



Why back-propagation method? The evaluation of the grad *E* is performed in the opposite direction of the evaluation of activities.

Step 1 (L_4)

$$\frac{\partial E}{\partial \vartheta_8} = x_8\left(1 - x_8\right)\left(x_8 - x_{req,8}\right)$$

$$\frac{\partial E}{\partial \vartheta_9} = x_9\left(1 - x_9\right)\left(x_9 - x_{req,9}\right)$$

Step 2 (L_3)

$$\frac{\partial E}{\partial \vartheta_7} = x_7\left(1 - x_7\right)\left(\frac{\partial E}{\partial \vartheta_8} w_{87} + \frac{\partial E}{\partial \vartheta_9} w_{97}\right)$$

$$\frac{\partial E}{\partial \vartheta_6} = x_6\left(1 - x_6\right)\left(\frac{\partial E}{\partial \vartheta_9} w_{96}\right)$$

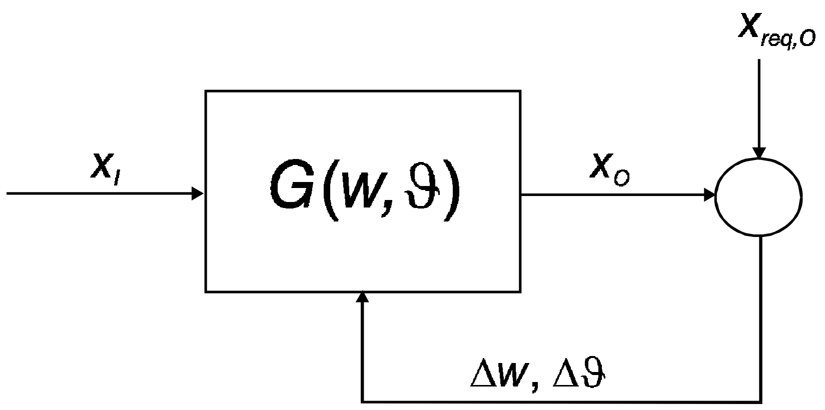
Step 3 (L_2)

$$\frac{\partial E}{\partial \vartheta_4} = x_4\left(1 - x_4\right)\left(\frac{\partial E}{\partial \vartheta_8} w_{84} + \frac{\partial E}{\partial \vartheta_7} w_{74}\right)$$

$$\frac{\partial E}{\partial \vartheta_5} = x_5\left(1 - x_5\right)\left(\frac{\partial E}{\partial \vartheta_7} w_{75} + \frac{\partial E}{\partial \vartheta_6} w_{65}\right)$$

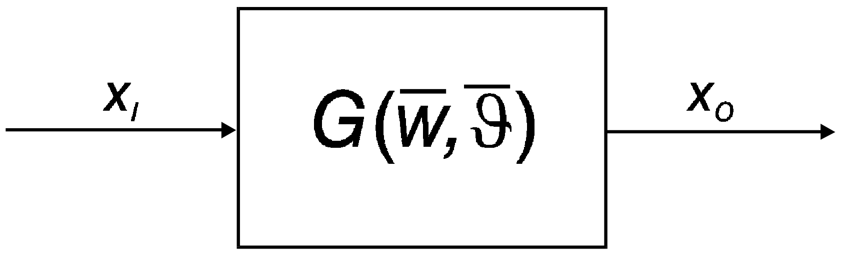
Adaptation phase

Weight and threshold coefficients are recurrently updated such that the best match between calculated and required output activities is achieved



Active phase

Weight and threshold coefficients are already "tuned" by the adaptation process, input activities are mapped onto output activities



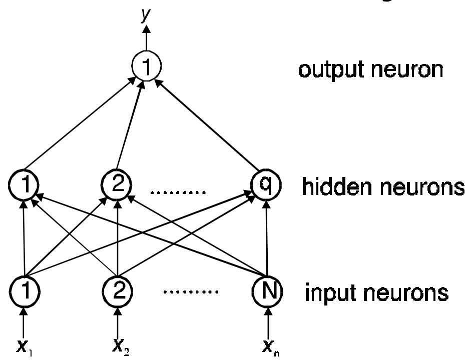
Neural networks are universal approximator

