Neural networks: Multilayer perceptron

Prof. Sven Lončarić

Faculty of Electrical Engineering and Computing

sven.loncaric@fer.hr

http://www.fer.hr/ipg

Overview of topics

- Introduction
- Multilayer perceptron
- Error backpropagation learning
- Discussion

Introduction

- In this topic we present multilayer feed-forward network, also known as multilayer perceptron (MLP)
- MLP has one input layer, one output layer, and one or more hidden layers
- MLPs are used to solve a wide spectrum of problems
- MLPs usually use supervised learning based on the error backpropagation algorithm

Introduction

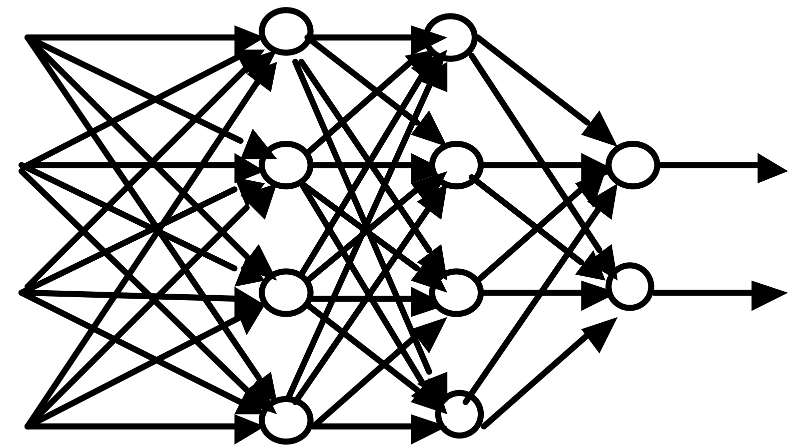
- Multilayer perceptron has three main properties:
  1. Neuron model that has nonlinear activation function, which is smooth (as opposed to Rosenblatt perceptron)
  2. Sigmoid activation function is often used:

$$y_i = \frac{1}{1+\exp(-v_i)}$$

  3. MLP is well connected (there is a large number of synapses)

Multilayer perceptron

- MLP in figure has two hidden layers
- Each layer can have different number of neurons
- Outputs of neurons in one layer are inputs to the next layer
- There are no feedbacks



Introduction

- Problems/disadvantages of MLPs:
  - Multiple nonlinearities and connections make analysis difficult
  - Learning process is difficult due to large number of neurons and due to a need for network to determine what the hidden neurons should learn (for output neurons this is known)

BP learning algorithm

- Error back-propagation (BP)
- Error signal at output of neuron *j* in step *n* is equal to difference between desired and obtained output:

  $$e_j(n) = d_j(n) - y_j(n)$$

  where *j* is an output neuron
- Mean square error in step *n,* in output layer is:

  $$E(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n)$$

  where C is a set of output neurons

BP learning algorithm

- Mean error E_{av} for all training examples is defined as:

