Data Mining
(Mining Knowledge from Data)

Neural Networks

Marcel Jiřina, Pavel Kordík

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FIT

What are neural networks?

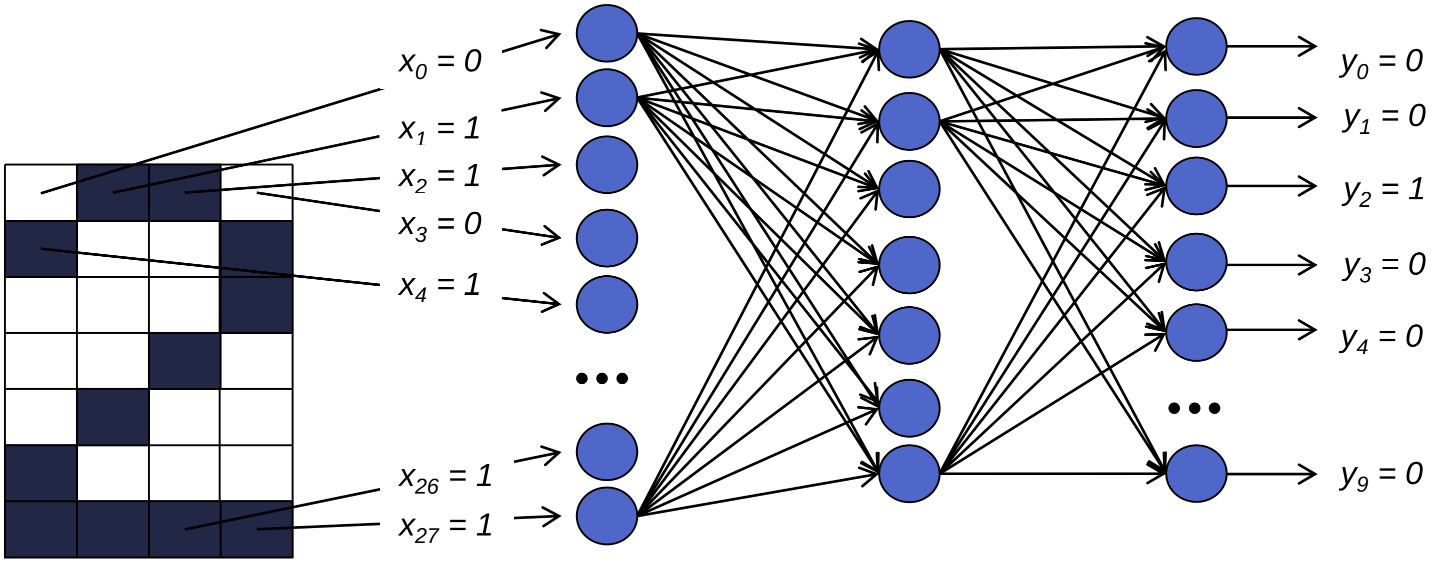
- Artificial Neural Network (ANN) is a computational model based on the connection of a large number of simple computational elements.
- Originally inspired by nature - neuronal connections in the nervous system of animals.

What does it serve for?

- Wide range of applications:
  - Classification
  - Regression
  - Clustering
  - Compression
  - Artificial Intelligence
  - Management
  - …

Examples of ANN

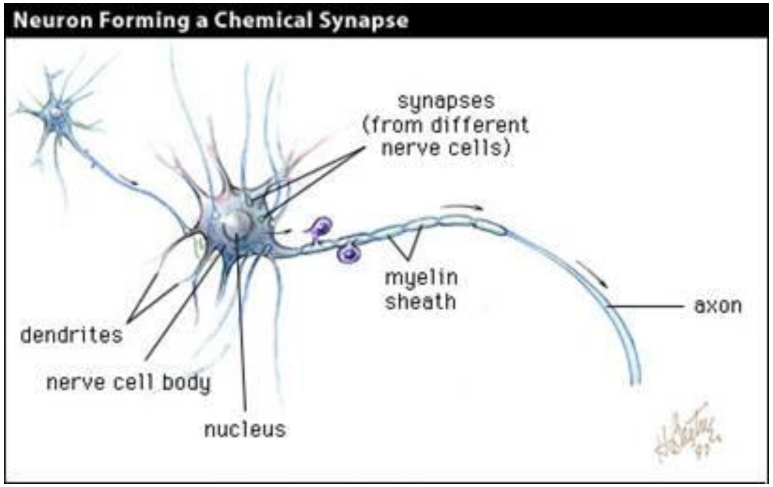
- ANN recognition of numbers:
  - Input: binary image 4x7 pixels
  - Output: the number in the figure in the code “1 of N”



