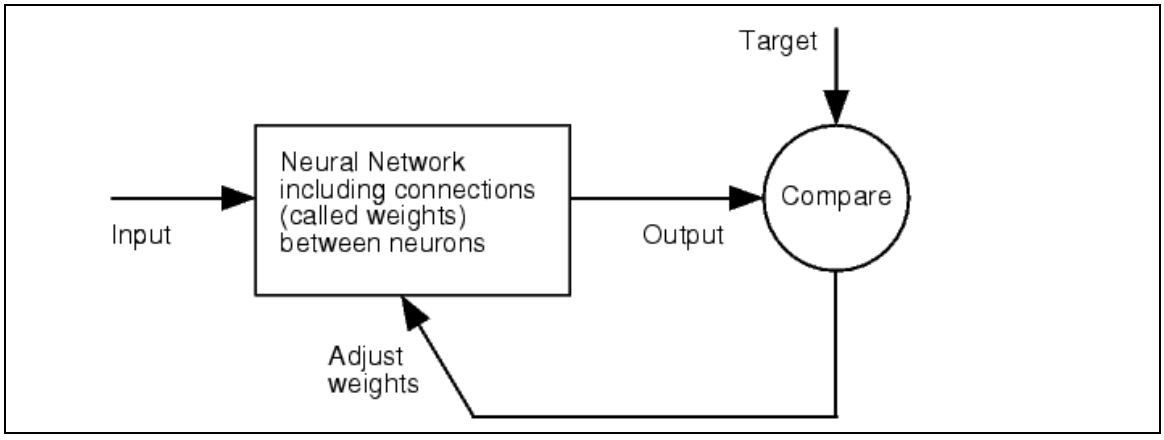
STRĘK Tomasz
Politechnika Poznańska
Instytut Mechaniki Stosowanej
ul. Piotrowo 3, 60-965 Poznań

ENGINEERING COMPUTATION BY ARTIFICIAL NEURAL NETWORKS

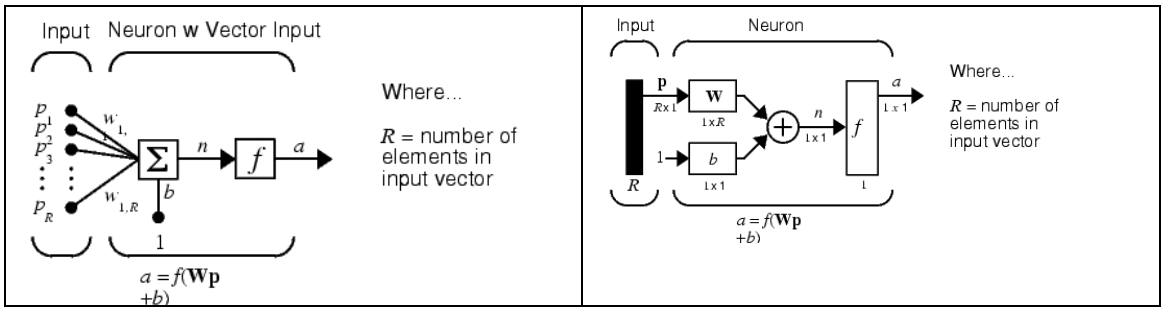
Explaining Neural Networks

Neural networks are composed of simple elements operating in parallel. These elements are inspired by biological nervous systems. As in nature, the network function is determined largely by the connections between elements. We can train a neural network to perform a particular function by adjusting the values of the connections (weights) between elements. Commonly neural networks are adjusted, or trained, so that a particular input leads to a specific target output. There, the network is adjusted, based on a comparison of the output and the target, until the network output matches the target. Typically many such input/target pairs are used, in this *supervised learning*, to train a network. Batch training of a network proceeds by making weight and bias changes based on an entire set (batch) of input vectors. Incremental training changes the weights and biases of a network as needed after presentation of each individual input vector. Incremental training is sometimes referred to as "on line" or "adaptive" training. Neural networks have been trained to perform complex functions in various fields of application including pattern recognition, identification, classification, speech, vision and control systems. Today neural networks can be trained to solve problems that are difficult for conventional computers or human beings (in engineering, financial and other practical applications).

