Hybrid Intelligent Systems

Lecture 5

Computational Intelligence.

Neural Networks. Feed Forward NN

Definition of Computational Intelligence (Wikipedia)

- **Computational intelligence (CI)** is a successor of artificial intelligence. As an alternative to GOFAI[1] it rather relies on heuristic algorithms such as in Fuzzy systems, Neural networks and Evolutionary computation. In addition, computational intelligence also embraces techniques that use Swarm intelligence, Fractals and Chaos Theory, Artificial immune systems, Wavelets, etc.
- Computational intelligence combines elements of learning, adaptation, evolution and Fuzzy logic (rough sets) to create programs that are, in some sense, intelligent. Computational intelligence research does not reject statistical methods, but often gives a complementary view (as is the case with fuzzy systems). Artificial neural networks is a branch of computational intelligence that is closely related to machine learning.
- Computational intelligence is further closely associated with soft computing, connectionist systems and cybernetics.

1. **GOFAI** stands for **"Good Old-Fashioned Artificial Intelligence"**. It is commonly used to denote a branch of Artificial Intelligence which mainly deals with symbolic problems

Also definition

- Computational Intelligence is set of paradigms and techniques inspired by simulation of natural processes dealing with information

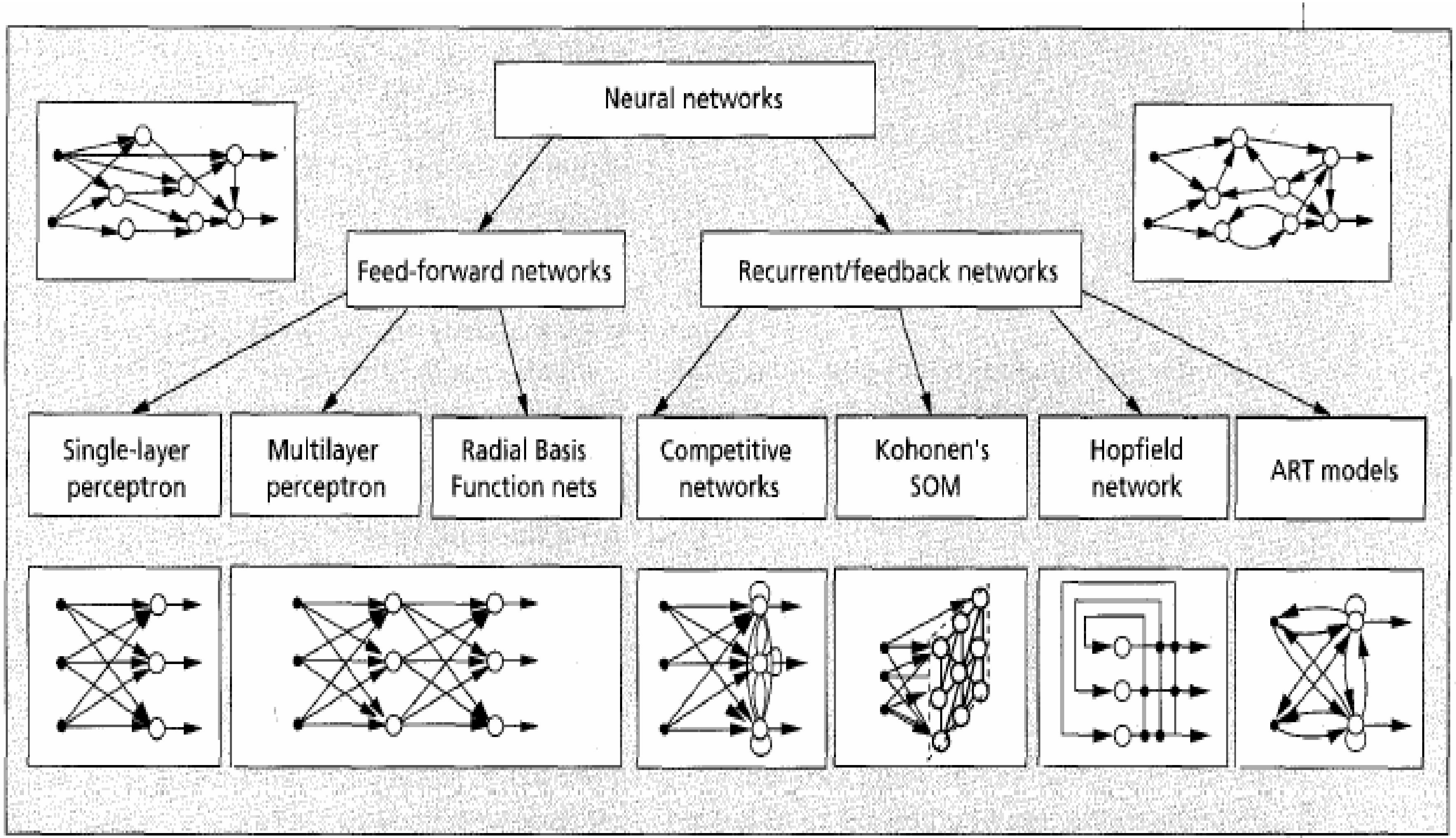
Points of view on artificial neural networks

- Model of real natural neural networks in brain
  - For example, spike neurons
- Model of parallel processing of information based on adaptation (learning) (neuron is function with changed parameters)
  - MLP, SOM, RBF
- Algorithms of nonlinear processing of information with representation of input and output information as net
  - For example, Hemming net, Bayesian nets

