ISSN 1060-992X, Optical Memory and Neural Networks, 2017, Vol. 26, No. 1, pp. 1–17. © Allerton Press, Inc., 2017.

Deep Learning: an Overview and Main Paradigms

V. A. Golovko[a, b]

[a] *Brest State Technical University, Brest, Belarus*

[b] *National Research Nuclear University MEPhI (Moscow Engineering Physics Institute), Moscow, Russia*

e-mail: gva@bstu.by

Received July 4, 2016; in final form, October 5, 2016

Abstract—In the present paper, we examine and analyze main paradigms of learning of multilayer neural networks starting with a single layer perceptron and ending with deep neural networks, which are considered regarded as a breakthrough in the field of the intelligent data processing. The baselessness of some ideas about the capacity of multilayer neural networks is shown and transition to deep neural networks is justified. We discuss the principal learning models of deep neural networks based on the restricted Boltzmann machine (RBM), an autoassociative approach and a stochastic gradient method with a Rectified Linear Unit (ReLU) activation function of neural elements.

Keywords: deep neural networks, restricted Boltzmann machine, multilayer perceptron

DOI: 10.3103/S1060992X16040081

INTRODUCTION

There are different periods of development of neural networks, which are associated with an optimistic or pessimistic view of the evolution of this technology in the context of development of systems of artificial intelligence. Thus, in 1958 F. Rosenblatt proposed a single layer perceptron [1], which then was criticized, because it failed to solve linearly inseparable problems [2]. However, in what follows we show that a single layer perceptron with a signal activation function is able to solve the problem of the "exclusive or". After that, there was a downturn of interest in the field of perceptron neural networks, and only in 1986, it began to rise due to development of a backpropagation algorithm for a multilayer perceptron. In the scientific community then a paradigm dominated that it did not make sense to use in a perceptron more than two hidden layers. This was due to the inability of the backpropagation algorithm to give advantage when solving problems of the training of a perceptron with more than two hidden layers. This paradigm was also based on the theorem of universal approximation with a given accuracy of any function by a perceptron with one or two hidden layers. Since they used only perceptrons with few hidden layers, there was a consensus that their ability were exhausted. The more so, the support vector machines (SVM) appeared that frequently in the course of pattern recognition performed more efficiently comparing a perceptron. This was the reason why the interest to multilayer perceptrons little by little decreased. Only beginning from 2006 due to the papers of G. Hinton [3–6] the interest in multilayered neural networks started to increase again. Now they are known under a new name: deep neural networks. In the general case, these neural networks are the further development of multilayer perceptrons and they integrate different paradigms of the learning of neural networks. Due to a multilayered architecture, they allow us to process and analyze large databases as well as to model cognitive processes in different fields. In the present time, the majority of the high-tech companies use the deep neural networks for designing their intelligent systems. They are assumed a revolutionary breakthrough in the field of the artificial intelligence.

In the present paper, we examine and analyze the main paradigms of learning of the multilayer neural networks starting with single layer perceptrons and moving to deep neural networks. We show the groundlessness of some of the myths about the abilities of multilayer neural networks and justify the transition to deep neural networks. We discuss the main models of learning of deep neural networks based on the restricted Boltzmann machine (RBM), an autoencoder approach and a stochastic gradient descent method (SGD).

