УДК 681.324

M. Płonkowski, Catholic University of Lublin, Poland

TRAINING NEURAL NETWORK FOR PATTERN RECOGNITION

The article presents the problems connected with the subjects of neural networks: biological prototypes of network and their adaptation to a mathematical model, network structure and its training process. The essential element of this article will be the presentation of a neural network used for recognition of letter patterns.

Introduction

Computers prove to be extremely useful in numerical calculations, in fields in which a process can be easily algorithmised. There is, however, a group of tasks, which are highly problematic for modern computers, yet completely natural and simplistic for humans. This, of course, refers to object recognition and classification tasks. The corollary of this observation was the idea of constructing an artificial brain, which would simulate some functions of the human brain.

The human brain as a prototype of neural network

The human brain consists, for the most part, of a large number of interconnected basic nerve cells – neurons. Their total number is estimated at 10^{10}.

The basic ability of neurons is to conduct and create nerve impulses. Each neuron consists of short branched dendrites, the soma and a long fibre called the axon. Dendrites have a tree structure, whereby every branch is connected to one neuron. The signals received through the dendrites reach the soma. The axon is a long appendix of the neuron, which conveys signals from the soma to other neurons. It is connected with the dendrites of other neurons by biochemical junctions called the synapses. The soma is a structure where the dendrites of the neuron converge, thus it receives and adds up the signals coming from all its inputs. If the total incoming signal exceeds a certain threshold level, the neuron becomes activated [1].

The intensity of the signal received by the neuron, which determines neuron activation, depends on the potential of signals supplied by the dendrites. These signals, received from other neurons, are modified by synapses, which can amplify or weaken the impulses. Therefore, synapses are the carriers of memory, significantly determining the functioning of the brain [2].

On the basis of the description presented above it is possible to design an artificial neuron simulating the function of the biological one.

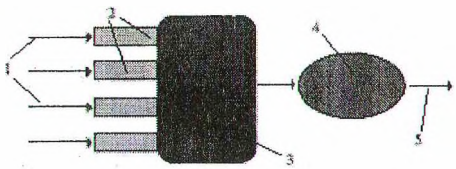


Fig. 1. A model of an artificial neuron. Explanations: 1 – inputs; 2 – weights; 3 – summation block; 4 – activation block; 5 – output

This model includes the most important elements of a biological neuron. Their artificial equivalents are: inputs (for dendrites), weights (for synapses), summation block (for nucleus), activation block (for axon hillock) and output (for axon).

Next chapters describes theory of building and learning neural network. On the basis of this information application was create. We will use it for recognition of letter patterns and more precise analysis of learning process. In particular, this application permit to: change coefficient of learning process, observe the network errors and survey speed of learning process. The results which was given by the application was present in tables and figures. Next, it was describe and analysis in detail.

Linear and non-linear neural networks

A single neuron layer is the simplest neural network. In a layer like this every neuron receives the same set of input signals $X = \langle x_1, x_2, \ldots, x_n \rangle^T$, However, each one has its own weight vector $W^{(m)} = \langle w_1^{(m)}, w_2^{(m)}, \ldots, w_n^m \rangle^T$ (where m is the number of the neuron (m = 1, 2, ...,k)). Thus, the output signal of the m^{th} neuron can be calculated from the following formula:

$$S^{(m)} = W^{(m)T} X = \sum_{i=1}^{n} w_i^{(m)} x_i .$$

The operation of a network defined in such way consists in calculating output signals of individual neurons for the same output vector X. This function reaches its maximum value when $\alpha = 0$, which means that vectors W and X have the same direction.

The network described above can recognize k different object classes. Each neuron becomes directed towards a single model object. A given input signal should be assigned to the same class as the neuron in which the maximum input signal has been reported [3]. It can be concluded that it is the weights that influence network responses in a significant way, thus determining proper functioning of a network.

Non-linear neural network consists of neurons performing non-linear mapping. This means that the activation function of such a neuron is a non-linear function. The application of this function is biologically justified, since biological neurons are non-linear.