Inspiration from biology

- The human brain contains about 10^{11} neurons.
- Each neuron is connected to 10^4 other neurons in average
- Neuron activation takes approx. 10^{-3} s (10^{-10} s compared with silicon chips)
- The brain is able to perform complex operations in a short time (face recognition 10^{-1} s)
- The longest signal path can be up over hundreds of neurons (max.)
- This is achieved by massively parallel architecture that mimics ANN

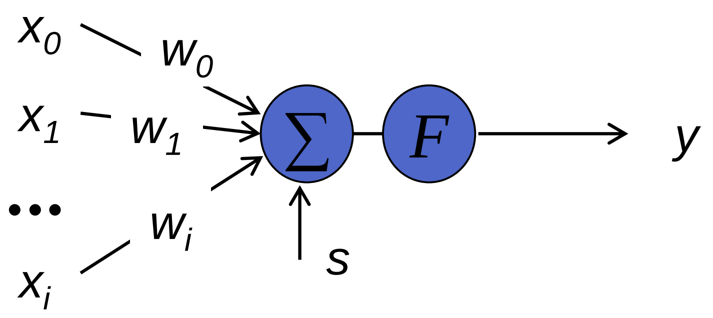
Human neuron



- In a neuron, a signal is received via synapses which are connected through dendrites to adjacent neurons
- A neuron sums the incoming signals and if the value exceeds a certain limit, the neuron creates tension (potential, voltage) which spreads over the axon to other neurons

Artificial neuron

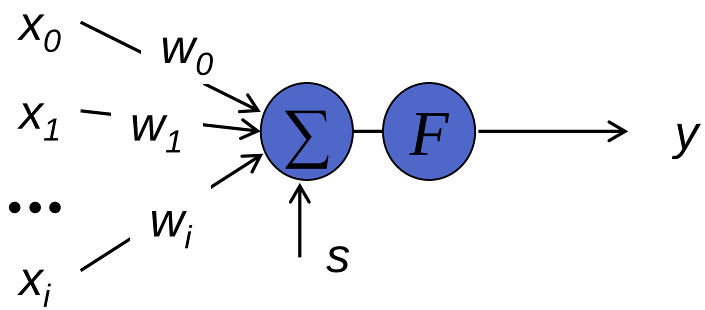
- The basic element of an artificial neural network is a neuron (also called processing element, unit)
- Each neuron consists of:
  - Several inputs $x_0 \dots x_i$
  - Weights $w_0 \dots w_i$ assigned to each input
  - Bias s
  - Activation function F
  - A single output y (can be lead to multiple neurons)



Neuron as a basic processing element

- The output of a neuron is a value of the activation function *F* applied to a weighted sum of inputs plus bias

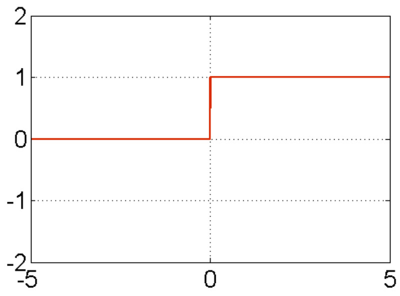
$$y = F\left(\sum_i (x_i \cdot w_i) + s\right)$$



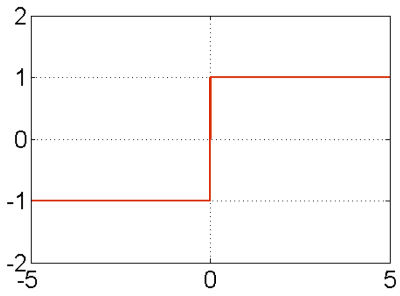
- There is a wide range of possible activation functions according to desired properties
  - There can be multiple kinds of activation functions within a single neural network

The most commonly used activation functions

- Heaviside function
  - binary – neuron is active(y = 1) or inactive (y = 0)
  - Used e.g. in the Perceptron