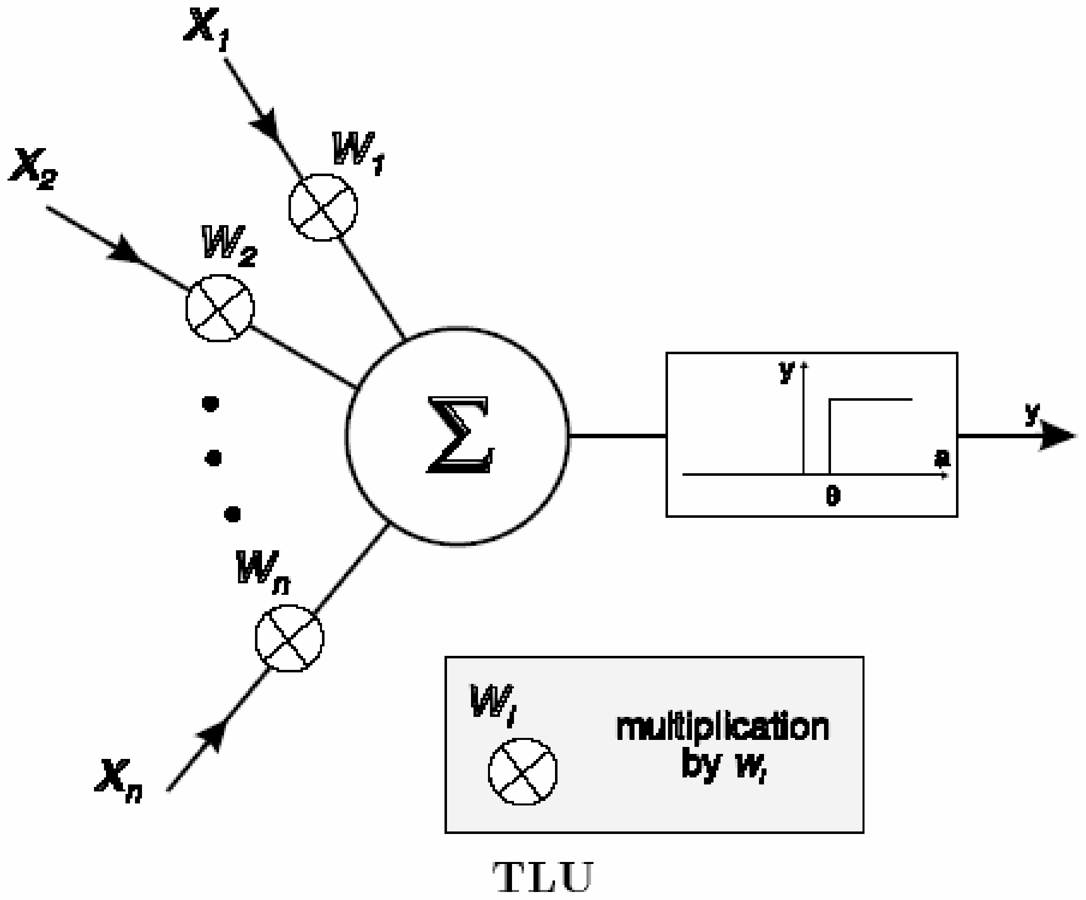
Approaches to use of neural networks for solving of tasks

- Use of service customers for solving of task
- Use of task-oriented neural network shell
  - Metastock, Predictor, Deductor
- Use of universal neural networks shell
  - Toolbox of MATLAB, Statistica Neural Networks, Brain Maker, Neural Planner, Neural Bench,
- Implementation of program model of neural network as separate program (with or without any components for programming of NN – SNNS,)
- Implementation of program model of neural network as component of any applied software
  - G2
- Programming of hardware neural network