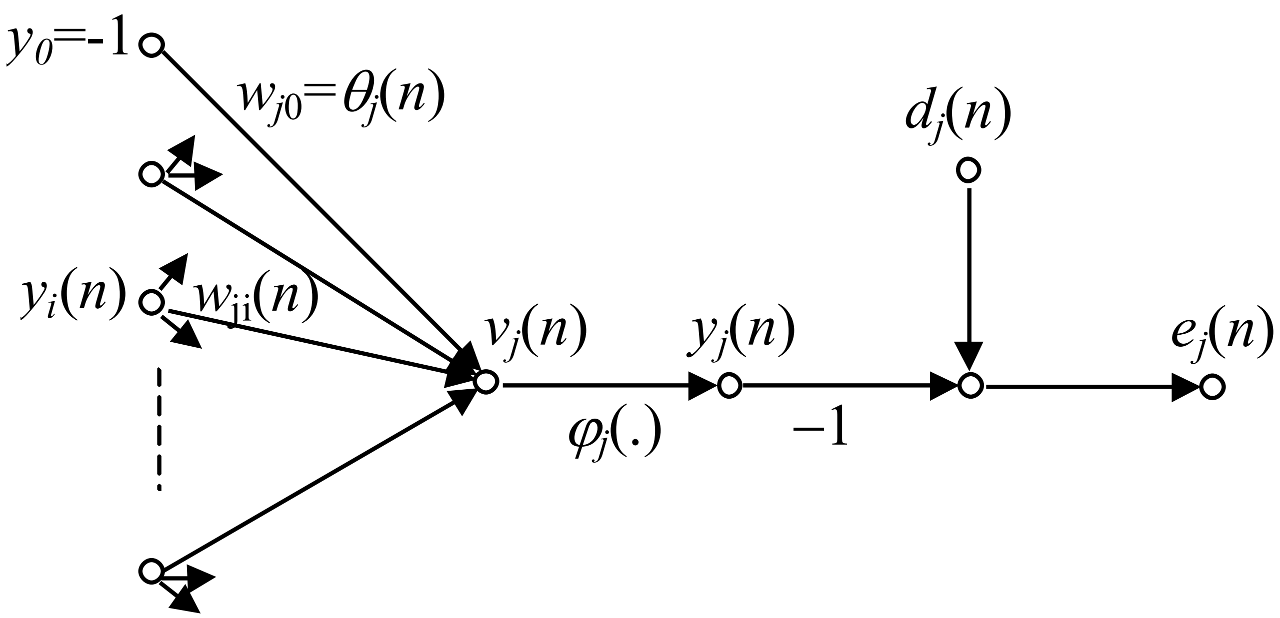
$$E_{av} = \frac{1}{N}\sum_{n=1}^{N} E(n)$$

where N is the number of examples in training set

- E_{av} represents a measure of learning quality
- The goal of learning is to determine a set of weights, which minimizes the mean error E_{av}
- This goal is achieved using an algorithm similar to LMS learning

BP learning algorithm

- Let us assume that we have the neuron shown in figure:



BP learning algorithm

- Activation of neuron *j* is equal to:

$$v_j(n) = \sum_{i=0}^{p} w_{ji}(n) y_i(n)$$

  where *p* is the number of inputs into neuron *j*

- Output of neuron *j* is equal to:

$$y_j(n) = \varphi_j\left(v_j(n)\right)$$

BP learning algorithm

- Error correction is performed similar to LMS algorithm in the direction of negative gradient (steepest descent minimization method)
- The gradient is calculated as follows:

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

$$\frac{\partial E(n)}{\partial e_j(n)} = e_j(n) \qquad \frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j^{'}(v_j(n))$$

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1 \qquad \frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n)$$

BP learning algorithm

- Based on the previous expressions we get:

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = -e_j(n)\varphi_j^{'}(v_j(n))y_i(n)$$

- Error correction is defined using the delta rule (steepest descent method):

$$\Delta w_{ji}(n) = -\eta \frac{\partial E(n)}{\partial w_{ji}(n)}$$

BP learning algorithm

- Therefore, the expression can be written as:

$$\Delta w_{ji}(n) = \eta e_j(n)\varphi_j'(v_j(n))y_i(n) = \eta\delta_j(n)y_i(n)$$

  where $\delta_j(n)$ is so-called local gradient:

$$\delta_j(n) = e_j(n)\varphi_j'(v_j(n))$$

- We see that error correction depends on error $e_j(n)$ at the output of neuron j
- To calculate the error we consider two cases:
  - Neuron j is an output neuron
  - Neuron j is a hidden neuron

