

*Modelleerimine ja Juhtimine
Tehisnärvivõrgudega
Identification and Control with
artificial neural networks*

*Eduard Petlenkov,
Automaatikainstituut*

(2nd part)



Supervised learning

Supervised learning is a training algorithm based on known input-output data.

$$|Y_p - NN(X)| = |Y_p - Y| \rightarrow 0$$

X is an input vector;

Y_p is a vector of reference outputs corresponding to input vector X

Y is a vector of NN's outputs corresponding to input vector X

NN is a mathematical function of the network

$(Y=NN(X))$



Õpetamine (2)

Network training procedure consists of the following 3 steps:

1. Computation of NN's outputs corresponding to the current NN's parameters;
2. Computation of NN's error;
3. Updating NN's parameters by using selected training algorithm in order to minimize the error.



Gradient descent error backpropagation (1)

Error function:

$$J(W, \Theta) = \sum_k (Y_p - Y_p^d)^2$$

Y_p - NN's output

$$Y_p = NN(X_p, W, \Theta)$$

X_p - inputs used for training;

NN - NN's function;

Y_p^d - etalons (references) for outputs=desired outputs.

function to be minimized:

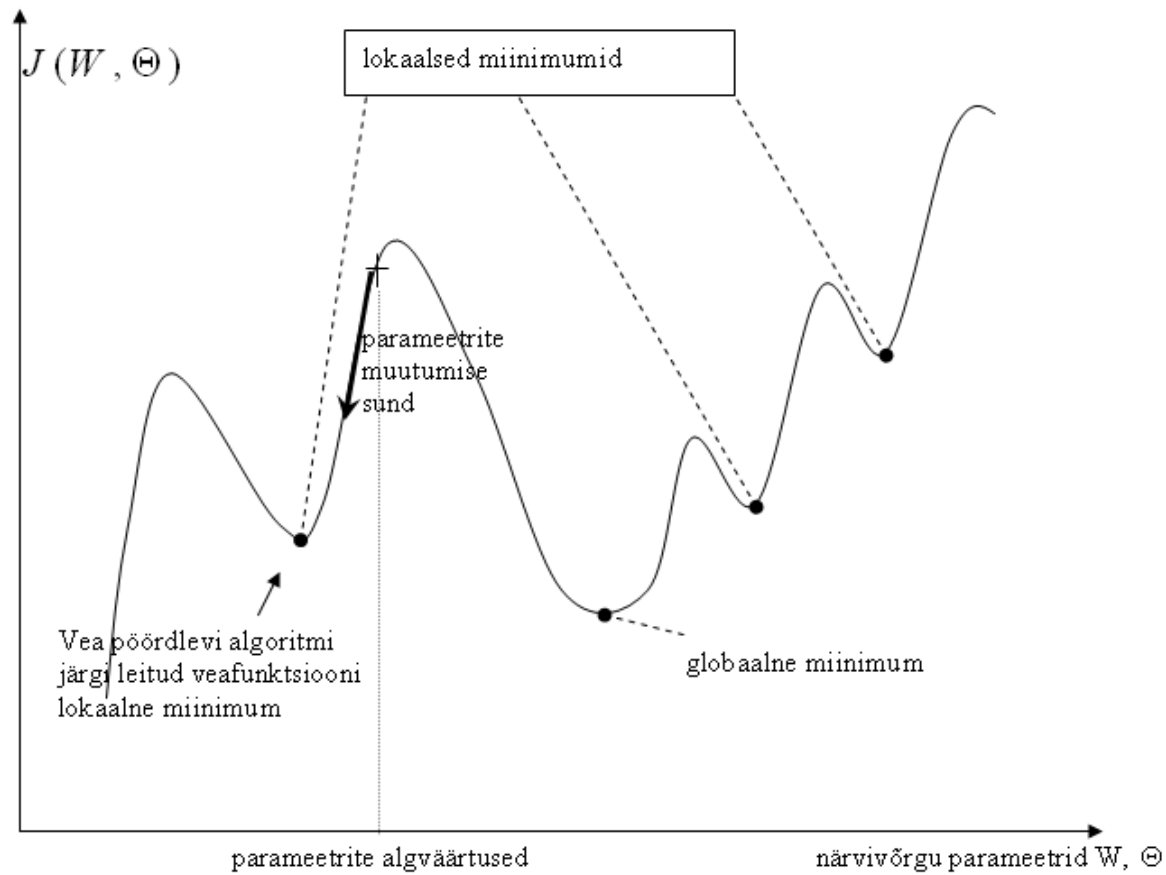
$$\min J(W, \Theta)$$

Gradient descent error backpropagation (2)

$$F(x_1, \dots, x_n) \quad \Rightarrow \quad \begin{aligned} \nabla F(x_1, \dots, x_n) &= \begin{pmatrix} \frac{\partial F}{\partial x_1} & \dots & \frac{\partial F}{\partial x_2} \end{pmatrix} \\ -\nabla F(x_1, \dots, x_n) &= \begin{pmatrix} -\frac{\partial F}{\partial x_1} & \dots & -\frac{\partial F}{\partial x_2} \end{pmatrix} \end{aligned}$$

$$J(W, \Theta) \quad \Rightarrow \quad -\nabla J(W, \Theta) = \begin{pmatrix} -\frac{\partial J}{\partial W} & -\frac{\partial J}{\partial \Theta} \end{pmatrix}$$

Global minimum problem



Training of a double-layer perceptron using Gradient descent error backpropagation (1)

$$yh_j(t) = F_1\left(\sum_i Wh_{ji}(t)x_i(t) + Bh_j(t)\right) \quad y_k(t) = F_2\left(\sum_i Wo_{kj}(t)yh_i(t) + Bo_k(t)\right)$$