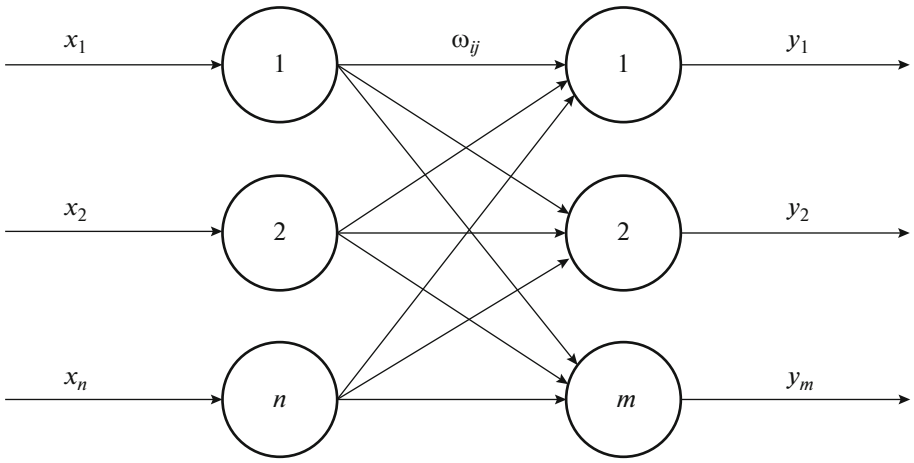


Fig. 1. Single layer perceptron.

1. SINGLE LAYER PERCEPTRON

In this Section, we discuss a single layer perceptron, its training and potential. At present time, there is strong opinion that with the aid of single layer perceptrons the problem of the "exclusive or" cannot be solved. This is true for a perceptron with a threshold or a monotonically increasing activation function. However, in what follows we show that when using a signal activation function of neural elements a single layer perceptron is able to solve the problem of the "exclusive or".

It is accepted to depict a single layer perceptron in the form of a two-layer neural network where the first layer of the neuron elements is a distributing layer and the second one is a processing one. The distributing layer transmits input signals on the processing layer of the neuron elements, which transforms the input information in accordance with synaptic connections and the activation function (Fig. 1). At that, each neuron from the distributing layer has the synaptic connections with all the neurons of the processing layer.

The output value of the j-th neuron element of the second layer can be presented as

$$y_j = F(S_j) = F\left(\sum_{i=1}^{n} \omega_{ij} x_i + T_j\right), \tag{1}$$

where T_j is the threshold of the j-th neuron element, ω_{ij} is the strength of the synaptic connection between the i-th neuron of the distributing layer and the j-th neuron of the processing layer, F is the operator of the nonlinear transformation or the activation function of the neuron elements.

Let us examine the training rule of the single layer perceptron, which has the linear activation function. This training rule is known as a delta rule. It can be used for training both the linear perceptron and the perceptron with a threshold activation function. In the 60-th the delta rule was proposed by *F.* Rosenblatt [1] for perceptron with threshold activation functions, and Widrow and Hoff [7] suggested it for linear networks.

The output value of the linear single layer perceptron is defined as

$$y_j = \sum_{i=1}^{n} \omega_{ij} x_i + T_j. \tag{2}$$

According the delta rule the total mean-square error of the neural network has to be minimized. For L input patterns, we define it as

$$E_s = \sum_{k=1}^{L} E(k) = \frac{1}{2} \sum_{k=1}^{L} \sum_{j=1}^{m} (y_j^k - e_j^k)^2, \tag{3}$$

where $E(k)$ is the mean-square error of the network for the k-th pattern; y_j^k and e_j^k are the input and the reference values for the k-th pattern, respectively.

The gradient descent method is used to minimize the total mean-square error. There are two main approaches to training of a single layer perceptron. They are an online learning and a batch learning. Under the online training, a modification of synaptic connections takes place after input of each pattern from the learning set to the neural network. In this case, in the gradient descent method we use the mean-square error of the neural network for one input pattern:

$$E = \frac{1}{2}\sum_{j=1}^{m}(y_j - e_j)^2. \tag{4}$$

Then in line with the gradient descent method, the weight coefficients and thresholds of the neural network have to be changed with time according the rules:

$$\omega_{ij}(t+1) = \omega_{ij}(t) - \alpha\frac{\partial E}{\partial \omega_{ij}(t)}, \tag{5}$$

$$T_j(t+1) = T_j(t) - \alpha\frac{\partial E}{\partial T_j(t)}, \tag{6}$$

where α is rate or step of learning.