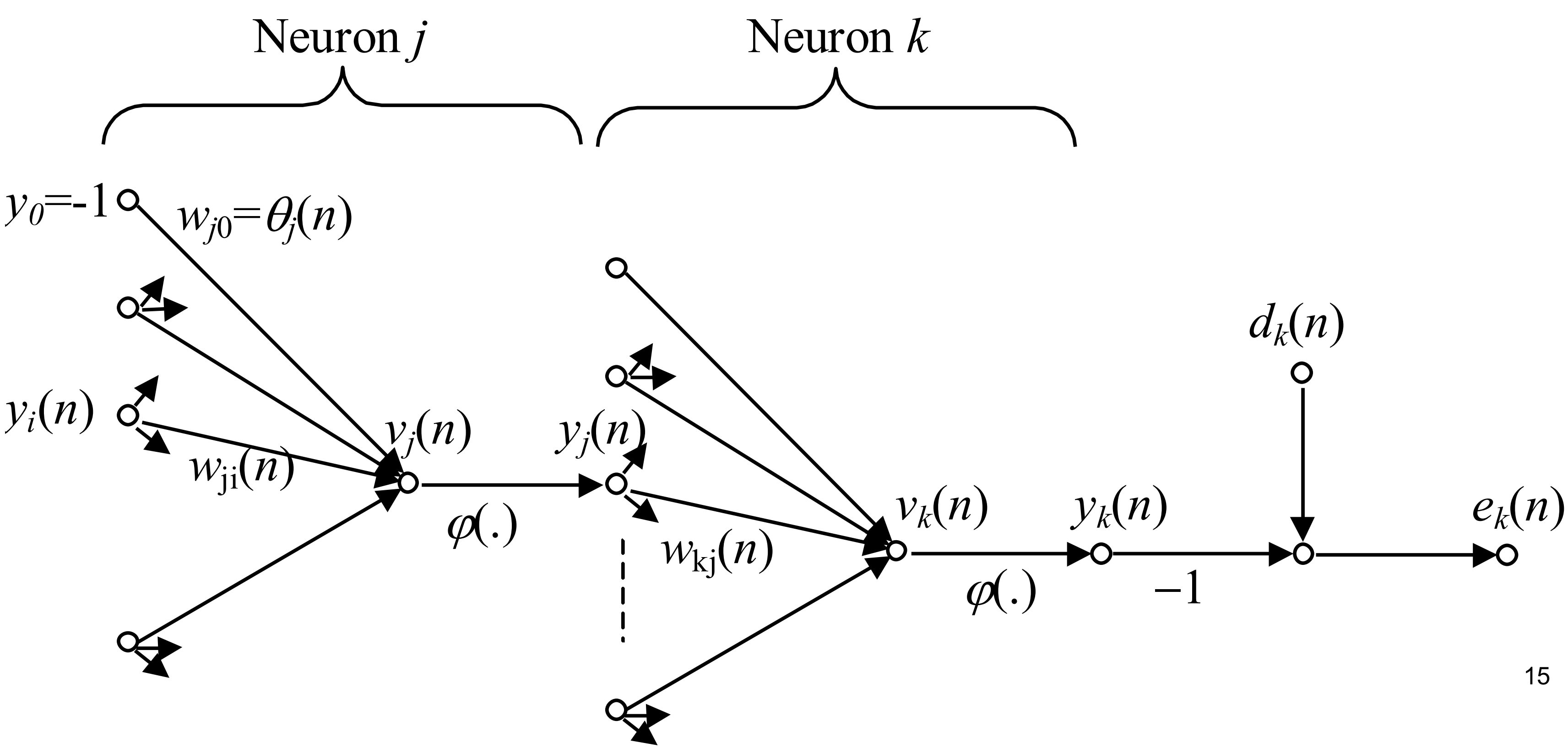
Neuron *j* is output neuron

- If neuron *j* is in output layer then error $e_j(n)$ is easily calculated because we know desired output $d_j(n)$
- Local gradient $\delta_j(n)$ is calculated using the previously derived expression:

$$\delta_j(n) = e_j(n)\varphi_j'(v_j(n))$$

Neuron *j* is hidden neuron

- Let us assume that neuron *j* is in hidden layer connected to output layer (see figure):



