Stanisław Jankowski

Institute of Electronic Systems
Warsaw University of Technology

Neural networks as statistical learning systems

Tutorial for EEVAL students

Warsaw 2015

INTRODUCTION

Artificial neural networks

Biological motivation

- Exploration of new ideas and tools of mathematics and physics, informatics for modeling of the brain functions.

Technical motivation

- Invention of new tools (hardware and software) able to solve complex problems.
- Application of modern statistical methods for solving hard problems.

Properties of the brain

1) Robustness to damage;
2) Flexibility – ability to learn;
3) Ability to process the fuzzy, disturb with noise and inconsistent information;
4) Massively parallel architecture;

Nervous cell scheme

Nervous cell has many inputs (organized into dendrite) and one output - an axon. The cells communicate by synapses. The cell in an excitatory state generates a series of pulses called spikes.

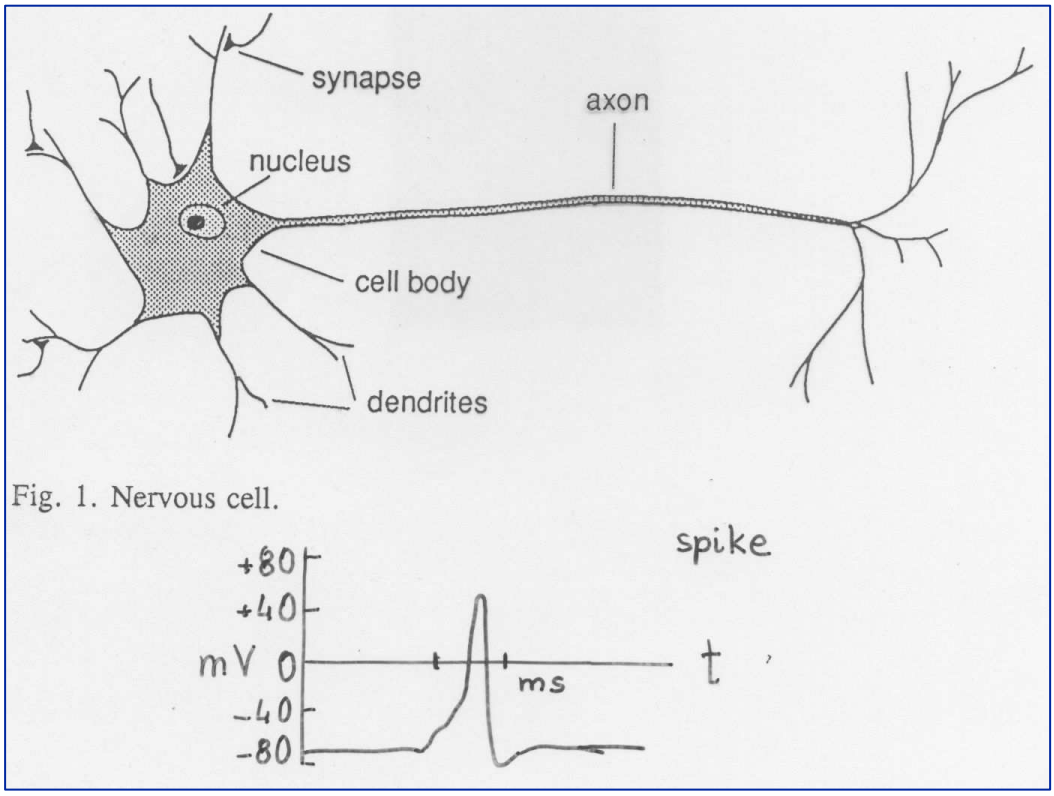


Fig. 1. Nervous cell.

Nervous cell properties

1) neural cell has many inputs and only one output,
2) unilateral information flow,
3) pulse cell activity,
4) the cell performs spatial, temporal and subthreshold summation of input signals;
5) synapses can be either excitatory or inhibitory;
6) the cell activation function has a threshold that can be adaptively changed;
7) the pulse (spike) generation is characterized by the refraction time that causes the frequency modulation;
8) pulses are generated with a synaptic delay;
9) axon – is a cell output;
10) learning is performed by local updating the weights of excitatory synapses.