Taxonomy of feed-forward and recurrent networks



Formal neuron by MacCallock-Pitts

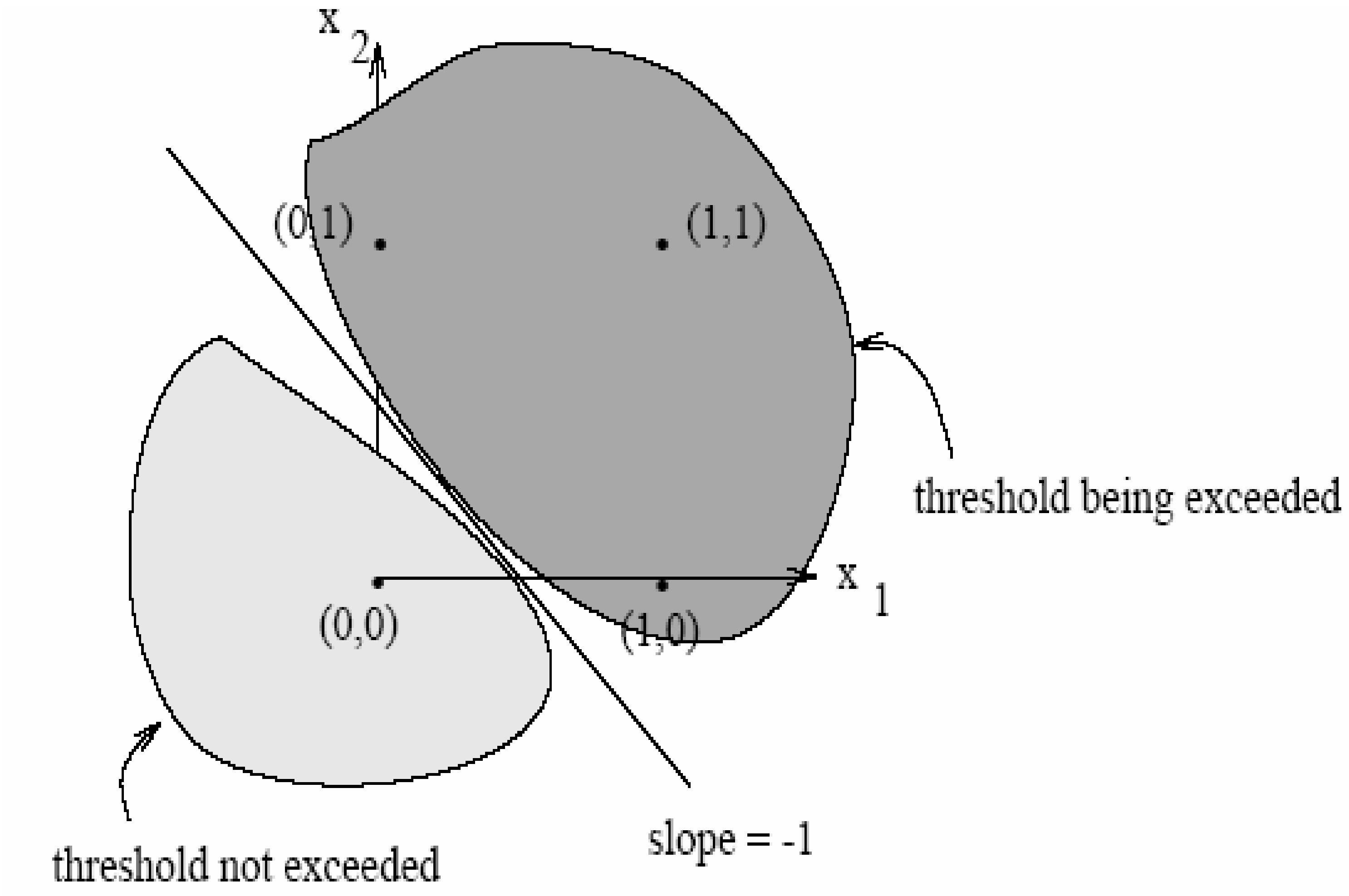


x_i – binary signal

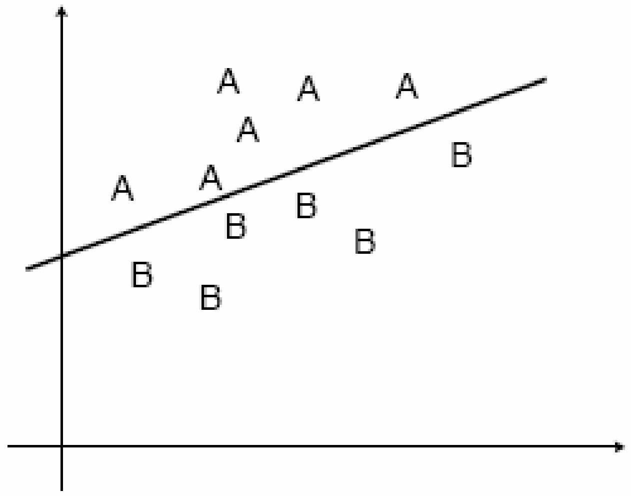
$$a = \sum_{i=1}^{n} w_i x_i$$

$$y = \begin{cases} 1 & \text{if} \quad a \geq \theta \\ 0 & \text{if} \quad a < \theta \end{cases}$$