Neuron *j* is hidden neuron

- If neuron *j* is in hidden layer then desired output of this neuron is unknown
- In this case, the error of a hidden neuron can be estimated based on errors of neurons in the next layer to which the hidden neuron is connected
- In that case we can write:

$$\delta_j(n) = -\frac{\partial E(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial E(n)}{\partial y_j(n)} \varphi_j^{'}(v_j(n))$$

- Partial derivative of error with respect to input can be calculated as follows

Neuron *j* is hidden neuron

- Let us start from the expression:

$$E(n) = \frac{1}{2} \sum_{k \in C} e_k^2(n)$$

- The partial derivative is then equal to:

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)}$$

- Two derivatives on the right side can be calculated as follows

Neuron *j* is hidden neuron

- Since:

$$e_k(n) = d_k(n) - y_k(n) = d_k(n) - \varphi_k(v_k(n))$$

then:

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi_k{}'(v_k(n))$$

Neuron *j* is hidden neuron

- Activity of neuron *k* is equal to:

$$v_k(n) = \sum_{j=0}^{q} w_{kj}(n) y_j(n)$$

  where *q* is the total number of inputs into neuron *k*

- Then it holds that:

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n)$$

Neuron *j* is hidden neuron

- Based on the derived expressions we see that:

$$\frac{\partial E(n)}{\partial y_j(n)} = -\sum_k e_k(n)\varphi_k'(v_k(n))w_{kj}(n) = -\sum_k \delta_k(n)w_{kj}(n)$$

- Finally we can write that for hidden neuron *j* the local gradient is equal to:

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n)w_{kj}(n)$$

and using the local gradient error correction can be calculated for hidden neuron *j*

Summary of BP algorithm

- Correction $\Delta w_{ji}(n)$ is calculated using delta rule:

(correction) = (parametar) × (local gradient of neuron j) × (i-th input into neuron j)

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)$$

- If neuron j is output neuron then:

$$\delta_j(n) = \varphi_j'(v_j(n)) e_j(n)$$

- If neuron j is hidden neuron then:

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)$$

Two passes in computation

- There are two passes in practical implementation of BP algorithm:

    1. forward pass
    2. backward pass

Forward pass

- In forward pass weights are not changed and neuron outputs are calculated starting from the input towards the output layer
- Signals are calculated using expressions for activity and output of each neuron:

$$v_j(n) = \sum_{i=0}^{p} w_{ji}(n) y_i(n)$$

$$y_j(n) = \varphi_j(v_j(n))$$

Backward pass

- Backward pass starts computation from the output layer
- Error signals are propagated from the output layer towards the input layer, layer-by-layer, recursively computing the local gradient δ for each neuron
- Based on delta rule, for each neuron, weight correction is calculated as shown in BP algorithm summary

Sigmoid activation function

- Most frequently used activation function is:

$$y_j(n) = \varphi_j\big(v_j(n)\big) = \frac{1}{1 + \exp(-v_j(n))}$$

- Derivation of this function appears in expressions for local gradient:

$$\varphi_j'\big(v_j(n)\big) = \frac{\exp(-v_j(n))}{\left[1 + \exp(-v_j(n))\right]^2} = y_j(n)\left[1 - y_j(n)\right]$$

Sigmoid activation function

- For neuron *j* in output layer $y_j(n) = o_j(n)$
- So for output neuron *j* we can write:

$$\begin{aligned}\delta_j(n) &= e_j(n)\varphi_j^{'}(v_j(n)) \\ &= \left[d_j(n) - o_j(n)\right]o_j(n)\left[1 - o_j(n)\right]\end{aligned}$$

- For hidden neuron we can write:

$$\begin{aligned}\delta_j(n) &= \varphi_j{}'(v_j(n))\sum_k \delta_k(n)w_{kj}(n) \\ &= y_j(n)\left[1 - y_j(n)\right]\sum_k \delta_k(n)w_{kj}(n)\end{aligned}$$