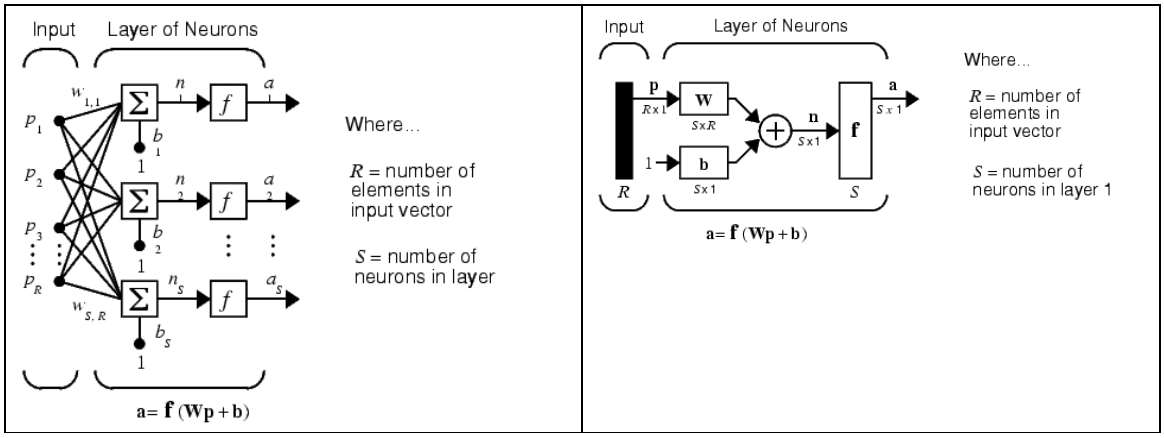
Supervising training of neural network



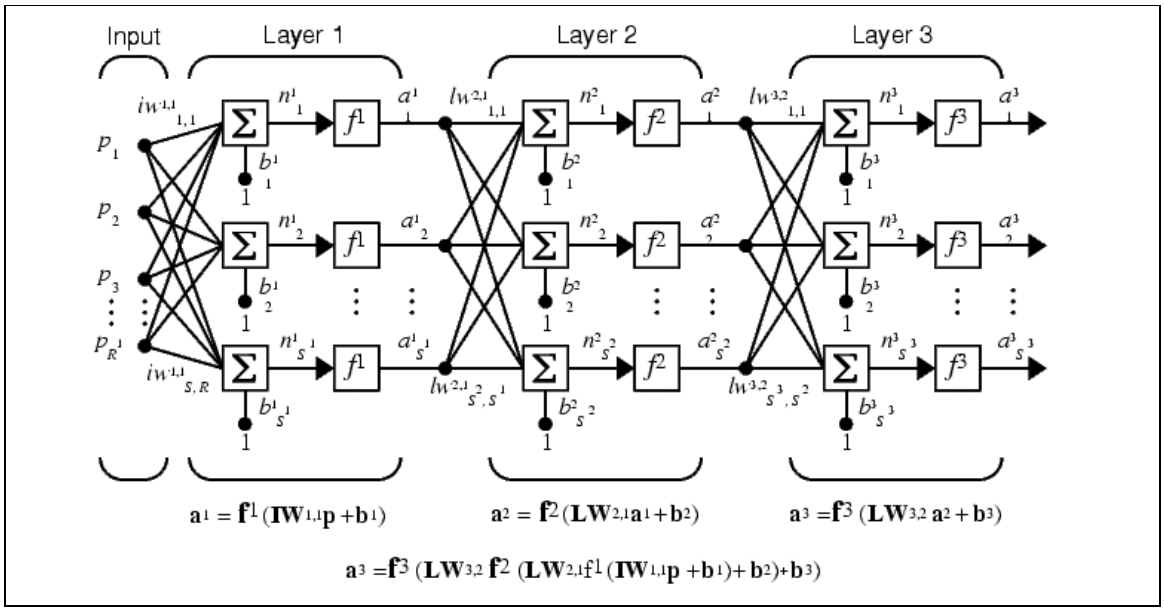
Neuron with vector input

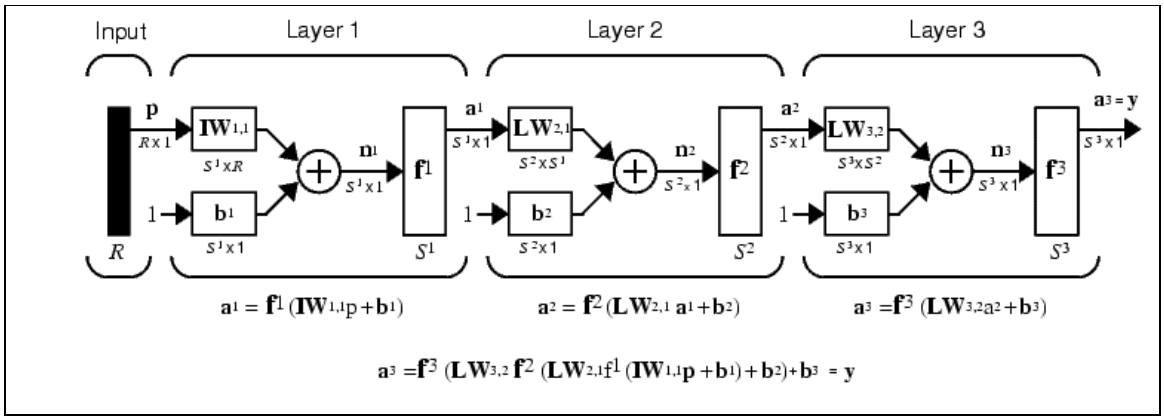


Layer of neurons



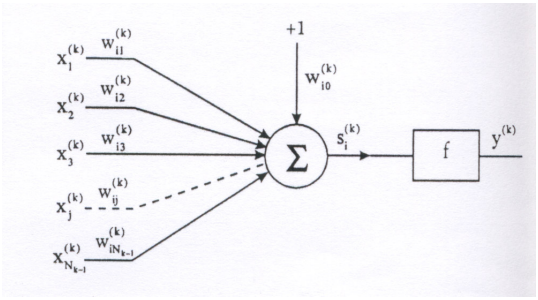
Multiple layers of neurons





Simple Neuron Model

Neuron AD_i^k ($i-th$ neuron in $k-th$ layer) has N_k inputs $\mathbf{x}^{(k)}(n)=[x_0^{(k)}(n),...,x_{N_{k-1}}^{(k)}(n)]^T$, where $x_0^{(k)}=1$ for $k=1,...,L$.



Network Architecture – Multiple Layers of Neurons

Considered multiple-layer neural network has:

- L layers with N_k neurons, $k=1,...,L$, in each layer denoted as AD_i^k, $i=1,...,N_k$;
- N_0 inputs $\mathbf{u}=[u_1(n),...,u_{N_0}(n)]^T$ $n=1,2,...$;
- N_L outputs.

Backpropagation Algorithm

The simplest implementation of backpropagation learning updates the network weights and biases in the direction in which the performance function decreases most rapidly - the negative of the gradient.

There are two different ways in which this gradient descent algorithm can be implemented: incremental mode and batch mode. In the incremental mode, the gradient is computed and the weights are updated after each input is applied to the network. In the batch mode all of the inputs are applied to the network before the weights are updated.