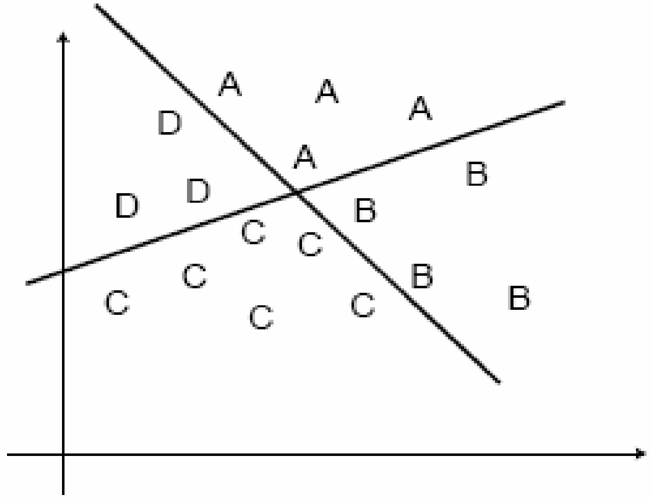
Classification by TLU



Classification

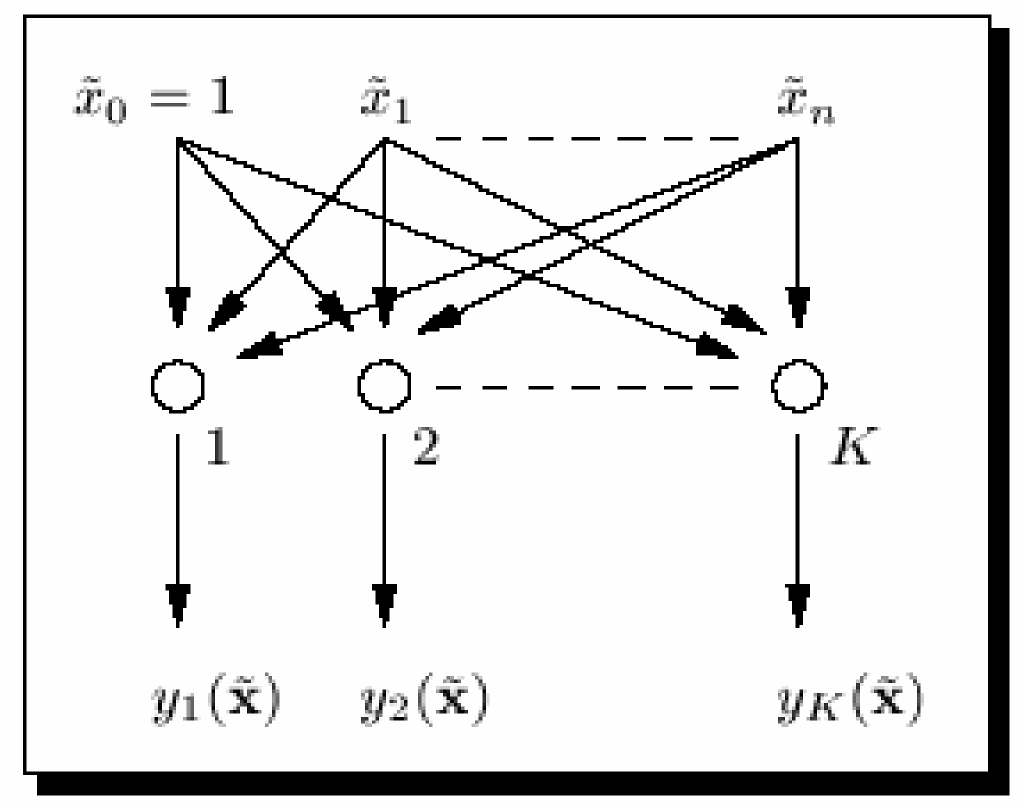


A/B classification

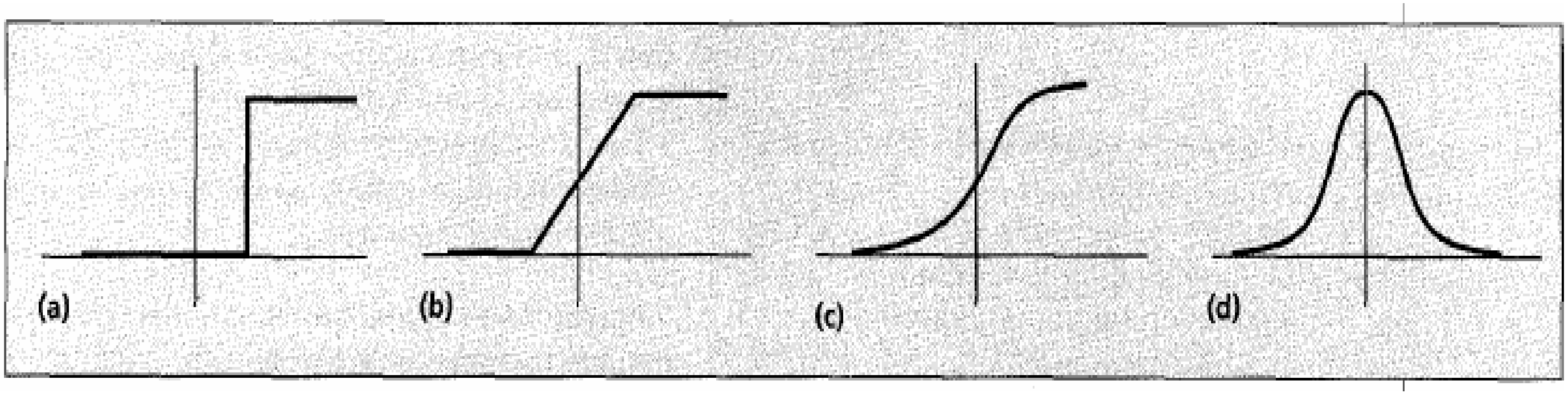


pattern space for A B C D

A single layer of neurons may act as a linear classifier for K classes


$\tilde{x}_0 = 1$
$\tilde{x}_1$
$\tilde{x}_n$
1
2
K
$y_1(\tilde{\mathbf{x}})$
$y_2(\tilde{\mathbf{x}})$
$y_K(\tilde{\mathbf{x}})$

Different kinds of activation function



Threshold, picewise linear, sigmoid, Gaussian

Kinds of sigmoid

Exponential

$$f(s) = \frac{1}{1 + e^{-2\alpha s}}$$

Rational

$$f(s) = \frac{s}{|s| + \alpha}$$

Hyperbolic tangent

$$f(s) = th\frac{s}{\alpha} = \frac{e^{-\frac{s}{\alpha}} - e^{-\frac{s}{\alpha}}}{e^{\frac{s}{\alpha}} + e^{-\frac{s}{\alpha}}}$$

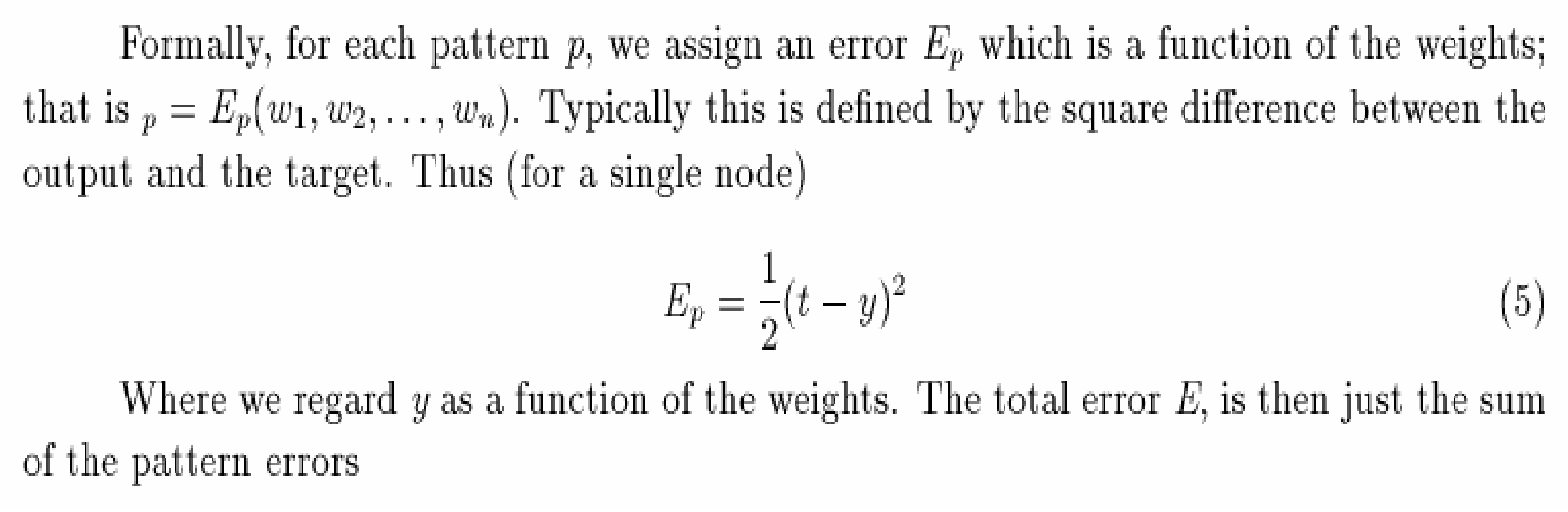
Different approaches to learning

- Error-correction rules
  - Use error (d-y) between desired and real output to modify the connections weights to reduce this error
- Boltzmann learning
  - To adjust the connections weights so that states of visible units satisfy a particular desired probability distribution
- Hebbian rule – strengthening connections between nodes with similar states and vise versa

$$w_{ij}(t+1) = w_{ij}(t) + \eta\, y_j(t)\, x_i(t)$$

- Competitive learning rules
  - Only one output is activate in any time (*winner-take-all*)

Error in error-correction rules



Formally, for each pattern p, we assign an error E_p which is a function of the weights; that is $_p = E_p(w_1, w_2, \ldots, w_n)$. Typically this is defined by the square difference between the output and the target. Thus (for a single node)

$$E_p = \frac{1}{2}(t - y)^2 \tag{5}$$

Where we regard y as a function of the weights. The total error E, is then just the sum of the pattern errors

$$E = \sum_p E_p$$

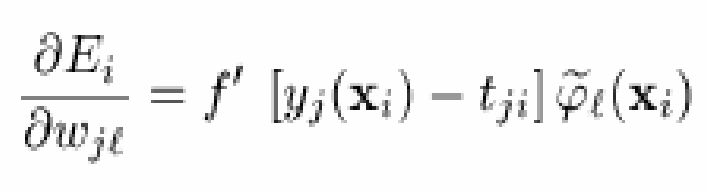
$$E_p = \frac{1}{2} \sum_{j=1}^{N} (t^j - y^j)^2$$

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{K} [y_j(\mathbf{x}_i, \mathbf{w}) - t_{ji}]^2$$