Sigmoid activation function

- From expression for derivative of sigmoid function we can see that derivative is largest for $y_j(n) = 0.5$
- For output values close to 0 or 1 derivative is very small
- This means that BP algorithm has largest corrections for neurons having medium output values (around 0.5)
- This contributes to stability of BP learning algorithm

Learning with momentum term

- Error correction is proportional to parameter η that determines learning rate
- Small learning rate η results in slow learning
- Large parameter η may lead to instability (oscillations)
- To incrase learning rate but avoid danger of instability a momentum term can be added:

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n)$$

- Momentum term stabilizes the weight update value (it acts as a low-pass filter)

Modes of training

- In BP algorithm, learning results from many presentations of a set of training examples
- One complete presentation of the entire training set during the learning process is called an epoch
- Learning process is repeated on epoch-by-epoch basis until the weights and bias levels stabilize and the mean squared error over the entire training set converges to some minimum value
- Good practice is to randomize the order of presentation from one epoch to the next
- This provides stochastic search in optimization space

Modes of training

- There are two basic ways of training:
- In sequential mode (presented so far) error correction is performed after each presented training example:
  - advantage: simple implementation
- In batch mode, weight updating is performed after presentation of all training samples that constitute an epoch, error is calculated for all presented pairs, and weight update is performed once for all samples
  - advantage: better estimate of gradient vector
  - disadvantage: more memory required

Stopping criteria

- In general, BP algorithm cannot be shown to converge and there are no well-defined stopping criteria
- Some stopping criteria are:

1. Euclidean norm of the gradient is smaller than a predefined threshold
2. Absolute rate of change of E_{av} between two epochs is sufficiently small
3. Network has good generalization property

Network initialization

- Question: What is a good choice of initial values of weights and thresholds in the network
- Initial values are usually selected randomly from uniform distribution in some interval of values
- Wrong choice can lead to premature saturation (error E_{av} does not reduce through a number of iterations but then starts reducing)

Premature saturation problem

- If for some input pattern neuron activity is very large (positive or negative) then the output of this neuron will be +1 or -1 and derivative in that point of sigmoid curve will be very small
- In that case the neuron is in saturation
- If desired value is -1, and obtained value is +1, or vice-versa, then the neuron is incorrectly saturated
- When this happens it takes many iterations to correct output due to small derivative and hence small weight correction

Premature saturation problem

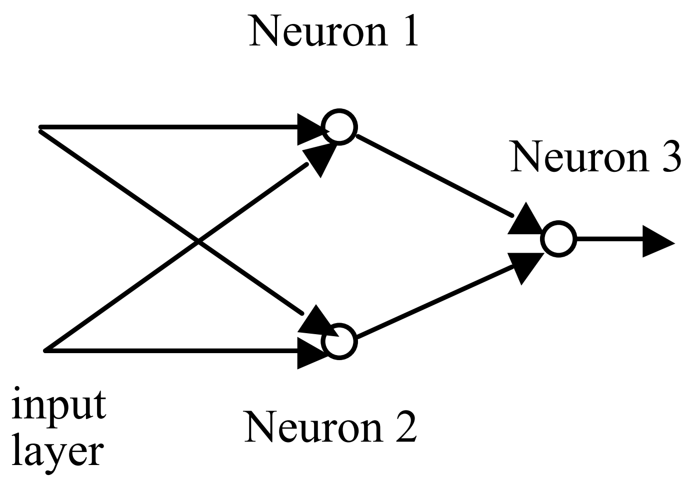
- In the beginning of learning we have saturated and non-saturated neurons in the network
- Weights of non-saturated neurons change faster because derivatives are larger and we see quick decrease in mean error E_{av}
- If at this time saturated neurons still remain saturated there will be premature saturation of the error E_{av}
- Preamature saturation of error E_{av} is when error stops decreasing through iterations as if the network completed learning, while this is not the case