Let F be a function, which maps space R^N onto open interval (0,1)

$$F : R^N \rightarrow (0,1)$$

Training set (regression table) is composed of r points, $A_{\mathrm{train}}=\{\boldsymbol{x}_i/F(\boldsymbol{x}_i);i=1,2,..,r\}$

Theorem (Hoecht-Nielsen, Hornik). For each ε>0 there exists a 3-layer NN



represented by $G(\boldsymbol{x};\boldsymbol{w},\vartheta)$, such that

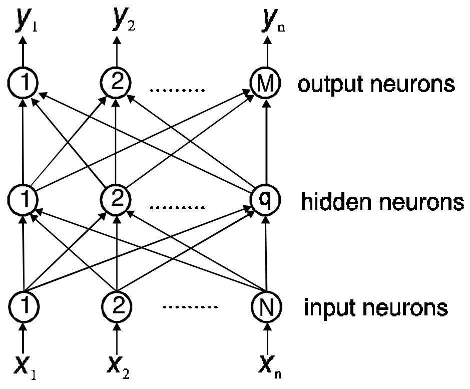
$$\sum_{i=1}^{r}\left(F\left(\boldsymbol{x}_i\right)-G\left(\boldsymbol{x}_i;\boldsymbol{w},\vartheta\right)\right(<\varepsilon$$

Note: This theorem is only an **existence** theorem, it does not specify the number of hidden neurons and actual values of weight and threshold coefficients.

The above theorem can be simply generalized for more general functions

$$F : R^N \rightarrow (0,1)^M$$

Then the corresponding NN is composed of *N* input neurons, *q* hidden neurons, and *M* output neurons.



The theorem is of the great importance for NN theory. It states that NNs may be considered as a **universal model-free regression tool**. Under the term "model-free" it is understood that there is not necessary to specify an analytical form of the model function, it is automatically constructed according to the chosen architecture of the used NN. In other words, **3-layer NNs are the universal approximator of functions.**

NNs as the classifier and/or predictor

An ability of NNs to **classify** objects and/or to **predict** object properties is the main subject of NN theory and its applications.

Let us consider two sets:

(1) Training set $A_{train} = \left\{ \boldsymbol{x}_I / \boldsymbol{x}_{req,O} \right\}$

(2) Test set $A_{test} = \left\{ \boldsymbol{x}_I / \boldsymbol{x}_{req,O} \right\}$

Both these sets are composed of different pairs of input activities (descriptors) and required output activities (required properties). The NN is adapted with respect to the training set A_{train}, an efficiency of the adapted NN to correctly predict/classify is tested by objects from the test set A_{test} . This approach is based on an assumption (or believe) that a "well" adapted NN is also able to "well" predict/ classify.

How **to realize a decomposition** of A onto A_{tran} and A_{test} ?

$$A = A_{train} \cup A_{test}$$

Training set A_{train} should contain only those objects of A that are good "representatives", i.e. it is expected that each object of A_{test} has at least one "counterpart" in A_{train}.

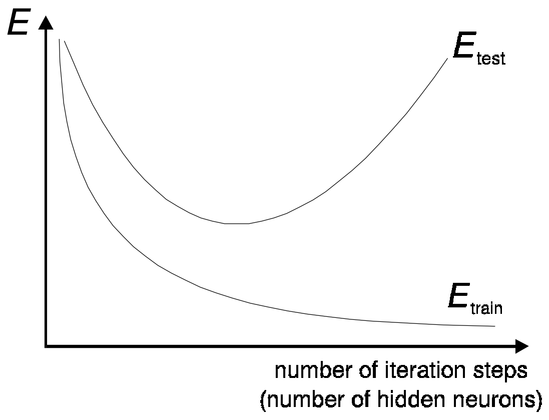
How **to adapt the NN** for a given decompo-sition of A onto A_{train} and A_{test} ?