From here, we obtain **the delta rule** for the online training:

$$\omega_{ij}(t+1) = \omega_{ij}(t) - \alpha(y_j - e_j)x_i, \tag{7}$$

$$T_j(t+1) = T_j(t) - \alpha(y_j - e_j). \tag{8}$$

Under the online learning, the value of the adaptive step is [8]:

$$\alpha(t) = \frac{1}{1 + \sum_i x_i^2(t)}.$$

When using the batch learning, modifications of the synaptic connections take place after we input L patterns from the learning set to the network. In this case, we can write the delta rule in the form [8]:

$$\omega_{ij}(L) = \omega_{ij}(0) - \alpha(t)\sum_{k=1}^{L}(y_j^k - e_j^k)x_i^k, \quad T_j(L) = T_j(0) - \alpha(t)\sum_{k=1}^{L}(y_j^k - e_j^k). \tag{9}$$

Under the batch learning, the value of the adaptive step is [8, 9]:

$$\alpha(t) = \frac{\sum_k\sum_j (y_j^k - e_j^k)a_j^k}{\sum_k\sum_j (a_j^k)^2}, \quad a_j^k = \sum_p (y_j^p - e_j^p)\left(1 + \sum_i x_i^p x_i^k\right). \tag{10}$$

From the point of view of pattern classification, the single layer perceptron with the threshold activation function forms a linear separating surface; and that is the reason why it cannot solve the problem of the "exclusive or". American scientists M. Minsky and S. Papert showed this in due time [2]. Their conclusions about the further development of neural networks were pessimistic. However, the last statement is true only for a single layer perceptron with a threshold or a monotonic continuous activation function (for example, a sigmoid function). When one uses the signal activation function, the single layer perceptron is able to solve the problem of the "exclusive or" [10]. Let us disclose this statement. In this case, it is necessary to select the area of ones and zeros with the aid of two straight lines (Fig. 2).

The area A that characterizes the class of ones is defined by a condition: $S_1 > 0 \wedge S_2 < 0$. Equations of the straight lines can be written in the form:

$$S_1 = X_2 + X_1 - 0.5,$$

$$S_2 = X_2 + X_1 - 1.5.$$

From here we obtain that $S_2 = S_1 - 1$ (Fig. 3).

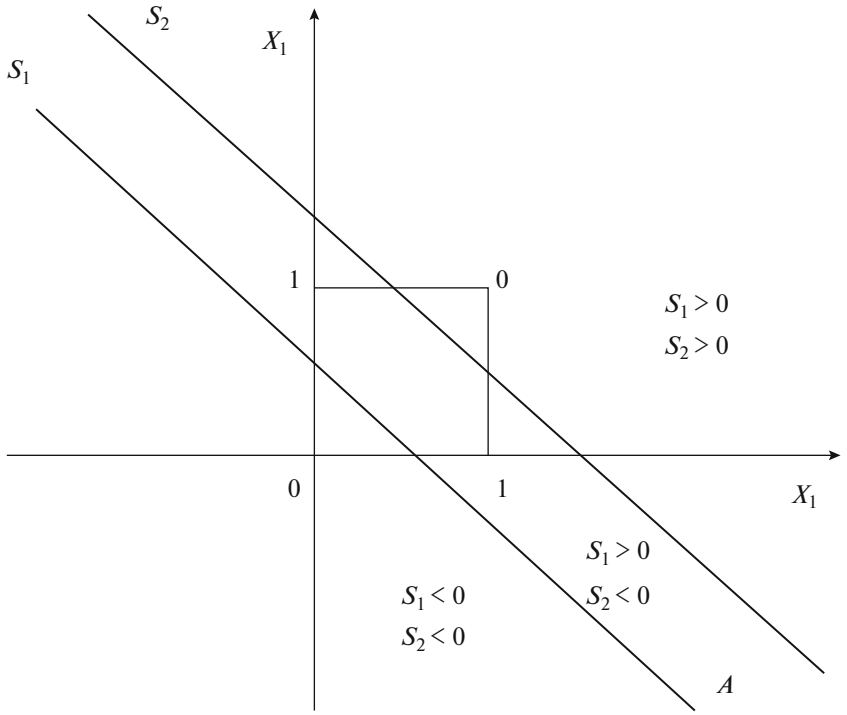


Fig. 2. Solution of problem of "exclusive or".