HISTORY OF NEURAL NETWORKS

1943 McCulloch and Pitts model of nervous cell
1960 Rosenblatt Perceptron, Widrow-Hoff Adaline
1982 Hopfield associative memory, Kohonen self-organizing map
1985 Multilayer Perceptron
1988 Cellular Neural Network
1990 Support Vector Machine - V. N. Vapnik
2005 Transduction
2006 Graph machines - G. Dreyfus
2007 Deep learning systems,
2008 Swarm optimization
2010 Causality

Neural Networks and statistics

The present research in the area of artificial neural networks is not oriented into modeling of biological nervous system. In fact, the neural networks are data-based blackbox models. The learning (or training) is a method of optimal structure selection and parameters (weights) computation. These models are based on numerical considerations.
The artificial neural networks are able to perform the following tasks:

1) regression (function approximation);
2) classification (pattern recognition)
3) data clustering.

It may be concluded that the neural networks are nonlinear statistical learning systems:

Architecture choice	**Regression form**
Learning set	**Observations**
Network learning/training	**Regression parameters estimation**
Generalization	**Interpolation, extrapolation**

Fundamental property of neural network universal approximation
(Stone-Weierstrass theorem)

For any function $F(x_1, x_2,..., x_n)$ there exists a neural network whose output signal $Y(x_1, x_2,..., x_n)$ is arbitraly close to a given function in a subspace of input signals.

Learning mechanism

- local phenomenon
- based on repetition
- based on examples
- supervised, unsupervised (or semi-supervised)

BASIC QUESTIONS

1. How to represent a given task as function approximation problem?
2. How to design an appropriate network structure?
3. How to calculate the network parameters (weights)?
4. How to estimate the quality of designed neural network?

NETWORK LEARNING

DEFINITION
Algorithm of network weights calculation {w}

COMPONENTS

- learning set (examples)
- error function (goal function)
- error function minimization method (e.g. gradient descent)

GENERALIZATION – A GOAL OF LEARNING

DEFINITION

- Estimation of the model properties on the general population.

COMPONENTS

- test set from the same general population as learning set

Criterion, e.g. mean-square error on the test set

LEARNING METHODS

SUPERVISED
All learning examples are assigned with an expert label – the goal is defined by a supervisor.

UNSUPERVISED
A network is self-organizing in order to find a data representation (e.g. Kohonen self-organizing map)
Application: Analysis and visualization of data clustering

Inteligencja obliczeniowa – computational intelligence

Sztuczne sieci neuronowe to formalne modele statystyczne zbudowane na podstawie danych eksperymentalnych. Nie mają związku ze współczesną wiedzą o funkcjonowaniu układu nerwowego.

A formal neuron

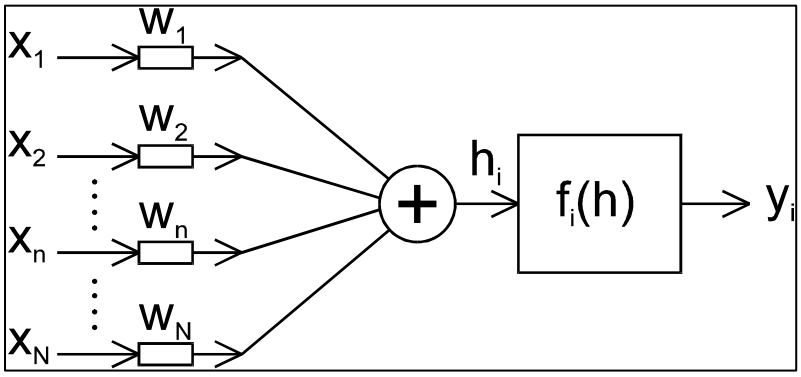


Fig.. Scheme of a formal neuron

x – input vector
w – weight vector (parametr vector)
h – neuron input signal
f(*h*) – neuron activation function
y – neuron output signal