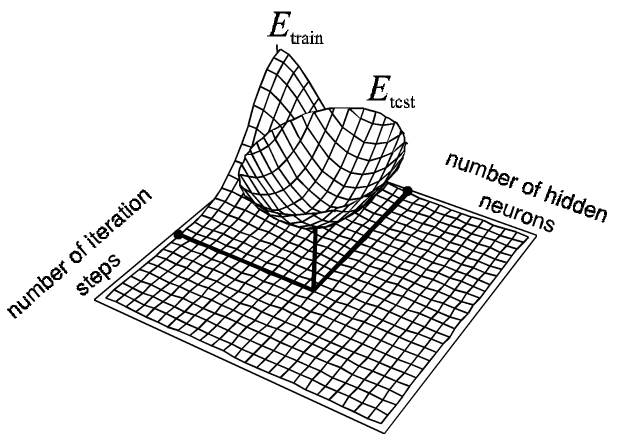
$$E_{train}\left(\boldsymbol{w},\vartheta\right)=\frac{1}{2}\sum_{A_{train}}\left(G\left(\boldsymbol{x}_I;\boldsymbol{w},\vartheta\right)-\boldsymbol{x}_{req,O}\right)^2$$

$$E_{test}\left(\boldsymbol{w},\vartheta\right)=\frac{1}{2}\sum_{A_{test}}\left(G\left(\boldsymbol{x}_I;\boldsymbol{w},\vartheta\right)-\boldsymbol{x}_{req,O}\right)^2$$

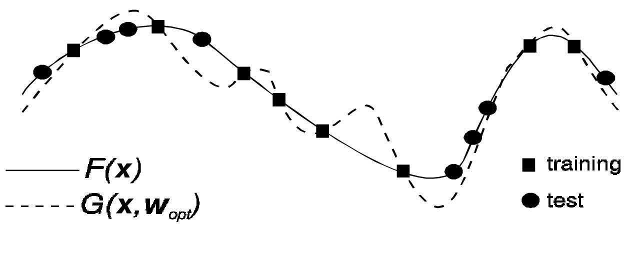
The adaptation process is performed with respect to A_{train}, i.e. E_{train} is minimized

If we plot values of E_{train} and E_{test} vs. numbers of iteration steps and hidden neurons, then we get the following two diagrams





Conclusion: There exist **optimal numbers** of iteration steps and hidden neurons, where the prediction ability of objects from the test set is the best. Increasing these numbers, then the prediction ability is worse. This observation is called the **overtraining** (**overfitting**) of NNs.



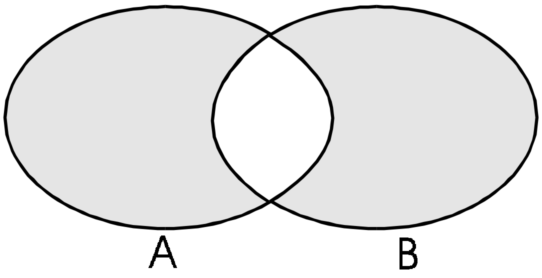
Illustrative examples

1. Boolean function XOR (playing an important role in the history of NNs)

XOR = eXclusive OR (verbally, either...or...)

Example: Symmetric difference of two sets is determined by

$$A - B = \left(A \cup B\right) \backslash \left(A \cap B\right)$$



It means than an element of the symmetric difference **either** belongs to A **or** B,

$$A - B = \left\{x; x \in A \quad \text{xor} \quad x \in B\right\}$$

x_1	x_2	x_1 **or** x_2
0	0	0
0	1	1
1	0	1
1	1	1

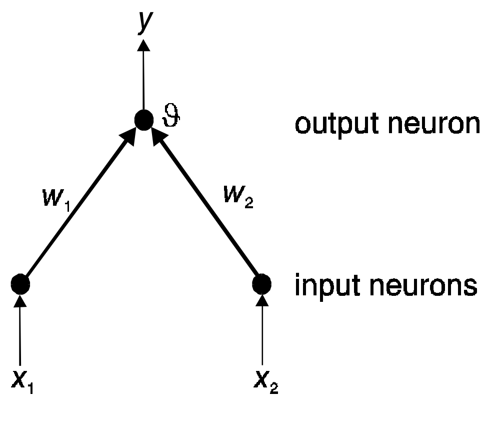
x_1	x_2	x_1 **xor** x_2
0	0	0
0	1	1
1	0	1
1	1	0