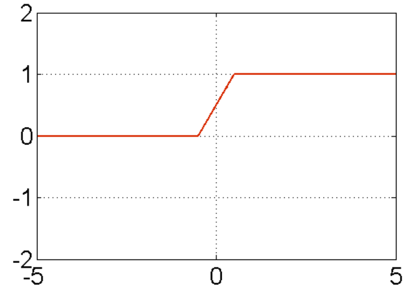


- Signum

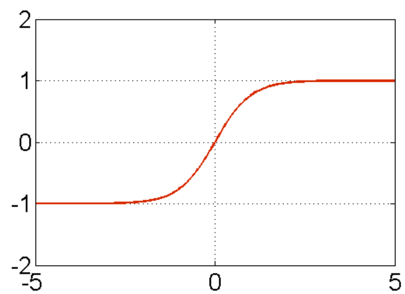


The most commonly used activation functions

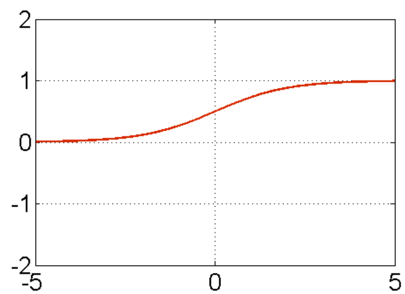
- Piecewise linear function



- Sigmoidal (logistic) function
  - limited function resembling letter S
  - $\mathrm{F}(x) = \frac{1}{1+e^{-x}}$ range (0; 1)
  - Similar to function tanh(x) }same shape, different range (-1; +1)
  - Used e.g. in MLP (Multilayer Perceptron)



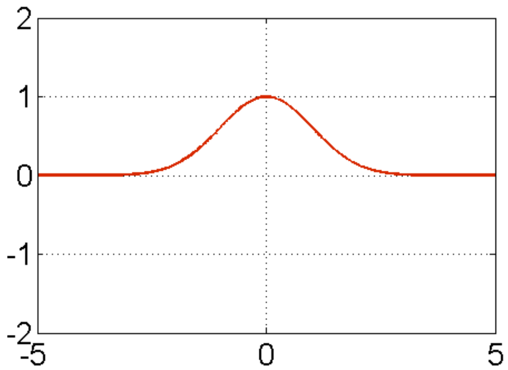
tanh



logistic function

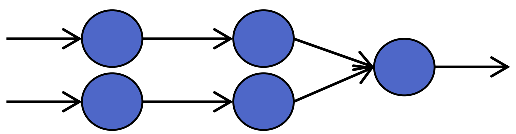
The most commonly used activation functions

- Bell curve (Gaussian function)
  - $e^{-\frac{(x-a)^2}{2\sigma}}$
  - Used e.g. in RBF, SOFM

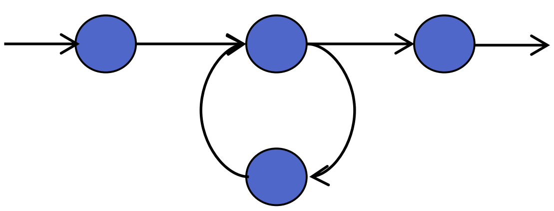


ANN structure

- According to the structure, the ANN can be divided to:
  - Forward neural networks (feedforward, FF)
    - ➢ include feedback
    - ➢ signal propagates in one direction from inputs to outputs only