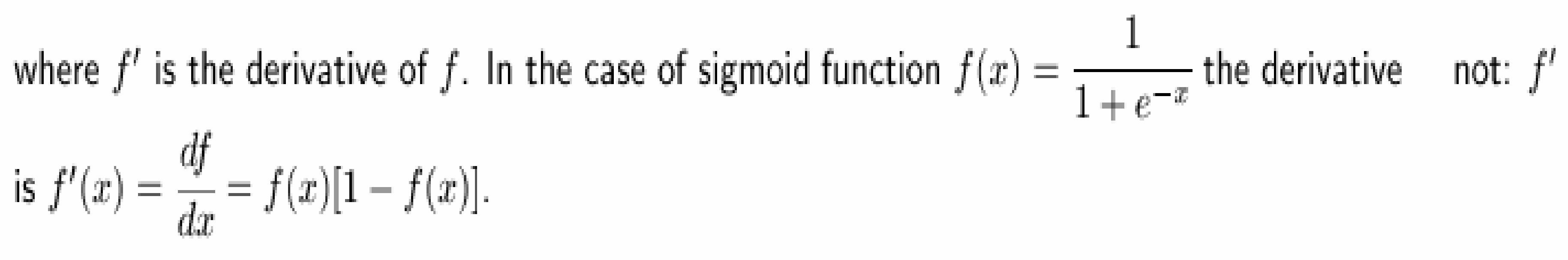
or

$$E = \frac{1}{2} \sum_{j=1}^{K} \|\vec{y}_j - \vec{t}_j\|^2 = \frac{1}{2} \sum_{j=1}^{K} \sum_{i=1}^{N} (\{\vec{y}_j\}_i - t_{ji})^2$$

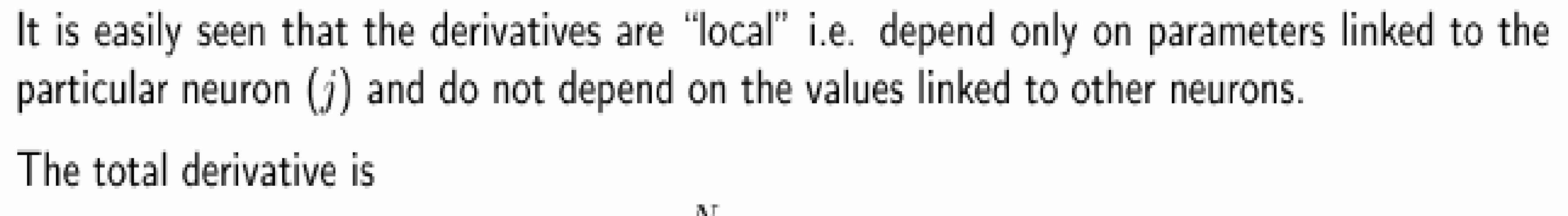
Error in error-correction rules (Cont.)



$$\frac{\partial E_i}{\partial w_{j\ell}} = f' \;\left[y_j(\mathbf{x}_i) - t_{ji}\right] \widetilde{\varphi}_\ell(\mathbf{x}_i)$$

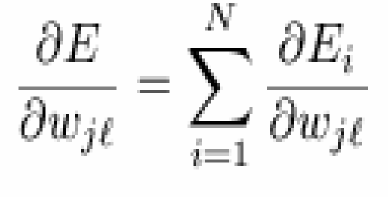


where f' is the derivative of f. In the case of sigmoid function $f(x) = \dfrac{1}{1+e^{-x}}$ the derivative not: f' is $f'(x) = \dfrac{df}{dx} = f(x)[1 - f(x)]$.



It is easily seen that the derivatives are “local” i.e. depend only on parameters linked to the particular neuron (j) and do not depend on the values linked to other neurons.

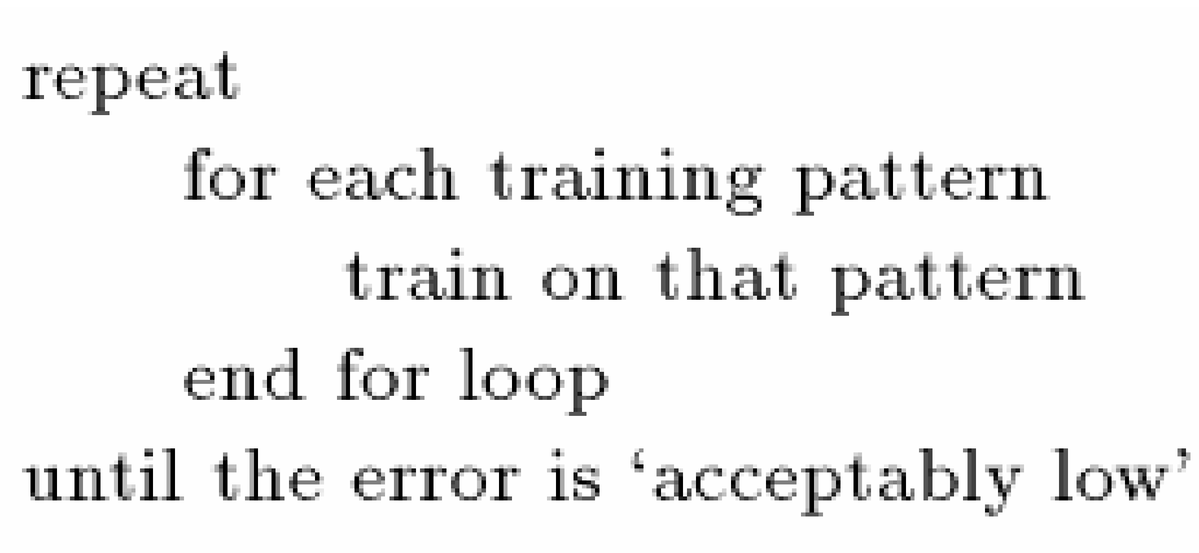
The total derivative is



$$\frac{\partial E}{\partial w_{j\ell}} = \sum_{i=1}^{N} \frac{\partial E_i}{\partial w_{j\ell}}$$

Feed forward Neural Networks (MLP)

- Main algorithm of training



```
repeat
    for each training pattern
        train on that pattern
    end for loop
until the error is ‘acceptably low’
```

Formulas for error back propagation algorithm

Modification of weights of synapses of *j*th neuron connected with *i*th ones, x_j – state of *j*th neuron (output)