XOR is considered as a Boolean function

$$y = \mathrm{F}(x_1, x_2)$$

Goal: We look for such NN, which simulates this Boolean function

Simplest NNs without hidden neurons are **perceptrons**



Training set is determined as follows:

$$A_{train} = \{(0,0)/0,\ (0,1)/1,\ (1,0)/1,\ (1,1)/0\}$$

Output activity y is determined by

$$y = t\left(w_1 x_1 + w_2 x_2 + \vartheta\right)$$

The activity is closely related to one if argument (potential) of the transfer function is sufficiently great (Q) . In the opposite case, if the argument is sufficiently small (-Q), then the activity is closely related to zero.

For single objects of A_{train} we get

$$\begin{aligned} \vartheta &= -Q \quad \left((0,0)/0\right) \\ \vartheta + w_2 &= \ \ Q \quad \left((0,1)/1\right) \\ \vartheta + w_1 &= \ \ Q \quad \left((1,0)/1\right) \\ \vartheta + w_1 + w_2 &= -Q \quad \left((1,1)/0\right) \end{aligned}$$

where $t(Q)=1-\varepsilon\approx 1$ and $t(-Q)=\varepsilon\approx 0$, where ε is a small positive number. The above system does not have a solution (e.g. subtracting from the fourth equation the second and third equation we get $\vartheta=3Q$, while the first equation is $\vartheta=Q$).

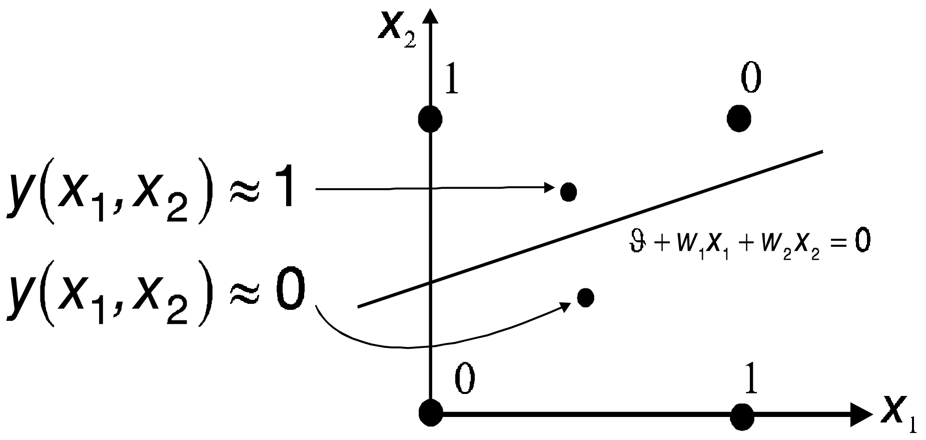
Conclusion: Simple percetron **is not able** to solve correctly Boolean function XOR.
How to interpret this result? Let us consider the equation

$$w_1x_1 + w_2x_2 + \vartheta = 0$$

It defines a straightline in plane x_1-x_2 , which divides plane onto two halfplanes. If a point (x_1,x_2) is placed above (below) the straightline, then

$$w_1x_1 + w_2x_2 + \vartheta > 0 \; (\text{point is above straightline})$$
$$w_1x_1 + w_2x_2 + \vartheta < 0 \; (\text{point is below straightline})$$