Non-linear mapping can be described in the following way:

$$y = f(s),$$

where f is a non-linear function and its parameter s is specified in the same way as in the case of a linear neuron, namely

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

The formula above should be completed by an additional neuron's threshold coefficient, which will be subsequently referred to as the *bias*. If we choose to signify it as w_0, our formula will be as follows:

$$s = \sum_{i=0}^{n} w_i x_i .$$

In this case the input vector X will be completed with supplemented with an additional element $x_0 = 1$ [3].

Let us return to the analysis of the activation function. The unit jump function is its simplest form, represented by the equation below:

$$y(s) = \begin{cases} 1, s \geq 0 ; \\ 0, s < 0 . \end{cases}$$

A neuron with thus specified activation function is called a perceptron. Only two numbers are input values of a perceptron: 1 and 0, which can be easily interpreted as 'true' and 'false'. This kind of neuron classifies input signals by means of a logical function. A perceptron divides a n - dimensional space by a certain hyperplane into two half-spaces (which are n – dimensional). It works in such a way that the vectors accepted by the neuron belong to the first half, those rejected to the second.

The division of a plane by means of a perceptron depends on it weight values as well as on the bias coefficient, which can be interpreted as a weight with a zero index. We can see that a

perceptron is able to solve various problems with the help of a properly situated plane. Still, it does not allow any change in the character of the performed mapping, which enables it to solve only linearly separable problems. A classic example of a problem that cannot be solved by a perceptron is "the XOR problem" introduced by Minsky [1]. What cannot be done by a single neuron can still be done by a neural network consisting of several layers.

Apart from the activation function of a perceptron, a sigmoidal function is more frequently used in neural networks. It has a few useful characteristics, as far as training neural network is concerned:

- it is continuous, bounded and increasing within the (0,1) interval,
- an easily calculable and continuous derivative,
- the possibility of changing the inclination of the curve by means of the beta parameter.

The sigmoidal function can be presented by the equation:

$$y(s) = \frac{1}{1+\exp(-\beta s)}.$$

It has an easily calculable derivative:

$$\frac{dy}{ds} = y(1-y).$$

As we shall see later on in the article, the differentiability of the function is used in the network training process [4].

We have already discussed how a neuron should be constructed. Still, the way of building connections is a more important aspect, since it is the connections that determine the capability of a neural network.

In our example we shall use a three-layer neural network, containing:

- 1 input layer (network input)
- 1 hidden layer
- 1 output layer (network output)

In thus constructed network, each neuron in a given layer connects with every neuron of the next layer [4].

Being acquainted with basic information concerning neural network, we can discuss a fundamental aspect of neural network functioning, namely training a neural network.

Training neural network

The process of learning in this case is the modification of neuron weights, so that eventually the response to a given input signal is consistent with the model signal.

In order to present the training algorithm in a mathematical way, let us use the following symbols:

- L – the number of layers in the network;
- N_k – the number of neurons in the k^{th} layer, k=1,..., L;
- N_0 – the number of input signals in the network;
- $y^{(k)} = \left\langle y_1^{(k)}, \ldots, y_{N_k}^{(k)} \right\rangle$ – output signal vector in the k^{th} layer, k=1,..., L;
- $x^{(k)} = \left\langle x_0^{(k)}, \ldots, x_{N_{k-1}}^{(k)} \right\rangle$ – input signal vector of the k^{th} layer, k=1,..., L;
- $w_i^{(k)} = \left\langle w_{i0}^{(k)}, \ldots w_{iN_{k-1}}^{(k)} \right\rangle$ – weight vector of the i^{th} neuron, i=1,..., N_k in the k^{th} layer, k=1,..., L;
- $d^{(k)} = \left\langle d_1^{(k)}, \ldots, d_{N_k}^{(k)} \right\rangle$ – model signal vector of the k^{th} layer, k=1,..., L.

In the kind of learning process we have been discussing, input vector x is fed into the network's input along with the desired output vector d. Both of them constitute the so-called 'training pair.' In successive steps of network training, output vector x of the network, which is the re-

sponse to input vector x, is calculated. Given model signal d and y (the output value of the network), we can calculate the error made by the i^{th} neuron:

$$\varepsilon_i^{(k)} = d_i^{(k)} - y_i^{(k)}.$$

The total error made by the network is specified as sum of squares of the errors at the outputs of individual neurons of the last layer (L). It is also called mean square error [2]:

$$Q = \sum_{i=1}^{N_L} \varepsilon_i^{(l)2} = \sum_{i=1}^{N_L} (d_i^{(L)} - y_i^{(L)})^2 .$$

The aim of the training process is to minimise the mean square error. This can be achieved by adjusting the weights of individual vectors. The training process ends when the value of the error made by the network falls below the fixed value.