Premature saturation problem

- Approaches to decrease likelihood of premature saturation are:
  1. Choose initial weight values and thresholds as uniformly distributed values in a narrow interval
  2. Probability of incorrect saturation is smaller with smaller number of hidden neurons – use a minimal necessary number of hidden neurons
  3. Probability of premature saturation is smaller when neurons work in linear part of their characteristic

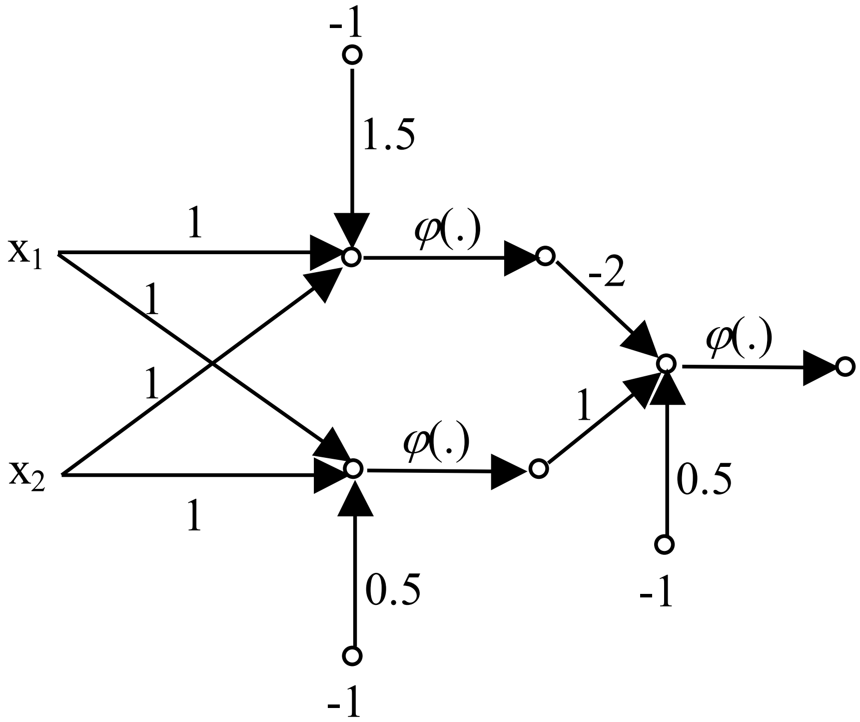
An example: XOR problem

- 0 XOR 0 = 0 0 XOR 1 = 1
- 1 XOR 1 = 0 1 XOR 0 = 1
- XOR can be solved with two hidden neurons and one output neuron:



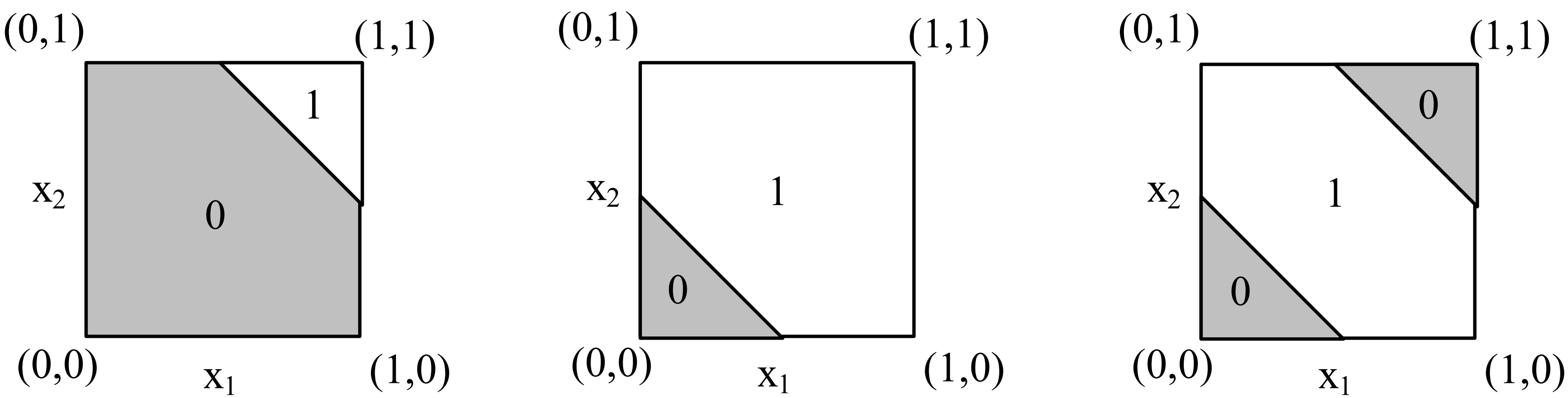
An example: XOR problem

- Flow diagram for realization of XOR function



An example: XOR problem

- Figure represents classification regions of three neurons



Neuron 1 | Neuron 2 | Neuron 3

$w_{11} = w_{12} = 1$ | $w_{21} = w_{22} = 1$ | $w_{31} = -2 \quad w_{32} = 1$

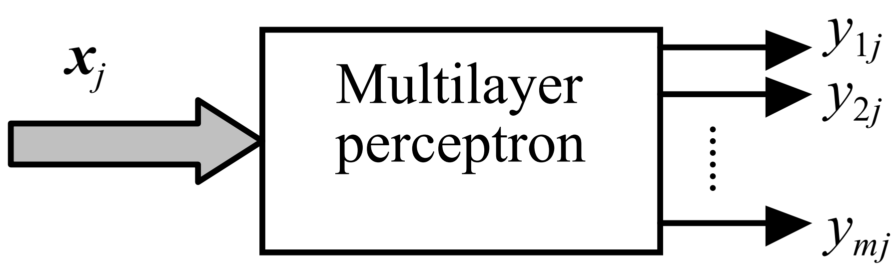
$\Theta_1 = 1.5$ | $\Theta_2 = 0.5$ | $\Theta_2 = 0.5$

Pattern recognition applications

- Multilayer perceptron is often used in pattern recognition
- In PR the problem is to classify unknown pattern into one of *N* classes
- Unknown pattern is a vector describing the object or phenomenon that needs to be classified (e.g. object shape, spoken word)

Pattern recognition problem

- Let us assume that we need to classify vector $\boldsymbol{x}_j$ into one of m classes C_k , k = 1, ..., m
- This problem can be solved using MLP with m outputs



- MLP must be trained so that for vector $\boldsymbol{x}_j$ belonging to class C_k k-th output has value 1 and the others 0

Pattern recognition problem

- After learning process is completed we bring vector **x**, which is not element of the training set, to the input
- MLP for input vector **x** at the output gives vector **y** = $[y_1 \ldots y_m]^T$ which is used to classify vector **x**
- Decision rule: Classify vector **x** into class C_k if:

$$y_k > y_i \quad \forall i \neq k$$

MLP parameter selection

- When designing MLP it is neccessary to select: number of hidden neurons, learning rate, and moment term constant (if moment term is used for weight update)
- Optimal parameters are usually determined experimentally
- A criterion for selection of parameters is recognition accuracy