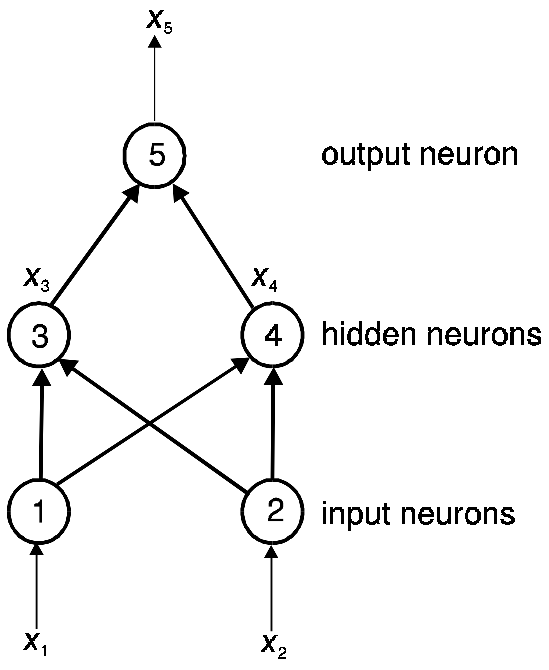
The corresponding activities of these two points are



Conclusion: We say that objects of XOR problems are **not linearly separable.** In general, perceptrons (i.e. NNs without hidden neurons) correctly classify only objects that are linearly separable.

From history of NN: In 1969 Minsky and Papert published seminal book *Perceptron*. One of Minsky and Papert's most discouraging results was that perceptrons are not able to solve problems (e.g. XOR) that are not linearly separable. What is very interesting, they discussed hidden neurons as a way how to overcome this serious drawback of perceptrons, but did not present a solution to the problem of how to adjust weight and threshold coefficients. An answer to this question was presented by Rumelhart et al. in 1986

XOR problem is correctly interpreted by 3-layer NN composed of **two hidden neurons** (Rumelhart et al. 1987)



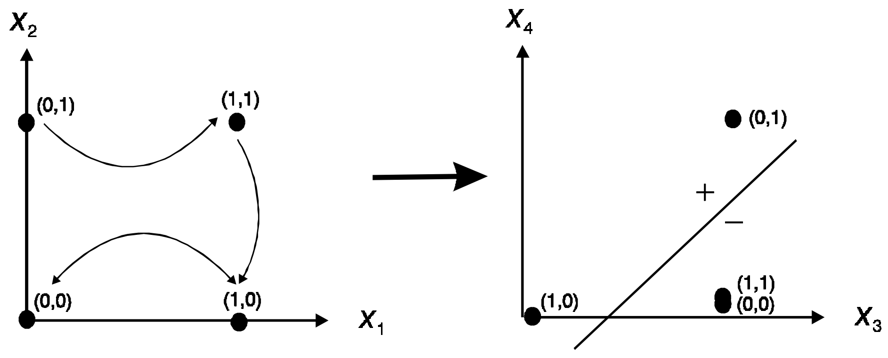
Training set of XOR problem

$$A_{train} = \{(0,0)/0\ ,\ (0,1)/1,\ (1,0)/1,\ (1,1)/0\}$$

Table of activities for different objects

No.	x_1	x_2	x_3	x_4	x_5	$x_{req,5}$
1	0.00	0.00	0.96	0.08	0.06	0.00
2	0.00	1.00	1.00	0.89	0.95	1.00
3	1.00	0.00	0.06	0.00	0.94	1.00
4	1.00	1.00	0.96	0.07	0.05	0.00

Hidden activities introduce new - **internal representation** of objects that is already linearly separable.



Conclusion: Hidden neurons in 3-layer NNs are very important for the correct interpretation of objects that are not linearly separable. Hidden neurons produce the so-called **internal representation** of objects, which is used by output neurons as input and it is processed by percetron-like approach to give correct results.

Conclusions

1. NNs with hidden neurons offer very **robust and effective numerical tool** for prediction of properties and/or classification of objects that are numerically determined by descriptors.

2. **Black-box** character of NNs is often mentioned in literature as the most serious drawback Recently, this aspect of NNs is very intensively studied by NN people. In particular, many different approaches how to "extract" symbolic information from the adapted NNs are suggested .

3. Adaptation process of NNs is usually **slightly lengthy**. Other more effective possibilities of optimization are studied, including stochastic optimization methods (see Pospichal's part of this series).

4. Implementation in C++ or Pascal is not very complicated. One of the best ways how to actively master NNs is to write a code of simple 3-layer NN. Do it!