Linear neuron

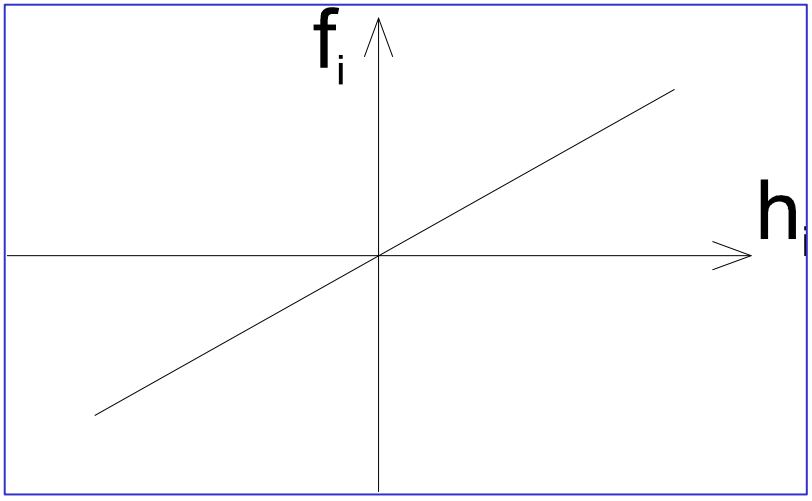


Fig. Linear activation function

Activation function is linear and differentiable

$$y = g(h) = h = \sum_{k=1}^{K} w_k x_k + b$$

We define an error function $E(w)$ – differentiable function of weights w:

$$E(w) = \frac{1}{2}\sum_n (d^n - y^n)^2 = \frac{1}{2}\sum_n (d^n - \sum_k w_k x_k^n)^2$$

d_n – desired value for *n*-th vector of the training set;
y_n – current value of neuron output for *n*-th training vector;
w_k - *k*-th weight.

We apply the gradient method for the minimization of error function *E*(w).
We attempt to find a learning update rule to obtain the weights values that correspond to the minimum of the error function *E*(w) starting from any initial weights w_0.

Learning rule – a method of weights calculation:

$$w_k(t+1) = w_k(t) + \Delta w_k$$

Gradient descent algorithm

$$\Delta w_k = -\eta \frac{\partial E}{\partial w_k} = \eta (y^n - d^n) x_k^n$$

$\eta < 1$ – a learning coefficient
If the weights are updated sequentially for every n-th example then the output error is:

$$\delta^n = y^n - d^n$$

We obtain the delta rule:

$$\Delta w_k = \eta \delta^n x_k^n$$

The other names of this rule:

- Widrow-Hoff rule,
- *Adaline* rule,
- Least-squares rule.

Nonlinear neuron *g(h)*

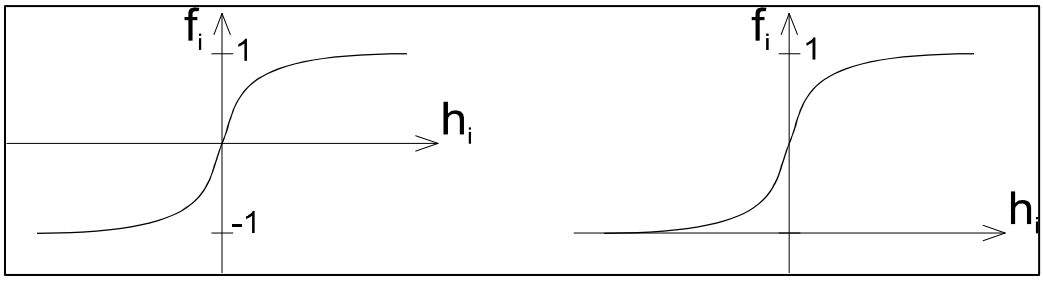


Fig. Sigmoid activation functions: tanh and logsig.