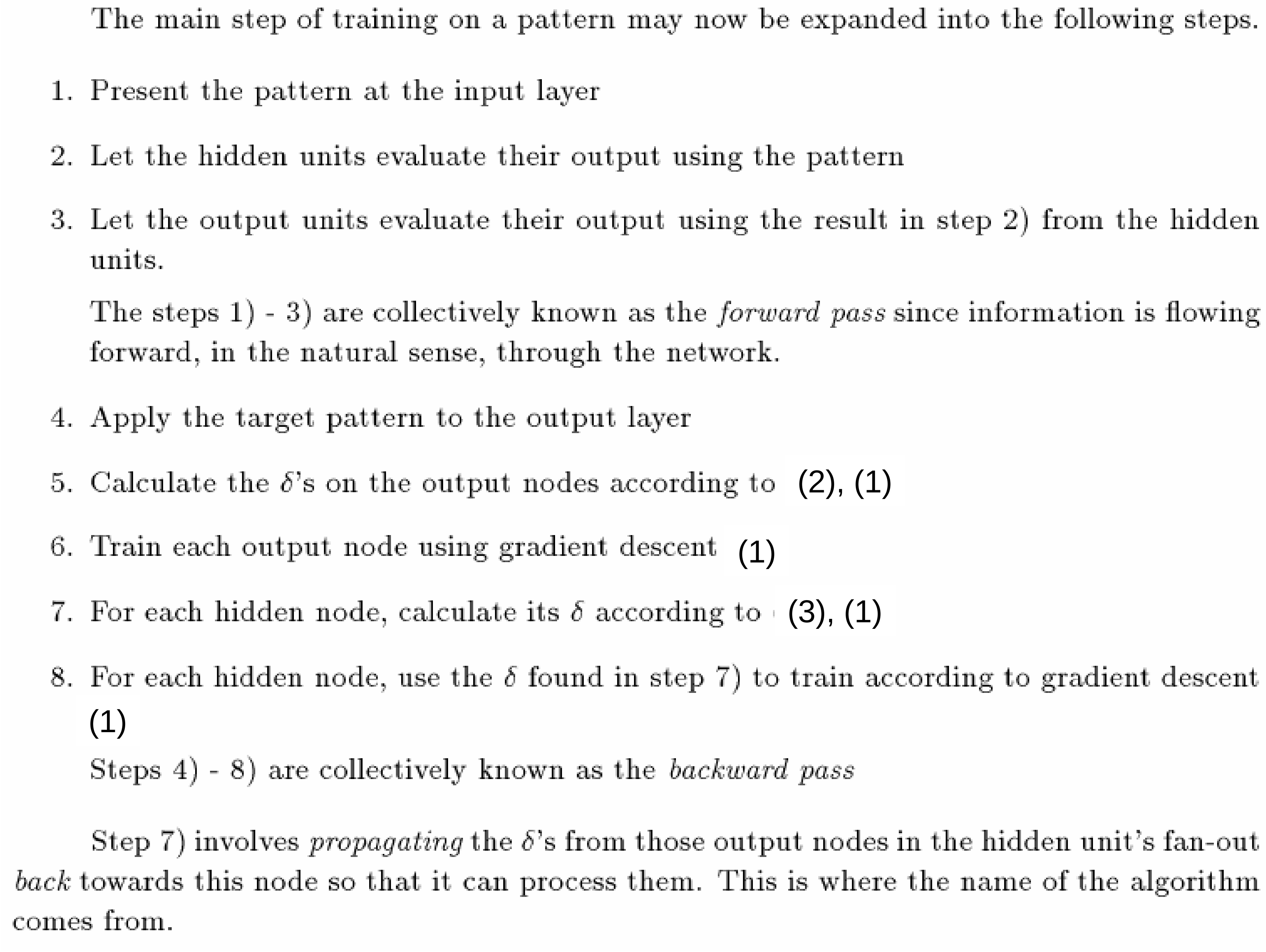
$$w_{ij}(t+1) = w_{ij}(t) + r g_j x'_i \qquad (1)$$

For output layer

$$g_j = y_j(1-y_j)(d_j-y_j) \qquad (2)$$

For hidden layers
k – number of neuron in next layer connected with *j*th neuron

$$g_j = x'_j(1-x'_j)\sum_k g_k w_{jk} \qquad (3)$$



The main step of training on a pattern may now be expanded into the following steps.

1. Present the pattern at the input layer

2. Let the hidden units evaluate their output using the pattern

3. Let the output units evaluate their output using the result in step 2) from the hidden units.

   The steps 1) - 3) are collectively known as the *forward pass* since information is flowing forward, in the natural sense, through the network.

4. Apply the target pattern to the output layer

5. Calculate the δ's on the output nodes according to (2), (1)

6. Train each output node using gradient descent (1)

7. For each hidden node, calculate its δ according to (3), (1)

8. For each hidden node, use the δ found in step 7) to train according to gradient descent (1)

   Steps 4) - 8) are collectively known as the *backward pass*

Step 7) involves *propagating* the δ's from those output nodes in the hidden unit's fan-out *back* towards this node so that it can process them. This is where the name of the algorithm comes from.