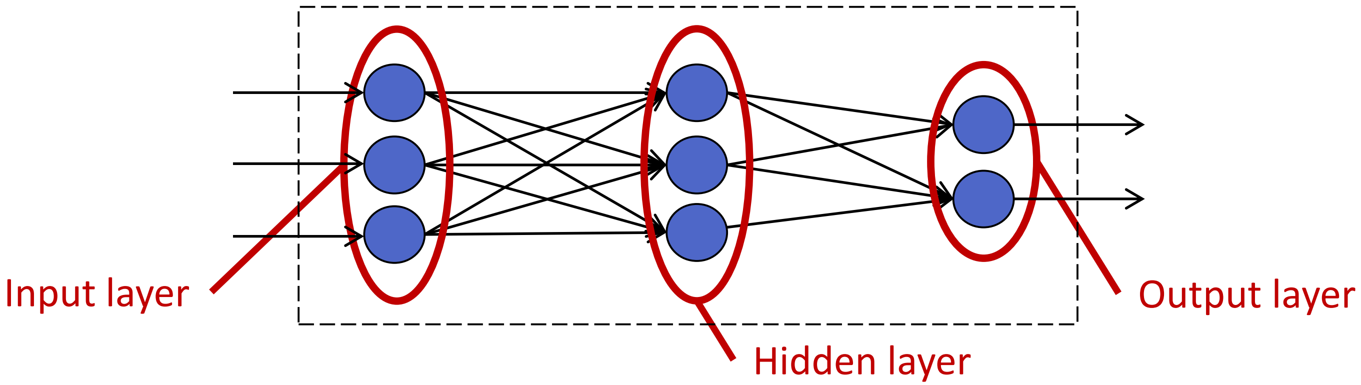


  - Recurrent neural networks
    - ➢ include a feedback



ANN as a graph

- The structure of a neural network can be often expressed by an oriented graph:
  - Nodes represent neurons
  - Edges represent connections of neuron outputs to inputs of another neurons
  - Usually the ANN is drawn in the form of several layers of neurons with the same characteristics
  - Orientation of edges in feedforward ANNs is usually not drawn – it is assumed that inputs are at the left and outputs at the right



ANN layers

- The task of the input layer is to promote the neural network inputs to other layers
  - The number of neurons in the input layer is determined by the number of attributes in the training set
- The output layer
  - # of neurons = # model outputs (in proper coding, e.g. “1 of N”)

Modes of ANN

- Neural networks operate in two modes:
  - Learning/training
    - Adjusting values of weights and the bias (and possibly the structure)
  - Recall/use
    - The already learned network predicts output value on the basis of a given instance

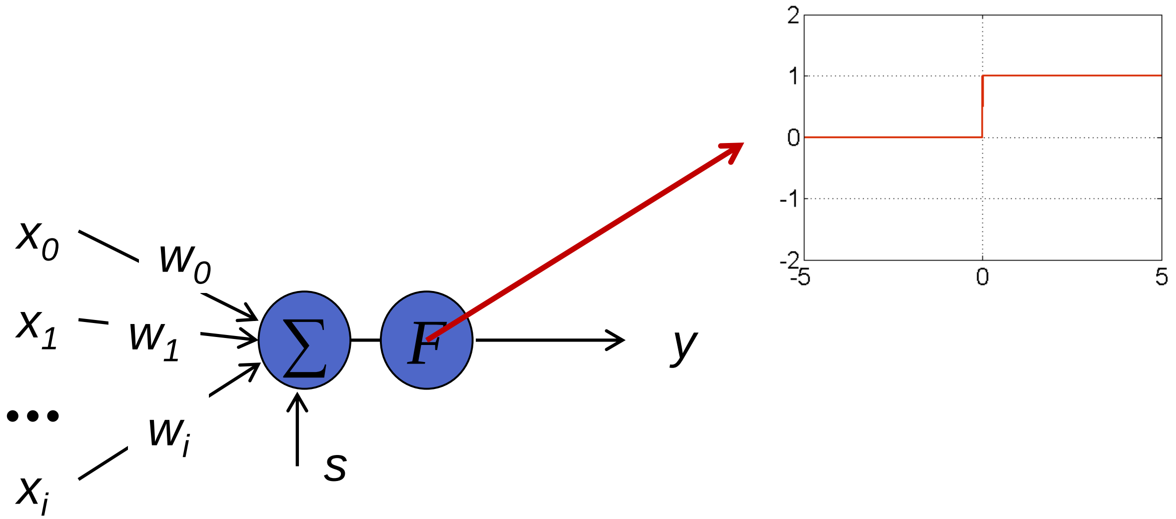
Learning/training

- Learning/training can be either with a teacher or without a teacher (supervised/unsupervised)
  - Samples (patterns, instances, cases) from the training set are presented to a neural network, and the neural network accordingly adjusts the weights and bias (possibly structure)

- Each instance of the training set is mostly used multiple times during the learning stage (mode)
- The use of all instances of the training set just once is called "epoch"

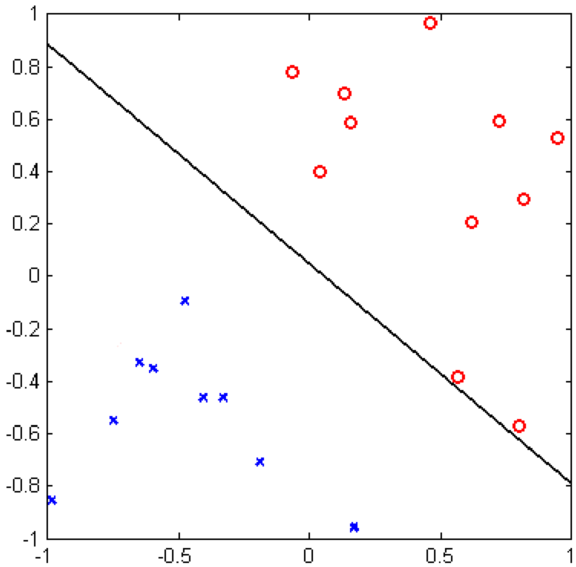
ANN - Perceptron

- Frank Rosenblatt, 1957
- single-layer neural network + training algorithm
- The Heaviside function is used as the activation function