Nonlinear activation functions:

a) hyperbolic tangent (tansig):

$$y(h) = \tanh(h) = \frac{\exp(h) - \exp(-h)}{\exp(h) + \exp(-h)}$$

b) logistic sigmoid function (logsig):

$$y(h) = \frac{1}{1+\exp(-h)}$$

The cost function – mean squares error:

$$E(\mathbf{w},\mathbf{x}) = \frac{1}{2}\sum_n (d^n - y^n)^2 = \frac{1}{2}\sum_n (d^n - g(\sum_k w_k x_k^n))^2$$

Gradient descent algorithm:

$$\frac{\partial E}{\partial w_k} = -\sum_n [d^n - g(h^n)]g'(h^n)x_k^n$$

The error δ^n contains the derivative of the activation function:

$$\delta^n = [d^n - g(h^n)]g'(h^n)$$

Gradient descent update rule:

$$\Delta w_k = \eta \delta^n x_k^n$$

The derivative of sigmoidal function, eg. g(h) = tanh(h) is greater for small argument values, i.e. small input signals and small initial weights.

Derivatives of sigmoidal functions:

$$g(h) = \tanh(\beta h) = \frac{\exp(\beta h) - \exp(-\beta h)}{\exp(\beta h) + \exp(-\beta h)} \quad \text{(tansig)}$$

$$g'(h) = \beta(1 - g^2)$$

$$g(h) = \frac{1}{1+\exp(-\beta h)} \quad \text{(logsig)}$$

$$g'(h) = \beta g(1-g)$$

The drawback of the gradient algorithm: Local minima of $E(\mathbf{x},\mathbf{w})$)!

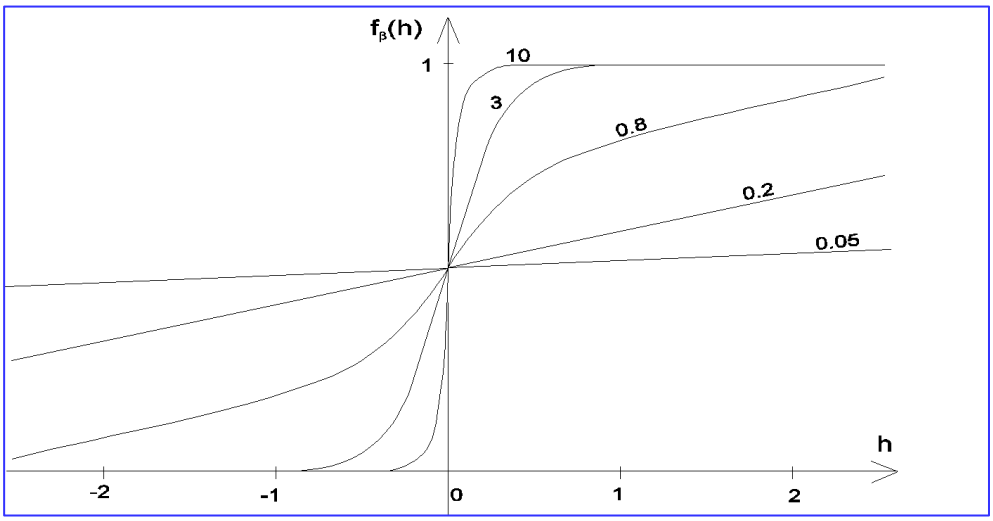


Fig. Hyperbolic tangent properties.

Multilayer perceptron for classification - determination of hypersurfaces separating classes

- Network inputs: descriptors
- Network outputs: output numbers equal to the number of classes; output neurons of sigmoid activation function.
- Learning: for an object belomging to class *Ci*, the desired i-th output value is +1, and the other desired output values are -1.
- The hypersurface separating class *Ci* form the other classes is described by the equation $gi(x) = 0$ where *gi* is the neuron output *i*.