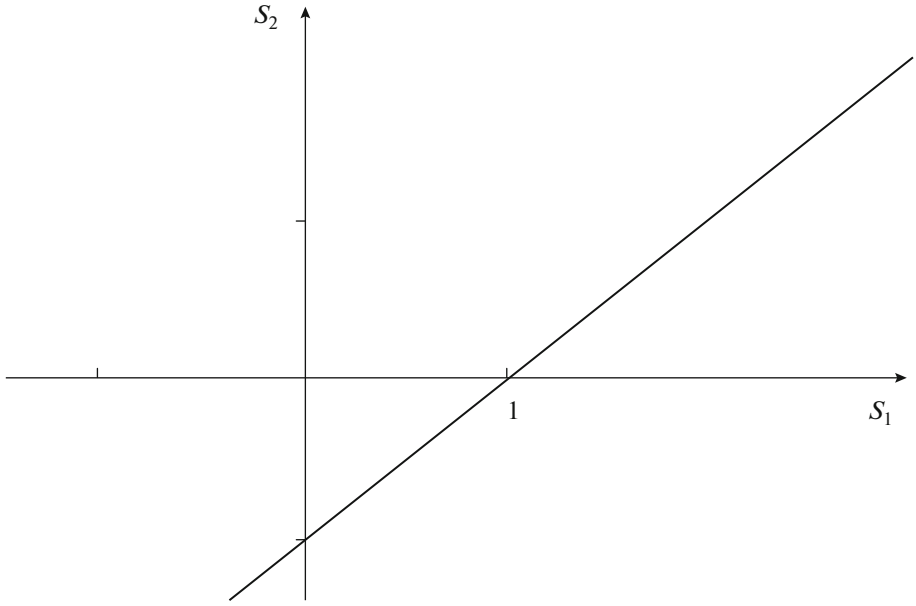


Fig. 3. Dependence of S_2 on S_1.

Then if $S_1 > 0 \wedge S_2 < 0 \rightarrow 0 < S_1 < 1$, we obtain the following activation function (Fig. 4):

$$y = F(S) = F(S_1) = \begin{cases} 1, & 0 < S_1 < 1 \\ 0, & \text{otherwise} \end{cases}. \tag{11}$$

As a result, the single layer perceptron for solving the problem of the "exclusive or" has the following weights and the threshold (Fig. 5):

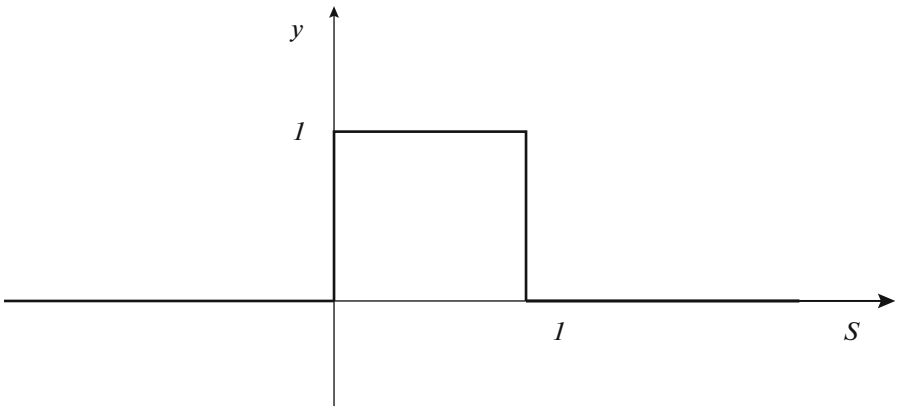


Fig. 4. Signal activation function.