Perceptron

- $y = \begin{cases} 1\,, & \sum_i (x_i \cdot w_i) + s > 0 \\ 0\,, & \boxed{\sum_i (x_i \cdot w_i) +} s \leq 0 \end{cases}$ linear combination of inputs
- A hyperplane is used as the discrimination/decision border (a line for 2 inputs)



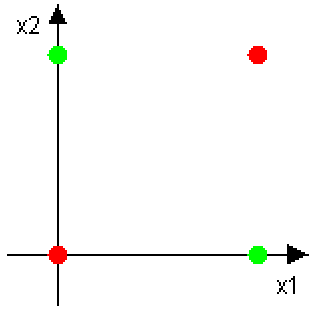
Perceptron - training

- 1. initialize weights $w_0 - w_n$ and bias s
  - Set them to small random values (e.g. from the range -0.3 to 0.3)
- 2. for each instance *j* from the training set:
  - Calculate output of a neuron $y = \sum_i (x_i \cdot w_i) + s$
  - Adjust the weights:
    - $w_i(t+1) = wi(t) + \alpha(\mathrm{y} - \hat{\mathrm{y}})\mathrm{x}_i$
- 3. Repeat step 2 until the error is sufficiently low

- α is so called *learning rate* and represents the speed of learning
  - Lower values indicate slower convergence
  - At a high value the optimum can be skipped and the algorithm may not converge
  - α can be gradually reduced

Perceptron

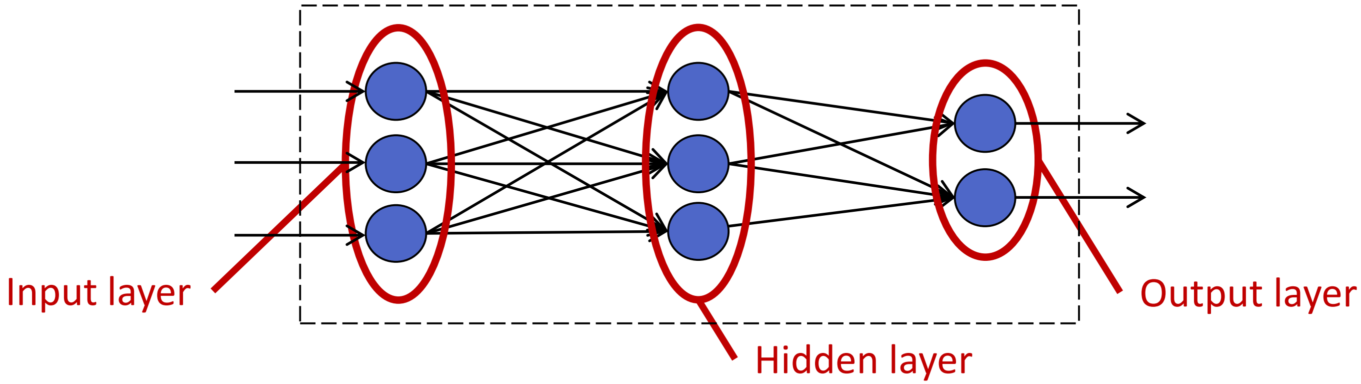
- The learning algorithm of the Perceptron works for linearly separable data only (= can be perfectly classified by a hyperplane)
- For linearly inseparable data the algorithm does not converge
- E.g. XOR:



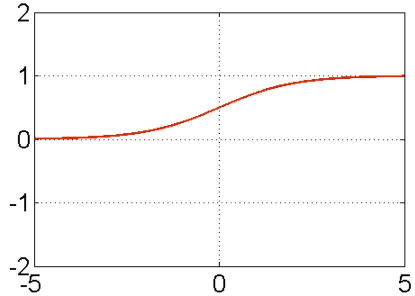
- See applet: http://lcn.epfl.ch/tutorial/english/perceptron/html/index.html