Therefore, the first layer weights are directional coefficients of separating hyperplanes, and the separating hypersurface is their combination.

Two-layer perceptron

w_{ji} - weight of connection between i-th and j-th neuron,
h_j - i-th neuron input signal,
z_j - j-th neuron output signal
(for the output neuron zk = yk),
x_i - network input signal,
d_k - desired k-th output signal,

$n = 1, 2,..., N$ - index of training set examples,
$i = 1, 2,..., I$ - index of network inputs,
$j = 1, 2,..., J$ - index of hidden neurons,
$k = 1, 2,..., K$ - index of output neurons,

g(*h*) - neuron activation function.

Error function: $E^n = E^n(y_1^n, y_2^n,..., y_K^n)$

x – input signal
z – hidden neuron output signal
y – output neuron output signal
h – neuron input

n-th input signal forward propagation:

$$\mathbf{x}^n = [x_i^n], \quad i = 1,..., I \quad n = 1,..., N$$

a) j-th hidden neuron input

$$h_j^n = \sum_{i=1}^{I} w_{ji} x_i^n, \quad j = 1,..., J$$

b) j-th hidden neuron output

$$z_j^n = g(h_j^n) = g(\sum_{i=1}^{I} w_{ji} x_i), \quad j = 1,..., J$$

c) k-th output neuron input

$$h_k^n = \sum_{j=1}^{J} w_{kj} z_j^n = \sum_{j=1}^{J} w_{kj}\, g(\sum_{i=1}^{I} w_{ji} x_i^n), \quad k = 1,..., K$$

d) k-th output neuron output

$$y_k^n = g(h_k^n) = g(\sum_{j=1}^{J} w_{kj} z_j^n) = g(\sum_{j=1}^{J} w_{kj}\; g(\sum_{i=1}^{I} w_{ji} x_i^n)), \quad k = 1,\ldots, K$$

Neural network is a multivariable nonlinear functions of:
- arguments - inputs **x**
- parameters - weights **w**

The output error is the difference between calculated *y* and desired network output signal *d*:

$$\varepsilon_k^n = d_k^n - y_k^n$$

Error function for n-th example E^n:

$$E^n = -\frac{1}{2}\sum_{k=1}^{K}(d_k^n - y_k^n)^2$$

Error function for the training set *E*:

$$E = \sum_{n=1}^{N} E^n = -\frac{1}{2}\sum_{n=1}^{N}\;\sum_{k=1}^{K}(d_k^n - y_k^n)^2$$

$$E = -\frac{1}{2}\sum_{n=1}^{N}\;\sum_{k=1}^{K}[d_k^n - g(\sum_{j=1}^{J} w_{kj}\; g(\sum_{i=1}^{I} w_{ji} x_i^n))]^2$$

*E is a function of weights {**w**} and of training ste examples {**x**n, **d**n}*

Idea of the error back propagation 1985

Minimization of the error function E by the gradient descent method:
the calculation of network weights update is performed by sending the error signal back from the network output to the input.
The derivatives of error function are calculated from the output to the input.

Error back propagation
1) Output (second) layer weight update Δw_{kj}:

$$\Delta w_{kj} = -\eta \frac{\partial E^n}{\partial w_{kj}}$$

Derivative of the error function E with respect to the weight w_{kj}

$$\frac{\partial E^n}{\partial w_{kj}} = \frac{\partial E^n}{\partial y_k^n} \cdot \frac{\partial y_k^n}{\partial h_k} \cdot \frac{\partial h_k}{\partial w_{kj}} =$$

$$= (d_k^n - y_k^n) g'(h_k) \frac{\partial h_k}{\partial w_{kj}} =$$

$$= (d_k^n - y_k^n) g'(h_k) z_j^n =$$

$$= \delta_k^n \cdot z_j^n$$

where:

$$\delta_k^n = (d_k^n - y_k^n) g'(h_k)$$

Output layer weight update:

$$\Delta w_{kj} = -\eta \delta_k^n \cdot z_j^n = -\eta (d_k^n - y_k^n) g'(h_k) z_j^n$$