Let us now take a look at the core of training process: modification of weight vectors of individual neurons. Method of steepest ascent is the most frequently used method in determining the direction of changes of the weight vector. This algorithm assumes the direction of negative vector of the error function gradient as the direction of mineralization. It can be presented in the following way:

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

where n is the step number.

The μ coefficient specifies the scale of changes occurring during individual steps of the training algorithm. The choice of proper value for this coefficient is the subject we shall deal with later on.

The application of gradient method in training process requires knowing the error function gradient relative to all network layers. However, this task can be directly performed only in relation to the output layer, owing to the fact that only model values of the network, which are expected to appear at the output of the last layer, are given [2].

The calculation of errors made by hidden layers is carried out with the help of the back-propagation algorithm: knowing a specific error value for a given neuron we can 'project' the error back to all the neurons whose output signals constitute the input for this particular neuron [3].

Let us now derive formulas, which will be necessary in calculating weight adjustments in individual neuron layers. According to gradient method of network training, we can describe weight adjustment by the following formula:

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

The derivative of $\frac{\partial Q(n)}{\partial w_{ij}^{(k)}(n)}$ is:

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

Let us represent $\delta_i^{(k)}(n) = -\frac{1}{2}\frac{\partial Q(n)}{\partial s_i^{(k)}(n)}$.

The weight adjustment algorithm will thus take the following form:

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

The technique of calculating $\delta_i^{(k)}$ varies across individual layers. At first, we can calculate its value for the final layer of the network, since we know the desired set of values at the output.

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

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

The calculations for the previous layers are as follows:

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

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

Obviously, the calculations have to be performed from the input to the output layer if the value of $\delta_i^{(k)}$ is to be determined. For the above calculations to be possible, function f has to be differentiable. This requirement is met by the already mentioned sigmoidal function [2].

Besides the indisputable advantage of the backpropagation algorithm, namely reducing the calculation complexity of the training algorithm, there are some disadvantages. The algorithm does not guarantee that during the training process a global minimum of the error function can be found. It is possible that the training process will end in one of the global minimums. Another drawback of the backpropagation algorithm is a great number of iterations leading to convergence of the training process. What is more, if we set the training coefficient μ to too small a value, it can significantly prolong the training process. But choosing too large μ value may lead to oscillation [1].

A solution to these problems is to apply a momentum backpropagation method. It introduces a certain degree of inertia (momentum) into the process of updating the weights. The value of momentum is proportional to weight alteration in the previous iteration. It prevents too large changes of weights from occurring, hence the process becomes more fluent. A momentum backpropagation method can be described in the following way:

$$w_{ij}^{(k)}(n+1) = w_{ij}^{(k)}(n) + 2\mu\varepsilon_i^{(k)} f'(s_i^{(k)}(n)) x_j^{(k)} + \alpha[w_{ij}^{(k)}(n) - w_{ij}^{(k)}(n-1)].$$

Constant α is called the momentum coefficient. Its value is taken from the range (0,1] and determines how strongly the momentum influences weight alteration. For high α values (close to 1) the influence is very strong, for α near zero it is very weak.

A detailed analysis of coefficient settings will be provided in the next chapter.

A practical example of training a neural network

In the present chapter we shall present a neural network whose task will be to recognize 26 capital letters of the alphabet. We shall also discuss problems such as representing character patterns as vectors, choice of network architecture, network training (along with an analysis of network training parameters) and analyse the number of errors made by the network.

Each character will be represented as a 12 x 16 matrix, whose elements are only 0 and 1. '0' stands for a white pixel, '1' – a black one.