ANN - MLP

- MLP = Multi-layer perceptron
- A multilayer feedforward neural network
  - Number of layers ≥ 3, there can be multiple hidden layers



- The activation function is usually the logistic sigmoid
  - $F(sum) = \frac{1}{1+e^{-sum}}$

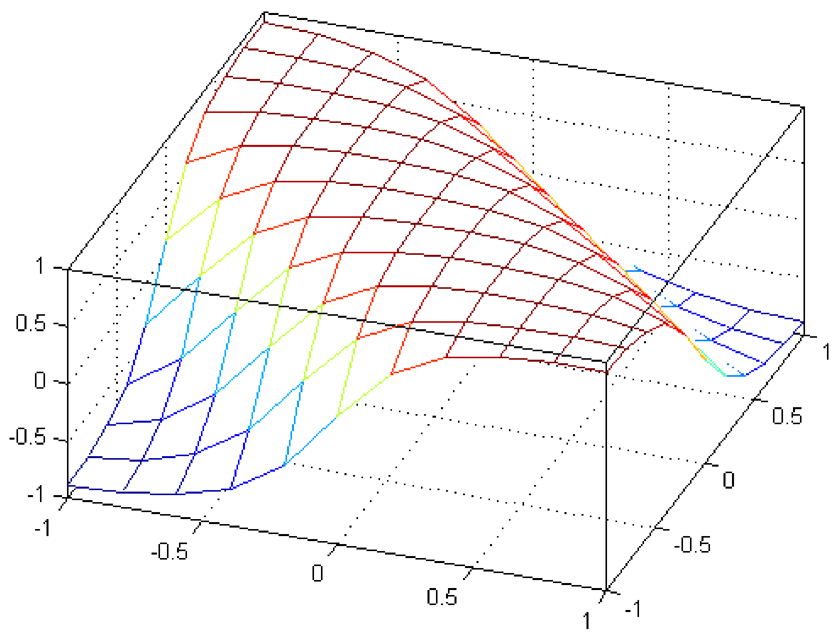


MLP

- The decision boundary is nonlinear
- The more neurons in the hidden layers, the more complex shape of the data can be expressed
- By using the sigmoid function the output in not binary any more, but is from continuous interval (0; 1)
- The value corresponds to the degree of membership to which the given class should belong

MLP - XOR

- Solving the XOR problem using MLP:



MLP training

- Backpropagation (backward propagation of error)
- Error minimization $Err = \frac{1}{2}\sum_j (y - out)^2$, where *j* are neurons of the input layer
- 2 stages:
  - Calculating outputs of neurons in all layers
  - Backward propagation of error – adjusting the weights starting from the output layer toward the input layer

Backpropagation – pseudocode

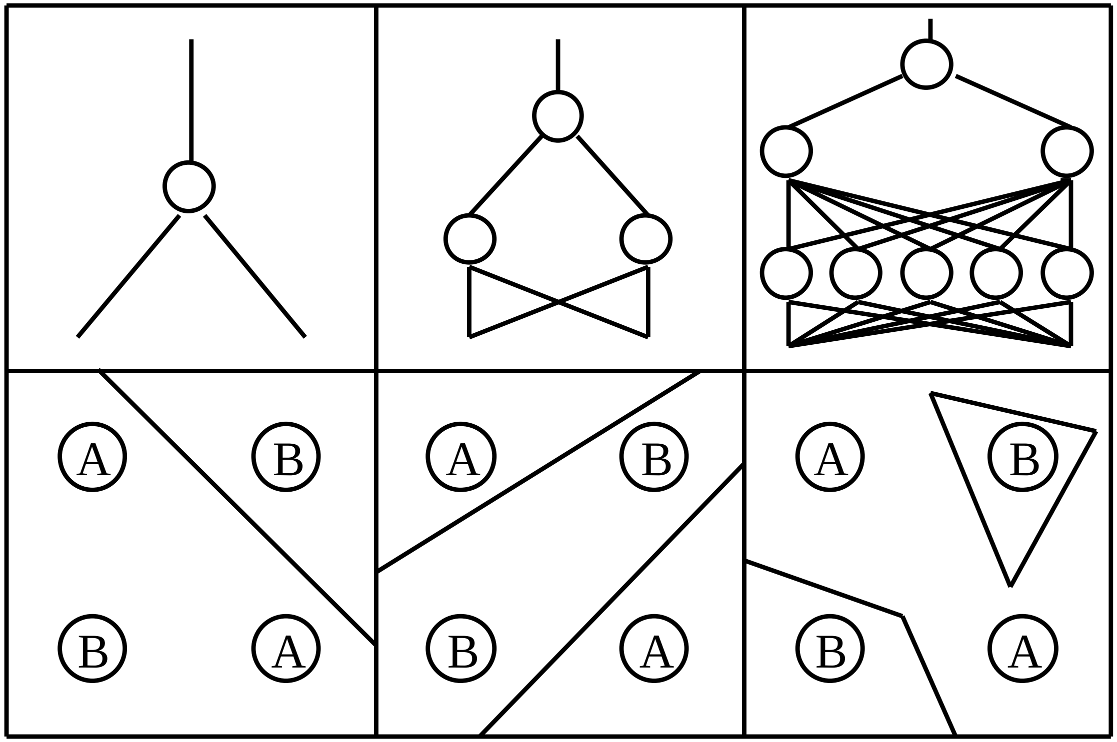
1. Initialize the weights to small random values
   - E.g. from the interval (-0.3,0.3)
2. Repeat until the stopping condition is fulfilled