Generalization

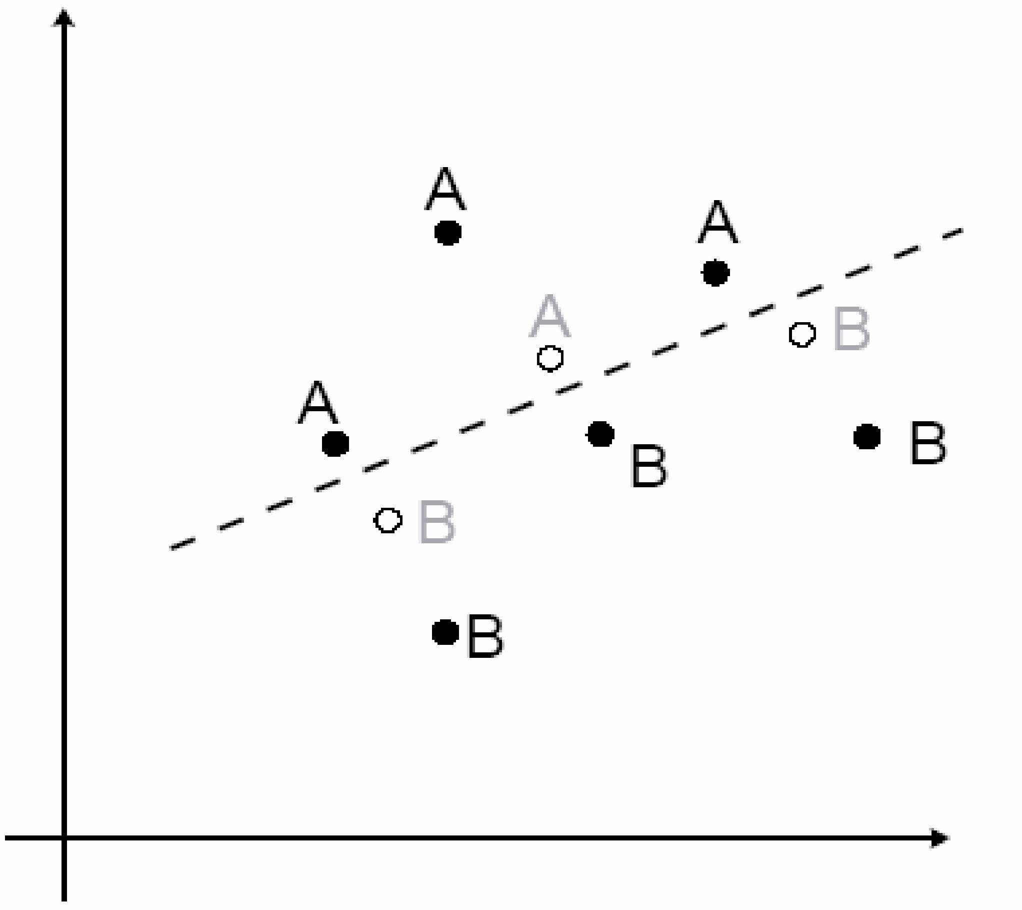
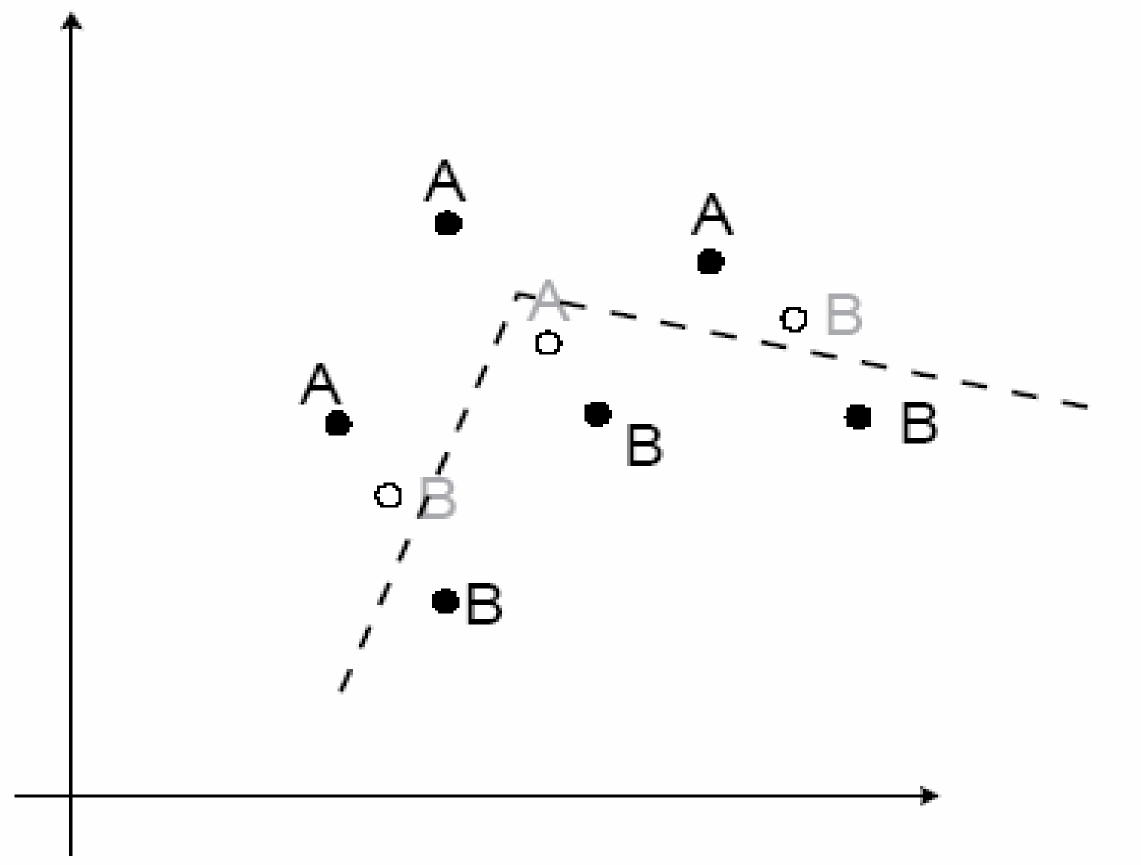