Our network will consist of three layers. Layer 1 will contain 12·16=192 neurons, for each

one is to learn only one letter of the alphabet. The choice of the number of neurons in the hidden layer can be made with the help of the following formula:

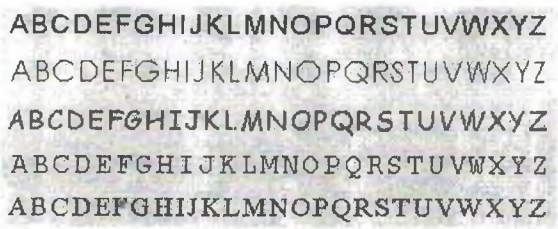
$$N_{ukryta} \approx \sqrt{N_{wejsciowa} \cdot N_{wyjsciowa}},$$

which means that the number of neurons in the hidden layer equals approximately the square root of the product of the number of neurons in the input and the output layer. In our example there are 70 neurons in the hidden layer.

Given the network architecture, we can specify the assumptions of the training process.

The training sequence will comprise 5 sets of 26 characters printed using five different fonts. Thus, it will contain 130 elements of the training sequence.

The patterns are given below:



On the basis of the above bitmaps, training sequences were made. Each character has been selected and scaled to the size of 12 by 16 pixels, according to the assumptions mentioned earlier. The training sequences will be fed into the network in random order, since presenting them in cyclical order may cause the weight adjustments to cancel out, and, consequently, slow down the process of learning.

The process of feeding the whole training sequence is called an epoch. In our considerations we have specified the number of epochs as 250. Each training process was repeated three times. The results obtained during the three tests were almost identical (they varied by approximately 0.1).

On the basis of this process, let us now analyse the influence of two coefficients: μ and α on the speed of network learning. Let us begin with the μ coefficient used in the backpropagation method.

The results of the experiment are in the table 1.

Table 1

The influence of the μ coefficient on the learning rate of the network

	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9
0	150,523	132,7311	127,1403	123,2504	121,0159	119,5726	118,4252	117,2245	115,9895
50	124,6297	124,3155	124,1655	123,8719	122,3748	110,7881	91,01443	49,87904	23,01466
100	124,6375	124,315	102,7759	19,81228	2,018052	1,015179	0,632491	0,466676	0,376304
150	124,6535	98,61039	3,347677	0,91607	0,50515	0,347953	0,263707	0,21413	0,18094
200	124,418	10,98638	0,877308	0,427995	0,277811	0,203805	0,163145	0,1365	0,117258
250	116,1293	1,729098	0,481213	0,272972	0,188995	0,142406	0,11699	0,099354	0,086073

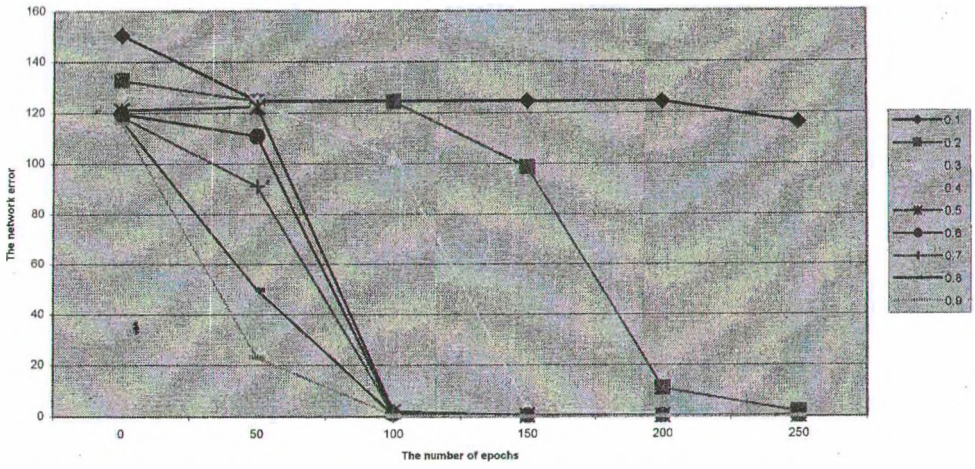


Fig. 2. The influence of the μ coefficient on the learning rate of the network

The analysis of the results shows that all coefficients, with the exception of μ=0.1, ascertain the convergence of the training process. In principle, the coefficients from the interval [0.5, 0.9] ensure a similar tempo of convergence. After only 150 epochs the network error falls below 0.5. It is a very small network error value. The network is very stable, the descent rate is fast, and with the number of epochs at the level of 150, it takes the training algorithm 17 seconds to learn (the experiment was carried out on a computer with a 2,6 GHz CPU).