   For all training data (epoch)

   1. Randomly select instance *[X,y]* from the training data set
      1. Calculate output out_u for each neuron
      2. For each neuron *v* in the input layer calculate error
         $$Err_v = out_v(1 - out_v)(y - out_v)$$
      3. For each neuron *s* in the hidden layer calculate error
         $$Err_s = out_s(1 - out_s)\sum_{v \in výstup}(w_{s,v} \cdot Err_v)$$
      4. For each connection from neuron *j* to *k* adjust weight $w_{j,k} = w_{j,k} + \Delta w_{j,k}$, where $\Delta w_{j,k} = \eta Err_k x_{j,k}$

Multilayer Perceptron Network

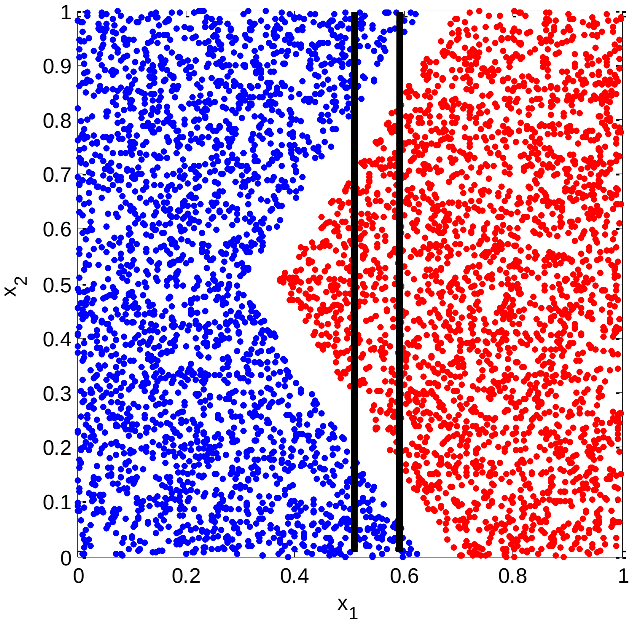
Structure of the network with respect to the problem solved