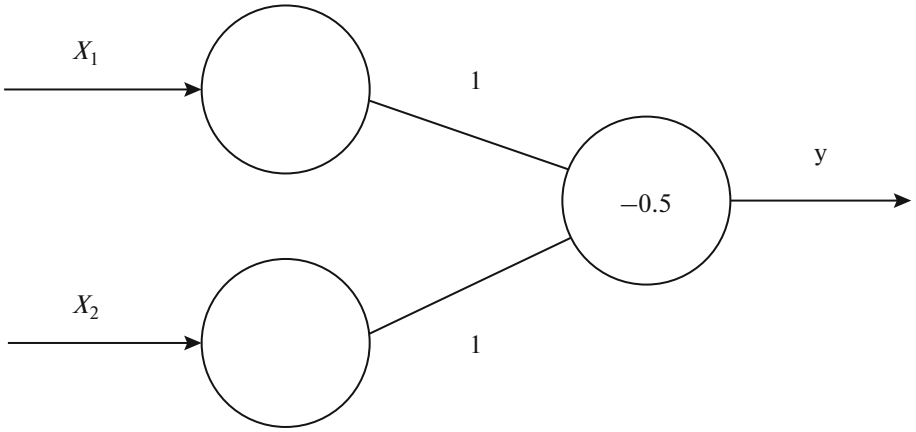


Fig. 5. Single layer perceptron for solving problem of "exclusive or".

Thus, if the signal activation function is used the single layer perceptron is able to solve the problem of the "exclusive or" since this function divides the input space of patterns into classes with the aid of two straight lines. Similarly, one can solve the problem of the "exclusive or" using a radial basis activation functions of neural elements.

2. MULTILAYER PERCEPTRON

In 1986, a number of authors (Rumelhart, Hinton, Williams) suggested independently a backpropagation algorithm, which became a powerful instrument for training of multilayer neural networks [11]. A perceptron with one hidden layer is shown in Fig. 6.

We define the output value of the j-th neuron of the last layer as

$$y_j = F(S_j),$$
$$S_j = \sum_i \omega_{ij} y_i + T_j.$$

Similarly for the hidden layer

$$y_i = F(S_i),$$
$$S_i = \sum_k \omega_{ki} x_k + T_i,$$

where S_j is a weighted sum for the neuron j, ω_{ij} is the weight coefficient of the i-th neuron on the j-th neuron, T_j is the threshold of the j-the neuron of the last layer.

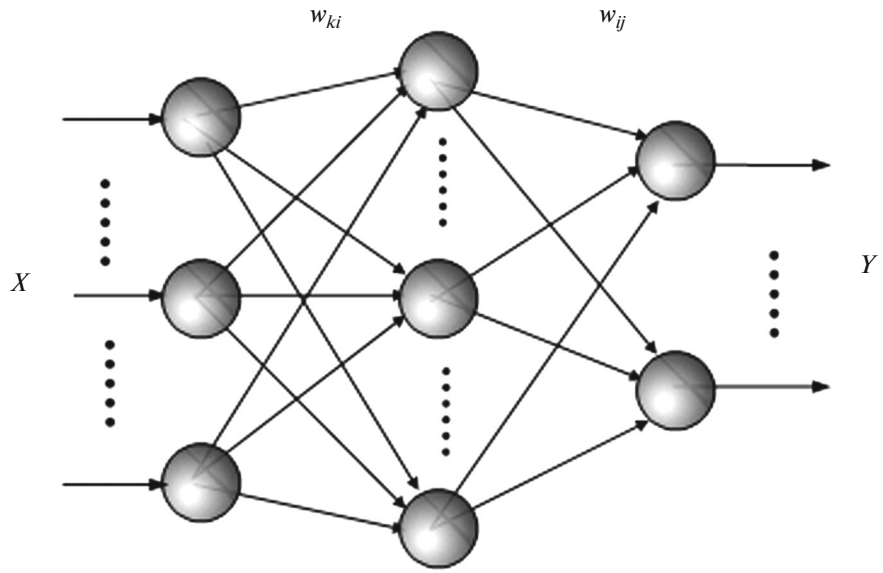


Fig. 6. Perceptron with one hidden layer.

In the case of the online training, to minimize the mean-square error the weight coefficients and thresholds have to change with time as [8]:

$$\omega_{ij}(t+1) = \omega_{ij}(t) - \alpha\gamma_j F'(S_j) y_i, \tag{13}$$

$$T_j(t+1) = T_j(t) - \alpha\gamma_j F'(S_j), \tag{14}$$

where $\frac{\partial E}{\partial S_j} = F'(S_j)$, γ_j is the error of the j-th neuron of the j-th layer.