How does the α coefficient influence the speed of learning? In this case we shall use apply the momentum backpropagation method. It uses two coefficients: μ and α. Investigating the value of α, we set the constant value of μ=0.5. The results are in the table2:

Table 2

The influence of the α coefficient on the learning rate of the network.

	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9
0	121,1221	121,2325	123,2759	123,6555	124,8441	126,269	128,0975	130,4903	130,7399
50	116,2281	107,7815	93,46853	51,93244	6,073188	1,26063	0,524166	1,258662	129,9997
100	1,289635	0,879033	0,627306	0,430178	0,300274	0,209532	0,132224	1,080406	129,9993
150	0,406591	0,325534	0,259389	0,198361	0,149847	0,111786	0,074842	1,047854	125,0217
200	0,233933	0,194746	0,159995	0,126803	0,098449	0,075364	0,051816	1,03395	125,0066
250	0,162204	0,137438	0,114577	0,092453	0,072829	0,056537	0,039497	1,026255	125,0039

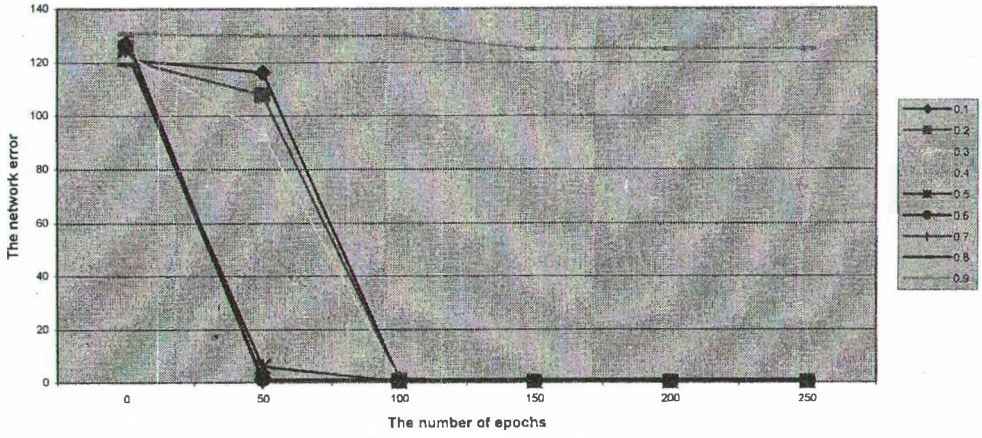


Fig. 3. The influence of the α coefficient on the learning rate of the network

The results shown above suggest the convergence of the training process for all the values of α, except for α=0,9. Even increasing the number of epochs up to 1000 does not ensure the convergence of the training process for this value of the coefficient. What follows is that the momentum amplitudes are too high. For α from the interval [0,1; 0,4] the network error level falls below 1,2 after 100 epochs. However, for α from the interval [0,5; 0,8] the error value does so after only 50 epochs. As we can see, an appropriate parameter setting can reduce the number of required training steps even by half.

Summing up both abovementioned methods, it is clear that the momentum backpropagation method considerably increases network learning rate. Notice the error value of the network trained by the backpropagation method (with μ=0,5) after 150 epochs. A network trained by the momentum backpropagation method (with μ=0,5 and α =0,7) reaches the same value of error after only 50 epochs. This yields a threefold acceleration of the learning process.

What error function value ensures almost 100% correctness of the network's responses? Multiple tests using various test sets enable us to estimate the threshold value at around 7.

REFERENCES

1. K. Kawaguchi. A multithreaded software model for backpropagation neural network applications. The University of Texas at El Paso, July 2000, http://www.ece.utep.edu/research/webfuzzy/docs/kk-thesis/kk-thesis-html/
2. W. Duch, J. Korbicz, L. Rutkowski, R. Tadeusiewicz. Sieci Neuronowe. Akademicka Oficyna Wydawnicza Exit, Warszawa, 2000.
3. R. Tadeusiewicz. Sieci Neuronowe.
4. P. Crochat, D. Franklin. Back-Propagation Neural Network Tutorial. University of Wollongong, Australia, http://ieee.uow.edu.au/~daniel/software/libneural/BPN_tutorial/BPN_English/BPN_English/