2) Hidden layer (first layer) weight update Δw_{ji}:

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

$$\frac{\partial E^n}{\partial w_{ji}} = \frac{\partial E^n}{\partial y_k^n} \cdot \frac{\partial y_k^n}{\partial h_k} \cdot \frac{\partial h_k}{\partial w_{kj}} = \sum_{k=1}^{K} (d_k^n - y_k^n) g'(h_k) w_{kj} g'(h_j^n) \frac{\partial h_j^n}{\partial w_{ji}} =$$

$$= \sum_{k=1}^{K} (d_k^n - y_k^n) g'(h_k) w_{kj} g'(h_j^n) x_i^n = g'(h_j^n) [\sum_{k=1}^{K} (d_k^n - y_k^n) g'(h_k) w_{kj}] x_i^n =$$

$$= g'(h_j^n) [\sum_{k=1}^{K} \delta_k^n w_{kj}] x_i^n = \delta_j^n \cdot x_i^n$$

where:

$$\delta_j^n = g'(h_j^n) [\sum_{k=1}^{K} \delta_k^n w_{kj}]$$

Hidden layer weight update:

$$\Delta w_{ji} = -\eta \delta_j^n \cdot x_i^n = -\eta g'(h_j^n) [\sum_{k=1}^{K} \delta_k^n w_{kj}] x_i^n$$

In order to calculate the error function gradient with respect to the first layer weights we sum up the errors of the output neurons connected by the weights w_{kj} to j-th neuron of the hidden layer.

Improvement of the error back propagation algorithm

Goal: damping the error function oscillations that can occur in the search of minimum $E(\mathbf{w})$ by gradient descent method.

- Adding a momentum term:

$$\Delta \mathbf{w}(t) = -\eta \nabla E(\mathbf{w}(t)) + \mu \Delta \mathbf{w}(t-1))$$

μ - momentum term $0 \le \mu \le 1$,

η - learning rate.

Effect:

a) If the slope of the error function $E(\mathbf{w})$ is small then the gradient ∇E is approximately constant and after many iterations we obtain:

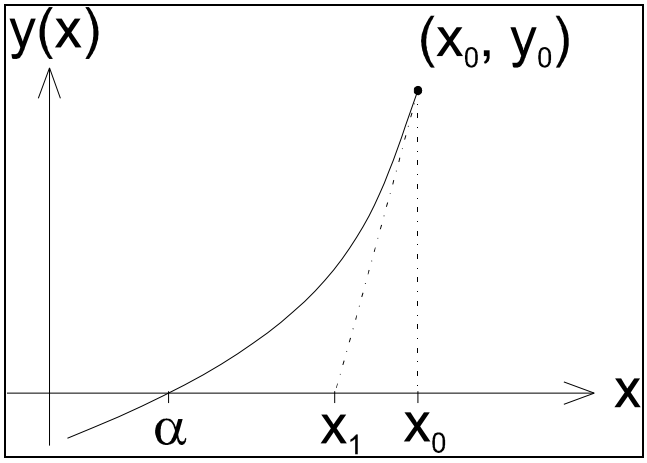
$$\Delta \mathbf{w} = -\eta (1 + \mu + \mu^2 + \cdots) \nabla E = -\left(\frac{\eta}{1-\mu} \right) \nabla E$$

Hence, the effect of the momentum term is equivalent to the increase of the learning rate.

b) If the error function gradient oscillates then the momentum term in subsequent oscillations annihilates and the learning term remains constant.