Let notice that inputs of neuron AD_i^k are connected with outputs of $(k-1)-th$ layer in following way

$$x_i^{(k)}(n) = \begin{cases} u_i(n) & k = 1 \\ y_i^{(k-1)}(n) & k = 2,...L \\ 1 \qquad i = 0, & k = 1,...,L \end{cases}.$$

Weights vector of neuron AD_i^k: $\mathbf{w}_i^{(k)}(n) = [w_{i0}^{(k)}(n),..., w_{iN_{k-1}}^{(k)}(n)]^T$, $n = 1,2,...$,

where $w_{i0}^{(k)}(n)$ is often called bias.

Output of AD_i^k we can define as: $y_i^{(k)}(n) = f(s_i^{(k)}(n))$

where: $$s_i^{(k)}(n) = \sum_{j=0}^{N_{k-1}} w_{ij}^{(k)}(n) x_j^{(k)}(n)$$

Transfer function (activation function) can be typically, for example:

- a step function (hard-limit function): $f(t) = \begin{cases} 1, & u > 0 \\ 0, & u \le 0 \end{cases}$;
- a linear function: $f(t) = t$;
- a sigmoid unipolar function: $f(t) = f_u(t) = \dfrac{1}{1+e^{-\beta t}}$;
- a sigmoid bipolar function: $f(t) = f_b(t) = \operatorname{tg} h(\beta t) = 2 f_u(t) - 1$.

Let notice that a unipolar sigmoid function is differentiate-able and continous:

$$\frac{df_u(t)}{dt} = \beta f_u(t)\big(1 - f_u(t)\big).$$

Let notice that outputs from last layer neurons: $y_1^{(L)}(n), y_2^{(L)}(n),..., y_{N_L}^{(L)}(n)$

are outputs from network.

We can compare them with pattern outputs: $d_1^{(L)}(n), d_2^{(L)}(n),..., d_{N_L}^{(L)}(n)$

and get errors

$$e_i^{(L)}(n) = d_i^{(L)}(n) - y_i^{(L)}(n), \; i = 1,..., N_L$$

We can formulate global error as sum of errors squares

$$Q(n) = \sum_{i=1}^{N_L} (e_i^{(L)}(n))^2.$$

We can adjust the values of the connections (weights) between elements using gradient descent algorithm

$$w_{ij}^{(k)}(n+1) = w_{ij}^{(k)}(n) - \eta \frac{\partial Q(n)}{\partial w_{ij}^{(k)}(n)}$$

where $\eta > 0$ denotes value of correction step.
We have

$$\frac{\partial Q(n)}{\partial w_{ij}^{(k)}(n)} = \frac{\partial Q(n)}{\partial s_i^{(k)}(n)} \frac{\partial s_i^k(n)}{\partial w_{ij}^{(k)}(n)} = \frac{\partial Q(n)}{\partial s_i^{(k)}(n)} x_j^{(k)}(n).$$

Let $$\delta_i^{(k)}(n) = -\frac{1}{2} \frac{\partial Q(n)}{\partial s_i^{(k)}(n)} \text{ so } \frac{\partial Q(n)}{\partial w_{ij}^{(k)}(n)} = -2\delta_i^{(k)}(n) x_j^{(k)}(n)$$

And $$w_{ij}^{(k)}(n+1) = w_{ij}^{(k)}(n) + 2\eta \delta_i^{(k)}(n) x_j^{(k)}(n)$$

The method of derive $\delta_i^{(k)}(n)$ belongs to the layer. For output (last) layer we have:

$$\delta_i^{(L)}(n) = -\frac{1}{2} \frac{\partial Q(n)}{\partial s_i^{(L)}(n)} = -\frac{1}{2} \frac{\partial \sum_{m=1}^{N_L} e_m^{(L)^2}(n)}{\partial s_i^{(L)}(n)} = -\frac{1}{2} \frac{\partial e_i^{(L)^2}(n)}{\partial s_i^{(L)}(n)} =$$

$$= \frac{1}{2} \frac{\partial \left(d_i^{(L)}(n) - y_i^{(L)}(n)\right)^2(n)}{\partial s_i^{(L)}(n)} = e_i^{(L)}(n) \frac{\partial y_i^{(L)}(n)}{\partial s_i^{(L)}(n)} = e_i^{(L)}(n) f'\left(s_i^{(L)}(n)\right)$$