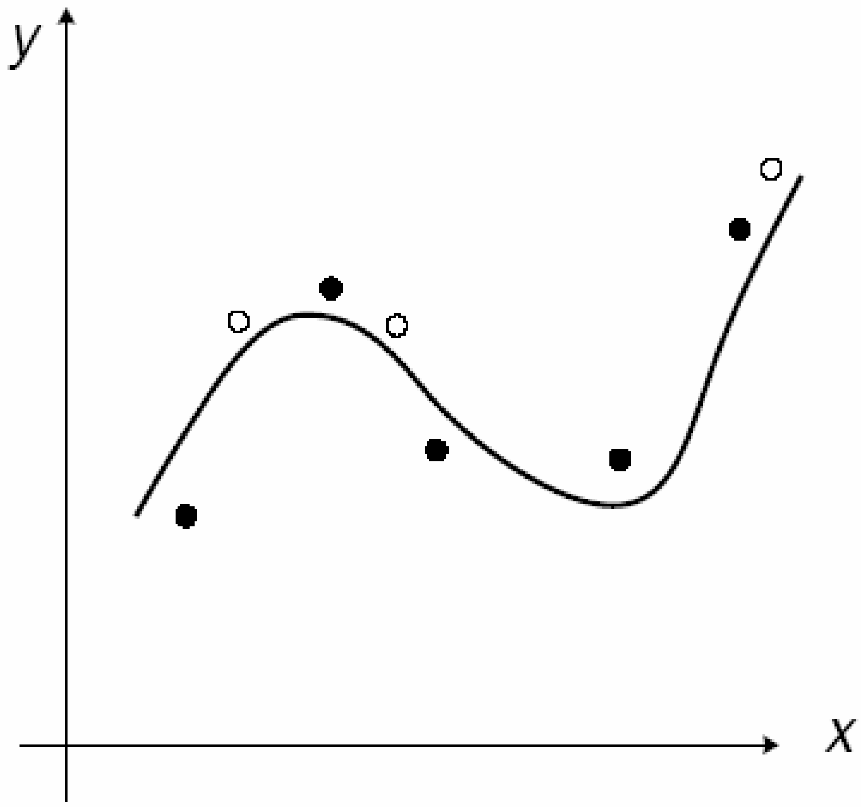
diagram sparse training - two classes



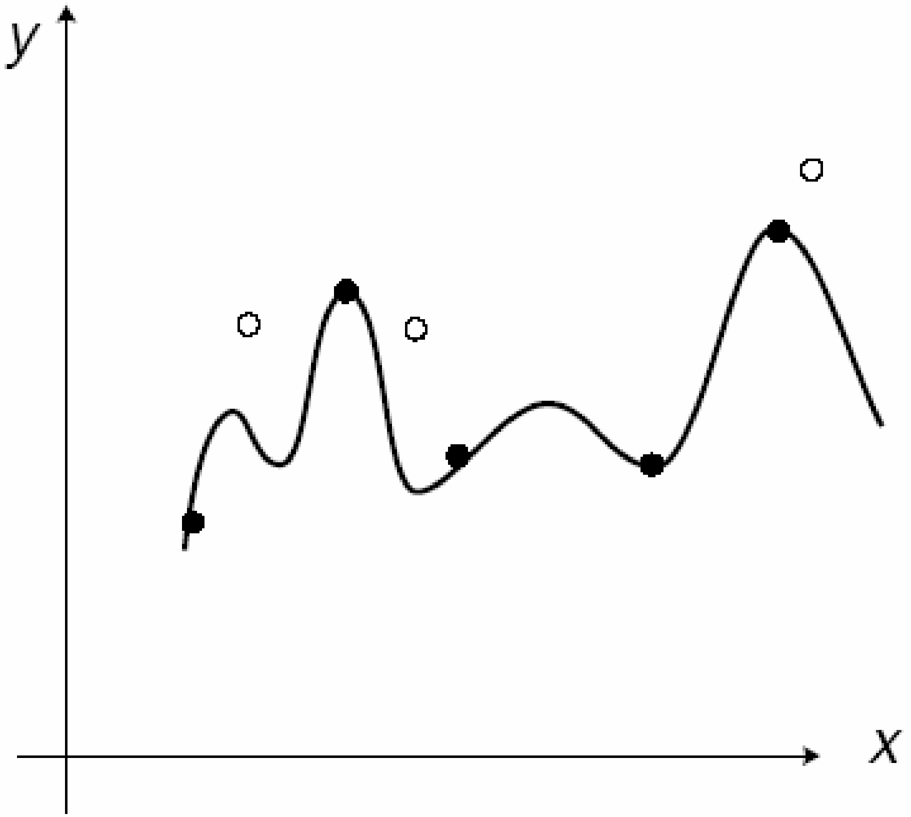
fitting two planes to A and B

Some of the test data are now misclassified. The problem is that the network, with two hidden units, now has too much freedom and has fitted a decision surface to the training data which follows its intricacies in pattern space without extracting the underlying trends.

Overfitting

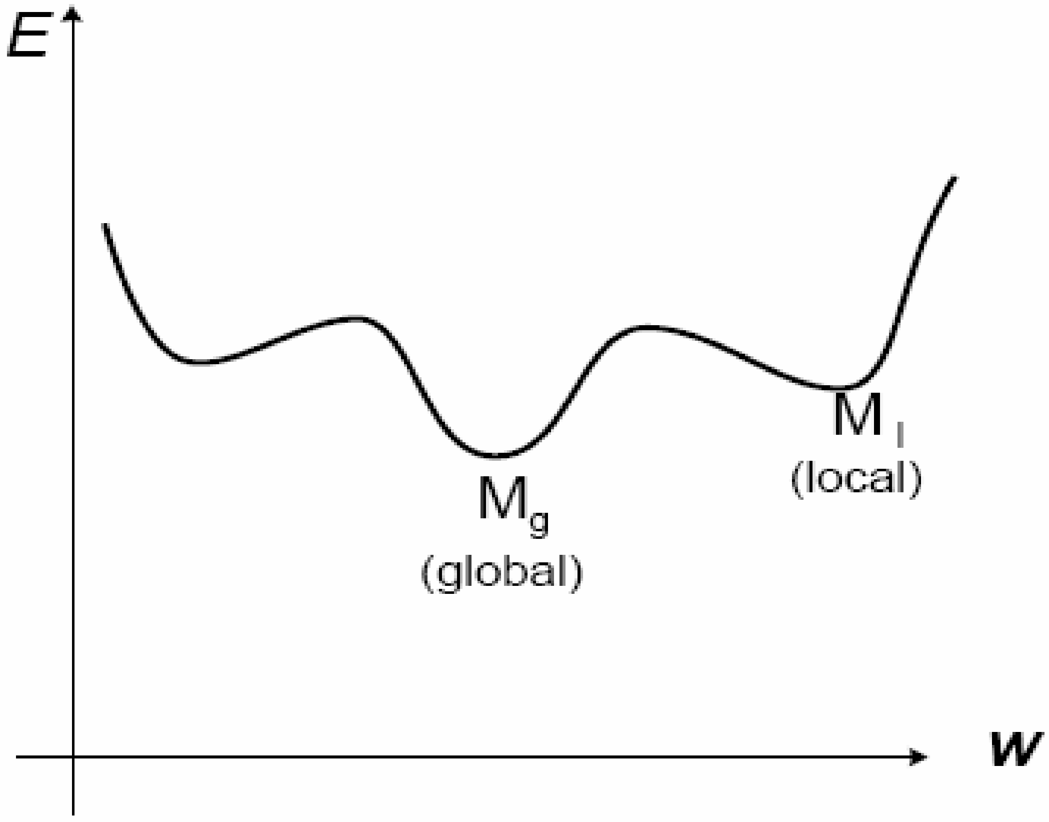


y against x for a binary classifier



overfitting in x-y space

Local minima



local minimum

Two tasks solved by MLP

- Classification (recognition)
  - Usually binary outputs
- Regression (approximation)
  - Analog outputs

Theorem of Kolmogorov

- “Any continuous function from input to output can be implemented in a three-layer net, given sufficient number of hidden units *nH,* proper nonlinearities, and weights.”

Advantages and disadvantages of MLP with back propagation

- Advantages:
  - Guarantee of possibility of solving of tasks
- Disadvantages:
  - Low speed of learning
  - Possibility of overfitting
  - Impossible to relearning
  - It is needed to select of structure for solving of concrete task (usually it is problem)

Increase of speed of learning

- Preprocessing of features before getting to inputs of percepton
- Dynamical step of learning (in begin one is large, than one is decreasing)
- Using of second derivative in formulas for modification of weights
- Using hardware implementation

Fight against of overfitting

- Don’t select too small error for learning or too large number of iteration

Choice of structure

- Using of constructive learning algorithms
  - Deleting of nodes (neurons) and links corresponding to one (prunning networks)
  - Appending new neurons if it is needed (growth networks)
- Using of genetic algorithms for selection of suboptimal structure (already hybrid system)