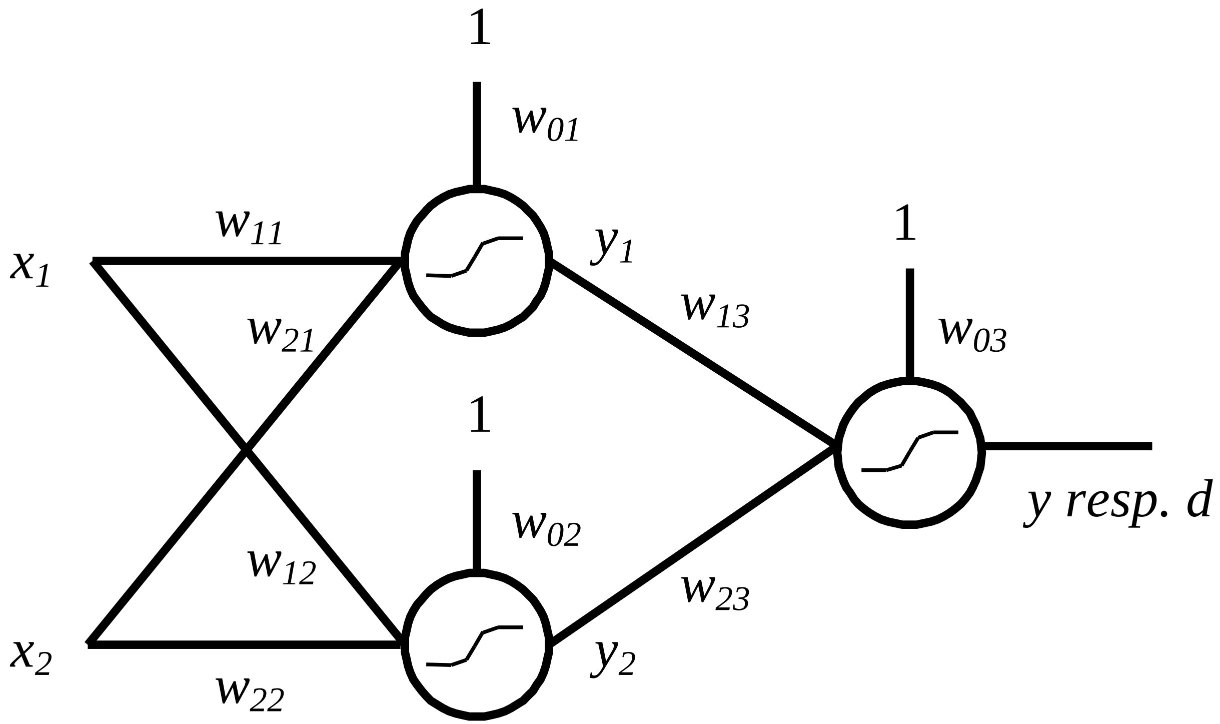
Multilayer Perceptron Network

- Example: Training Multilayer Perceptron Network

x1	x2	třída
0.8680	0.2640	1
0.1790	0.4230	0
0.6940	0.1630	1
0.5380	0.6450	1
0.6230	0.0030	0
0.7870	0.2680	1
0.4610	0.3860	1
0.6030	0.2790	1
0.1700	0.8050	0
...	...	...



Multilayer Perceptron Network



Multilayer Perceptron Network

$\xi_1 = w_{01} \cdot 1 + w_{11} \cdot x_1 + w_{21} \cdot x_2$ Post-synaptic potential of the first node

$y_1 = f(\xi_1)$ Output from the first node

$\xi_2 = w_{02} \cdot 1 + w_{12} \cdot x_1 + w_{22} \cdot x_2$ Post-synaptic potential of the second node

$y_2 = f(\xi_2)$ Output from the second node

$\xi_3 = w_{03} \cdot 1 + w_{13} \cdot y_1 + w_{23} \cdot y_2$ Post-synaptic potential of the third node

$y = f(\xi_3)$ Output from the third node (output from the network)

Multilayer Perceptron Network

$$\delta = y(1-y)(d-y)$$

$$w_{03} = w_{03} + \eta \cdot \delta \cdot 1,$$
$$w_{13} = w_{13} + \eta \cdot \delta \cdot y_1,$$
$$w_{23} = w_{23} + \eta \cdot \delta \cdot y_2,$$

$$\delta_1 = y_1(1-y_1) \cdot \delta \cdot w_{13}$$

$$\delta_2 = y_2(1-y_2) \cdot \delta \cdot w_{23}$$

Multilayer Perceptron Network

$$\xi_1 = w_{01} \cdot 1 + w_{11} \cdot x_1 + w_{21} \cdot x_2 = 0\,,$$
$$\xi_2 = w_{02} \cdot 1 + w_{12} \cdot x_1 + w_{22} \cdot x_2 = 0\,.$$

$$x_2 = -\frac{w_{11}}{w_{21}} \cdot x_1 - \frac{w_{01}}{w_{21}}\,,$$
$$x_2 = -\frac{w_{12}}{w_{22}} \cdot x_1 - \frac{w_{02}}{w_{22}}\,,$$

$$k_1 = -\frac{w_{11}}{w_{21}}\,,\; k_2 = -\frac{w_{12}}{w_{22}}$$

$$q_1 = -\frac{w_{01}}{w_{21}}\,,\; q_2 = -\frac{w_{02}}{w_{22}}$$

Multilayer Perceptron Network