For other layers ($k \neq L$) we have

$$\delta_i^{(k)}(n) = -\frac{1}{2} \frac{\partial Q(n)}{\partial s_i^{(k)}(n)} = -\frac{1}{2} \sum_{m=1}^{N_{k+1}} \frac{\partial Q(n)}{\partial s_m^{(k)}(n)} \frac{\partial s_m^{(k+1)}(n)}{\partial s_i^{(k)}(n)} =$$

$$= \sum_{m=1}^{N_{k+1}} \delta_m^{(k+1)}(n) w_{mi}^{(k+1)}(n) f'\left(s_i^{(k)}(n)\right) =$$

$$= f'\left(s_i^{(k)}(n)\right) \sum_{m=1}^{N_{k+1}} \delta_m^{(k+1)}(n) w_{mi}^{(k+1)}(n)$$

We can define error in $k-th$ ($k \neq L$) layer for $i-th$ neuron

$$e_i^{(k)}(n) = \sum_{m=1}^{N_{k+1}} \delta_m^{(k+1)}(n) w_{mi}^{(k+1)}(n), \; k = 1,\ldots,L-1$$

And we get: $$\delta_i^{(k)}(n) = e_i^{(k)}(n) f'\left(s_i^{(k)}(n)\right).$$

One of most known modification of backpropagation algorithm consist in adding extra term (momentum term) as

$$w_{ij}^{(k)}(n+1) = w_{ij}^{(k)}(n) + 2\eta \delta_i^{(k)}(n) x_j^{(k)}(n) + \alpha\left(w_{ij}^{(k)}(n) - w_{ij}^{(k)}(n-1)\right)$$

where $\alpha \in (0,1)$.

Levenberg-Marquardt Algorithm

$$\mathbf{X} = \{\mathbf{x}_i\};\ i = 1..n \qquad \mathbf{x}_i = \{x_{il}\};\ l = 1..n_x \qquad \mathbf{y} = \{y_i\};\ i = 1..n$$

$$(\mathbf{x}_i, y_i);$$

$$f(\mathbf{x}_i, \mathbf{a}) = f(\mathbf{x}_i, \mathbf{a}^{(k)}) + \sum_{j=1}^{p} Z_{ij}^{(k)} (a_j - a_j^{(k)}) + \varepsilon_i$$

$$\mathbf{a} = \{a_j\};\ \mathbf{a}^{(k)} = \{a_j^{(k)}\};\ Z_{ij}^{(k)} = \left.\frac{\partial f(\mathbf{x}_i, \mathbf{a})}{\partial a_j}\right|_{\mathbf{a}=\mathbf{a}^{(k)}} \quad i = 1..n,\ j = 1..p,\ k = 1,2,3,...$$

$$y_i = f(\mathbf{x}_i, \mathbf{a}),\ f_i^{(k)} = f(\mathbf{x}_i, \mathbf{a}^{(k)}),\ b_j^{(k)} = a_j - a_j^{(k)}$$

$$y_i - f_i^{(k)} = \sum_{j=1}^{p} b_j^{(k)} Z_{ij}^{(k)} + \varepsilon_i$$

$$\mathbf{Z}^{(k)} = \{Z_{ij}^{(k)}\},\ \mathbf{b}^{(k)} = \{b_j^{(k)}\},\ \mathbf{y} - \mathbf{f}^{(k)} = \{y_i - f_i^{(k)}\}$$