Function minimization

Newton's method (tangent method):



Goal: Zero (root) of a monovariable function

Equation of the tangent line at point P(x0, y0):

$$y - f(x_0) = f'(x_0)(x - x_0)$$

Point of intersection with *Ox* axis:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)},$$

$$f'(x_0) \neq 0$$

It can be stated that:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \xrightarrow[n\to\infty]{} \alpha$$

The minimum of function: zero (root) of derivative $f'(x^*) = 0$

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

Gradient ∇ - Victor of the first derivatives of multivalued function
Hessian H – second derivatives matrix

$$\frac{\partial^2 E}{\partial w_{ji} \partial w_{lk}}$$

Taylor series expansion of the error function gradient at point $\mathbf{w}_0$::

$$\nabla E(\mathbf{w}) = \nabla E(\mathbf{w}_0) + \mathbf{H}(\mathbf{w} - \mathbf{w}_0) = 0$$

The gradient equals 0 if:

$$\mathbf{w} = \mathbf{w}_0 - \mathbf{H}^{-1}\nabla E(\mathbf{w}_0)$$

This equation is the basis of Newton's method of multivariable function minimization.

Problem: computation of the Hessian inverse **H**-1.

Advantage: calculation of the square function minimium in 1 step.

Metody zmiennej metryki (ang. *quasi-Newton methods*) - aproksymacja odwrotności hesjanu.

The gradient and the Jacobian matrix of a neural network with one output neuron:

The gradient of the mean squares terror function $E(\mathbf{w})$ at $\mathbf{w}=\mathbf{w}_0$ is a q-dimensional vector (number of neural network weights):
j-th component equals:

$$\frac{\partial E}{\partial w_j}|_{\mathbf{w}_0} = -\sum_{n=1}^{N}[d^n - y^n(\mathbf{w}_0)]\frac{\partial y^n}{\partial w_j}|_{\mathbf{w}_0} = -\sum_{n=1}^{N} e^k(\mathbf{w}_0)\frac{\partial y^n}{\partial w_j}|_{\mathbf{w}_0}$$

Hence

$$\frac{\partial E}{\partial \mathbf{w}}|_{\mathbf{w}_0} = \begin{bmatrix} \frac{\partial y^1}{\partial w_1}e^1 + \cdots + \frac{\partial y^N}{\partial w_1}e^N \\ \cdots \\ \frac{\partial y^1}{\partial w_q}e^1 + \cdots + \frac{\partial y^N}{\partial w_q}e^N \end{bmatrix}$$

The Jacobian matrix at point **w** can be expressed by:

$$\mathbf{Z}(\mathbf{w}_0) = \frac{\partial y}{\partial \mathbf{w}^T}|_{\mathbf{w}_0} = -\frac{\partial \mathbf{e}}{\partial \mathbf{w}^T}|_{\mathbf{w}_0} \begin{bmatrix} \frac{\partial y^1}{\partial w_1} \cdots \frac{\partial y^N}{\partial w_1} \\ \cdots \\ \frac{\partial y^1}{\partial w_q} \cdots \frac{\partial y^N}{\partial w_q} \end{bmatrix}_{\mathbf{w}_0}$$

The relation between the gradient and the Jacobian matrix of a neural network is as follows:

$$\nabla E(\mathbf{w}_0) = \frac{\partial E}{\partial \mathbf{w}} = \sum_{n=1}^{N}\frac{\partial E^n}{\partial \mathbf{w}} = \sum_{n=1}^{N} e^n \frac{\partial e^n}{\partial \mathbf{w}} = -\sum_{n=1}^{N} e^n \frac{\partial y^n}{\partial \mathbf{w}} = -\mathbf{Z}^T\mathbf{e}$$

Hessian – approximation by the Jacobian matrix

The error function for one output neuron *y* is:

$$E = \frac{1}{2}\sum_n (y^n - d^n)^2$$

then the Hessian:

$$\frac{\partial^2 E}{\partial w_{ji}\partial w_{lk}} = \sum_n \frac{\partial y^n}{\partial w_{ji}}\frac{\partial y^n}{\partial w_{lk}} + \sum_n (y^n - d^n)\frac{\partial^2 y^n}{\partial w_{ji}\partial w_{lk}}$$