i - number of input;

j - number of neuron in the hidden layer;

k - number of output neuron;

$x_i(t)$ NN's input at time instance t ;

$Wh_{ji}(t)$ - Values of the hidden layer weighting coefficients at time instance t ;

$Bh_j(t)$ - Values of the hidden layer biases at time instance t ;

F_1 - Activation function of the hidden layer neurons

$yh_j(t)$ - Outputs of the hidden layer neurons at time instance t ;

$Wo_{kj}(t)$ - Values of the output layer weighting coefficients at time instance t ;

$Bo_k(t)$ - Values of the output layer biases at time instance t ;

F_2 - Activation function of the output layer neurons;

$y_k(t)$ - Outputs of the network at time instance t ;

Training of a double-layer perceptron using Gradient descent error backpropagation (2)

error: $e(t) = y(t) - y_{etalon}(t)$

gradients: $\frac{\partial J}{\partial W_{ij}} = \delta_j(t)x_i(t)$

$$\frac{\partial J}{\partial W_{kj}} = \delta_k(t)yh_j(t)$$

$\delta_j(t), \delta_k(t)$ Signals distributing information about error from output layer to previous layers (that is why the method is called error backpropagation):

$$\delta_k(t) = (y_k(t) - y_k^d(t)) \cdot F_2'$$

$$\delta_j(t) = F_1' \cdot \sum_k \delta_k(t) \cdot W_{okj}(t)$$

F_1', F_2' Are derivatives of the corresponding layers activation functions.

Training of a double-layer perceptron using Gradient descent error
backpropagation (3)

Updated weighting coefficients:

$$W_{o_{kj}}(t+1) = W_{o_{kj}}(t) - \eta \cdot \frac{\partial J}{\partial W_{o_{kj}}} = W_{o_{kj}}(t) - \eta \delta_k(t) y h_j(t)$$

$$W_{h_{ji}}(t+1) = W_{h_{ji}}(t) - \eta \cdot \frac{\partial J}{\partial W_{h_{ji}}} = W_{h_{ji}}(t) - \eta \delta_j(t) x_i(t)$$

η learning rate

$$f(x) = \frac{1}{1 + e^{-x}} \Rightarrow f'(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = \frac{1}{1 + e^{-x}} - \frac{1}{(1 + e^{-x})^2} = f(x) - f^2(x)$$