$$\mathbf{b}^{(k+1)} = \mathbf{b}^{(k)} + \mathbf{c}^{(k)}$$

$$\mathbf{c}^{(k)} = \left(\mathbf{Z}^{(k)^T} \mathbf{Z}^{(k)}\right)^{-1} \mathbf{Z}^{(k)^T} \left(\mathbf{y} - \mathbf{f}^{(k)}\right);$$

$$\mathbf{c}^{(k)} = \left(\mathbf{Z}^{(k)^T} \mathbf{Z}^{(k)} + \lambda \mathbf{I}\right)^{-1} \mathbf{Z}^{(k)^T} \left(\mathbf{y} - \mathbf{f}^{(k)}\right)$$

$$\overline{y_i} = d_0 + \sum_{j=1}^{s} d_j x_{ij} \qquad \overline{\mathbf{y}} = \mathbf{X}\mathbf{d};\ \qquad \mathbf{X} = \{x_{ij}\};\ \qquad \mathbf{d} = \{d_j\};$$

$$S = \sum_{i=1}^{n} \left(y_i - \overline{y_i}\right)^2 = (\mathbf{y} - \mathbf{X}\mathbf{d})^T (\mathbf{y} - \mathbf{X}\mathbf{d}) =$$

$$= \mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{X}\mathbf{d} - \mathbf{d}^T\mathbf{X}^T\mathbf{y} + \mathbf{d}^T\mathbf{X}^T\mathbf{X}\mathbf{d}$$

Because $\mathbf{y}^T\mathbf{X}\mathbf{d} = \left(\mathbf{y}^T\mathbf{X}\mathbf{d}\right)^T = \mathbf{y}^T\mathbf{X}\mathbf{d}$ =>

$$S = \mathbf{y}^T\mathbf{y} - 2\mathbf{d}^T\mathbf{X}^T\mathbf{y} + \mathbf{d}^T\mathbf{X}^T\mathbf{X}\mathbf{d}$$

$$\frac{\partial S}{\partial \mathbf{d}} = -2\mathbf{X}^T\mathbf{y} + \mathbf{X}^T\mathbf{X}\mathbf{d} + \left(\mathbf{d}^T\mathbf{X}^T\mathbf{X}\right)^T = -2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\mathbf{d}$$

$$\frac{\partial S}{\partial \mathbf{d}} = \mathbf{0};\ \qquad \mathbf{X}^T\mathbf{X}\mathbf{d} = \mathbf{X}^T\mathbf{y};\ \qquad \mathbf{d} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

References

- Stanisław Osowski: *Sieci neuronowe* , Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa, 2000.
- Stanisław Osowski: *Sieci neuronowe w ujęciu algorytmicznym*, WNT, Warszawa, 1996.
- D. Rutkowska, M. Piliński, L. Rutkowski: Sieci neuronowe, algorytmy genetyczne i systemy rozmyte, PWN, Warszawa, 1997.
- Duch W., Korbicz J., Rutkowski L., Tadeusiewicz R.: Biocebernetyka i inżynieria Biomedyczna 2000, Sieci neuronowe, tom 6, Akademicka Oficyna Wydawnicza Exit, Warszawa 2000.

- Hinton G.E. (2000): Computation by neural networks, Nature America, vol. 3, p.1970.
- Pietruschka U., Brause R.W. (1999): Using Growing RBF-Nets in Rubber Industry Process Control, *Neural Computing & Applications*, 8, 95-105.
- Baratti R., Cannas B., Fanni A., Pilo F. (2000): Automated Recurrent Neural Network Design to Model the Dynamics of Complex Systems, *Neural Computing & Applications,* 9, 190-201.
- Rao M. Srinivas J. (2000): Torsional Vibrations of Pre-Twisted Blades using Artificial Neural Technology, *Engineering with Computers,* 16: 10-15.
- Labonte G. (2001): Neural network reconstruction of fluid flows from tracer-particle displacements, *Experiments in Fluids,* 30: 399-409.