For the output layer

$$\gamma_j = y_j - e_j.$$

For any i-th hidden layer the error of the i-th neuron element is determined recursively through the errors of neurons of the next layer j:

$$\gamma_i = \sum_{j=1}^{m} \gamma_j F'(S_j)\omega_{ij}, \tag{15}$$

where где m is the number of neurons of the next layer with respect to the i-th layer; ω_{ij} is the synaptic connection between the i-th and j-th neurons of different layers; S_j is the weighted sum of the j-th neuron.

This learning rule of the multilayer neural networks is called the **generalized delta rule**.

Under the batch learning when the synaptic connections modify after L patterns are fed to the neural network we have

$$\omega_{ij}(t+1) = \omega_{ij}(t) - \alpha\sum_k \gamma_j^k F'(S_j^k) y_i^k, \tag{16}$$

$$T_j(t+1) = T_j(t) - \alpha\sum_k \gamma_j^k F'(S_j^k). \tag{17}$$

The number of layers of the multilayer neural network characterizes how the input space can be divided into subspaces of lower dimensions. Thus as we have shown in the previous Section, a single layer perceptron divides the input space of patterns into classes with the aid of a **hyperplane**. The perceptron with one hidden layer and a nonlinear activation function of neuron elements allows us to generate in the space of solutions both convex and nonconvex dividing surfaces. With the aid of a perceptron with two or more hidden layers, we also can obtain solution regions of any form and complexity including the nonconvex ones.

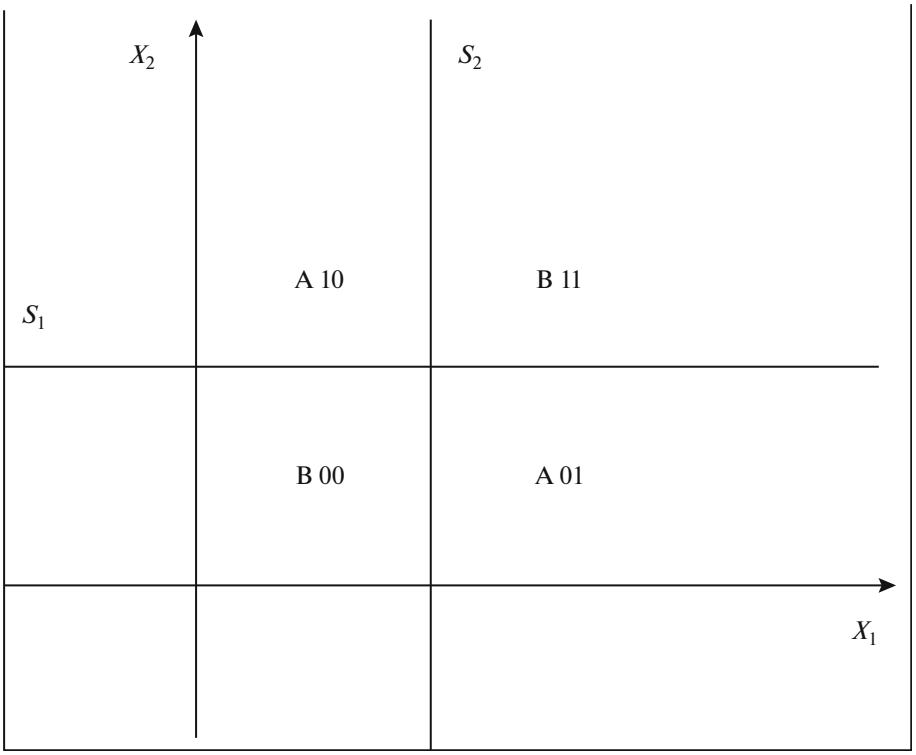


Fig. 7. Problem of binary classification of two classes A and B.

Let us examine the dependence of the capability a multilayer perceptron on the number of hidden layers. With regard to this problem, there are a lot of errors in the literature. For example, authors of many papers [12] state that from the point of view of classification of patterns the perceptron with one hidden layer can generate a convex dividing surface only. This inaccurate statement got into textbooks however later on it was shown that such perceptron could form a nonconvex dividing surface.