Gradient Descent (GD) method vs Levenberg-Marquardt (LM) method

$$Z_n = \{[u(t), y(t)], t = 1, \dots, N\}$$

$$J(\theta, Z_N) = \frac{1}{2N} \sum_{t=1}^N [y(t) - \hat{y}(t)]^2 = \frac{1}{2N} \sum_{t=1}^N e(t, \theta)^2$$

$$G(\theta) = \frac{\partial J(\theta, Z_N)}{\partial \theta} = \frac{1}{N} \sum_{t=1}^N \frac{\partial e(t, \theta)}{\partial \theta} e(t, \theta) \quad \text{- Gradient}$$

$$H(\theta) = \frac{\partial^2 J(\theta, Z_N)}{\partial \theta \partial \theta^T} = \underbrace{\frac{1}{N} \sum_{t=1}^N \frac{\partial e(t, \theta)}{\partial \theta} \left[\frac{\partial e(t, \theta)}{\partial \theta} \right]^T}_{R(\theta)} - \frac{1}{N} \sum_{t=1}^N \frac{\partial^2 e(t, \theta)}{\partial \theta \partial \theta^T} e(t, \theta) \quad \text{- Hessian}$$

<p>GD: $\theta(k+1) = \theta(k) - \lambda G(\theta)$</p>	<p>LM: $\theta(k+1) = \theta(k) + \Delta\theta(k)$ $[R(\theta(k)) + \lambda I] \Delta\theta(k) = -G(\theta(k)) \Rightarrow \Delta\theta(k)$</p>
---	--

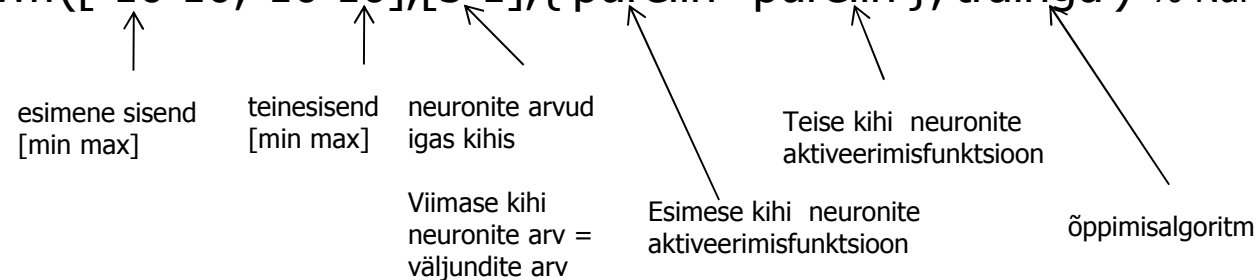
Näide MATLABis (1)

```
P1=(rand(1,100)-0.5)*20 % Juhuslikud sisendid. 100 väärtust vahemikus [-10 +10]
P2=(rand(1,100)-0.5)*20
```

```
P=[P1;P2] % Närvivõrgu sisendite moodustamine
```

```
T=0.3*P1+0.9*P2 % Vastavate väljundite arvutamine
```

```
net=newff([-10 10;-10 10],[5 1],{'purelin' 'purelin'},'traingd') % Närvivõrgu genereerimine
```



```
net.trainParam.show=1;
net.trainParam.epochs=100;
```

Treenimise parameetrid



Näide MATLABis (2)

```
net=train(net,P,T) % NÄrvivõrgu treenimine
```

```
out=sim(net,[3 -2]) % NÄrvivõrgu simuleerimine
```

↑
Teise sisendi
väärtus

↑
Esimese sisendi
väärtus

```
W1=net.IW{1,1} % esimese kihi kaalukoefitsientide maatriks
```

```
W2=net.LW{2,1} % teise kihi kaalukoefitsientide maatriks
```

```
B1=net.b{1} % esimese kihi neuronite nihete vektor
```

```
B2=net.b{2} % teise kihi neuronite nihete vektor
```