After performing of some iterations of the error back ropagation learning the output signal y^n is close to the desired output d^n. Then the second sum can be neglected.
In case of additive noise the mean value of error (y^n - d^n) tends to zero. Then the Hessian can be approximated as:

$$\frac{\partial^2 E}{\partial w_{ji}\partial w_{lk}} = \sum_n \frac{\partial y^n}{\partial w_{ji}}\frac{\partial y^n}{\partial w_{lk}}$$

The neural network Jacobian matrix $\mathbf{Z}$ of size $N \times q$ containing the partial derivatives of the neural network output y at point $\mathbf{w}_0$ with respect to q weights for N elements of learning set:

The relation between the error function gradient and the Jacobian matrix of neural network:

$$\nabla E(\mathbf{w}) = \mathbf{Z}^T \mathbf{e}$$

The Hessian matrix approximation:

$$\hat{\mathbf{H}} = \mathbf{Z}^T \mathbf{Z}$$

Levenberg-Marquardt - algorithm

If the matrix $\mathbf{Z}^T\mathbf{Z}$ is ill-conditionned or not invertible the following algorithm can be applied:

$$\mathbf{w}(t) = \mathbf{w}(t-1) + (\mathbf{Z}^T\mathbf{Z} + \lambda\mathbf{I})^{-1}\mathbf{Z}^T\mathbf{e}$$

$$e_n = y_n - d_n$$

$\lambda \approx 0.1$.

Example – neural approximator:

Sigmoidal activation function of hidden neurons:

$$g(h) = \tanh(h) = \frac{\exp(h) - \exp(-h)}{\exp(h) + \exp(-h)}$$

Derivative of activation function:

$$g'(h) = 1 - g^2(h)$$

One linear output neuron:

$$y = \sum_{j=1}^{J} w_j z_j + b_0 = \sum_{j=1}^{J} w_j \tanh\left(\sum_{i=1}^{I} w_{ji} x_i + b_i \right) + b_0$$

Error function:

$$E^n = \frac{1}{2}(y^n - d^n)^2$$

Error back propagation algorithm:

1. Initial weights: random numbers of uniform distribution from the interval (-0.15, 0,15).
2. Presentation of input learning vectors x - the output vector y is obtained:

$$y = \sum_{j=1}^{J} w_j z_j + b_0 = \sum_{j=1}^{J} w_j \tanh\left(\sum_{i=1}^{I} w_{ji} x_i + b_i \right) + b_0$$

3. Comparison of computed **y** and desired output **d** – calculation of the error function *E*.

$$E^n = \frac{1}{2} \sum_k (y_k - d_k)^2$$

4. Second layer weights update:

   error:

$$\delta = y^n - d^n$$

   gradient:

$$\nabla E = \delta z_j$$

$$w_j(t+1) = w_j(t) - \mu(t) \delta z_j$$

5. First layer update

   error:

$$\delta_j = (1 - z_j^2) w_j \delta$$

   gradient:

$$\nabla E = \delta_j x_j$$

$$w_{ji}(t+1) = w_{ji}(t) - \mu(t) x_i (1 - z^2{}_j) w_j \delta$$

We present the input vectors and repeat the computation until the minimum error function is obtained.

REFERENCES:

1. C. M. Bishop: *Pattern Recognition and Machine Learning*, Springer 2006
2. N. Cristianini, J. Shawe-Taylor: *Support Vector Machines*, Cambridge University Press, 2000
3. N. R. Draper, H. Smith: *Applied Regression Analysis*, Wiley and Sons 1998
4. G. Dreyfus: *Neural Networks – Methodology and Applications*, Springer 2005
5. T. L. Fine: *Feedforward Neural Network Methodology*, Springer 1999
6. T. Hastie, R. Tibshirani, J. Friedman: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer 2008
7. L. Personnaz, I. Rivals *Réseaux de neurones formels pour la modélisation, la commande et la classification*, Editions CNRS, Paris 2003