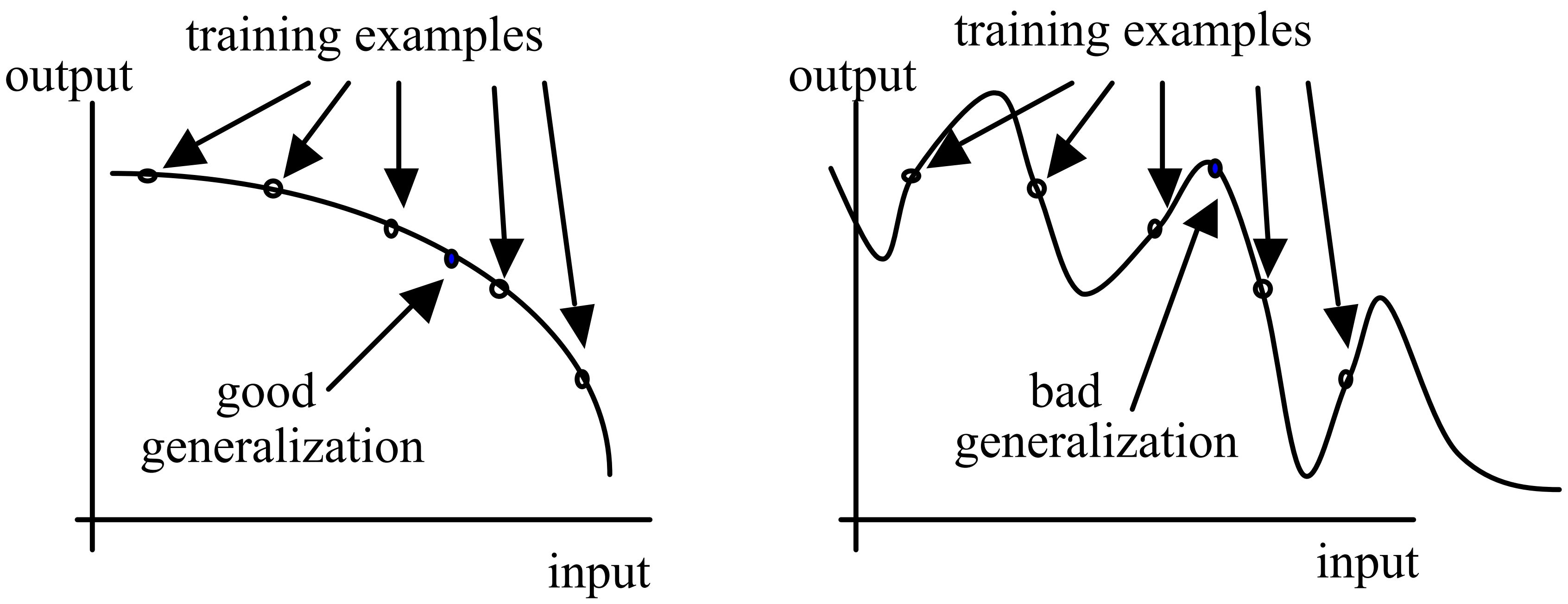
Generalization

- For BP learning we use a training set to train the network
- We use a maximum number of training pairs that we have available
- When learning process is completed we hope that the network will also work “correctly” for vectors that are not contained in the training set
- Such property is called generalization property

Generalization

- MLP can be viewed as a non-linear input-output mapping
- Learning process can be viewed as a process of approximating desired input-output mapping
- In that case generalization property can be viewed as ability for good non-linear interpolation

Generalization



Generalization

- Good generalization is interpolation of available dana (knowledge) with the simplest curve
- The simplest curve is the smoothest one – i.e. the one without large variations
- Bad generalization can be due to too large training set (overfitting)
- The number of training examples is too large when there are more examples than the degrees of freedom of the process that generates examples
- In that case it is necessary to have a larger network complexity

Cross-validation

- Available dana set is divided into learning set and validation set
- The learning set is further divided into set for model estimation (training set) and into model evaluation set
- Validation set is usually 10-20 % of the learning set

Cross-validation

- The goal of this procedure is that validation is conducted on data that is different from the data in the training set
- After the best model (network) is selected, all training data is used for learning
- When learning is complete, network is tested using validation set

Properties of BP algorithm

- BP algorithm is the most popular learning algorithm for supervised learning for MLPs
- BP algorithm is a gradient method that has two main properties:

  1. It is simple and has local support

  2. Performs stochastic gradient descent in the weight space

Applications

- MLP is used in many pattern recognition applications including, but not limited to:
  - Speech recognition
  - Optical character recognition
  - System identification
  - Control systems
  - Autonomous driving
  - Signal analysis
  - Medical diagnostics