First, it worth to be noted that the capabilities of the perceptron with one hidden layer differ depending on the used activation function. In 1989, G. Cubenko proved a theorem about the universal approximation [13]. In the general case, it can be formulated as follows:

Theorem. A perceptron with one hidden layer is **the universal approximator**. In other words, it can approximate any continuous function with any degree of accuracy when using a monotonically-increasing continuous and bounded function as the activation function of the neuron elements of the hidden layer. At that, the accuracy of approximation depends on the number of neurons in the hidden layer. The more the number of neurons in the hidden layer, the better the accuracy of approximation of the function. However, if the dimension of the hidden layer is too large a phenomenon known as the overfitting of the network can take place. In this case, the neural network has poor generalization ability.

This is the main theorem for approximation and classification of functions with the aid of multilayer neural networks. From the theorem, it follows that as the activation functions one can use sigmoidal functions: the sigmoid function, the bipolar sigmoid function and the hyperbolic tangent.

Consequently, the neural network with one hidden layer allows one to perform any mapping of the input signals into the output ones if as the activation function we use the sigmoid function or any bounded, continuous and monotonically-increasing function. This is why such neural network is also the universal classifier: it can generate an arbitrary dividing surface in the space of solutions.

Let us examine another case when as the activation function one uses a threshold function. The neurons of the hidden layer are the local feature detectors. They divide the input space of patterns into the regions with the aid of hyperplanes (the straight lines in the two-dimensional case). The output layer of the neural elements unites the obtained regions into the relevant classes. It is easy to show that a perceptron with one hidden layer and a binary activation function allows us to generate a nonconvex dividing surface in the space of solutions, but it is not the universal classifier [10]. In this case, only the perceptron with two hidden layers is the universal classifier. We show that by the example of solving a simple problem of binary classification (Fig. 7).

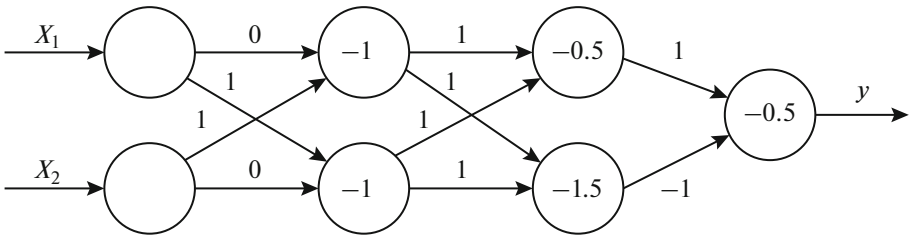


Fig. 8. Perceptron with two hidden layers for solution of binary classification problem.

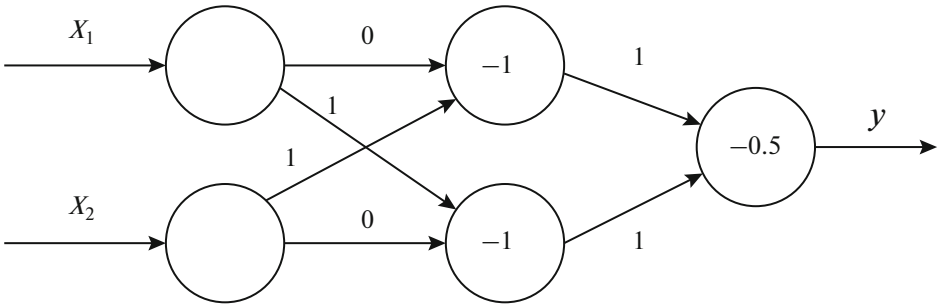


Fig. 9. Perceptron with one hidden layer for solution of binary classification problem.

As we see from the figure, the classes are divided by two straight lines S_1 and S_2. The equations of these curves are $S_1 = X_2 - 1$ and $S_2 = X_1 - 1$. From Fig. 7 it follows that as a result the given problem is reduced to the problem of the "exclusive or". This is why for solution of this problem the perceptron has to be composed of two hidden layers, where the first hidden layer generates the dividing straight lines S_1 and S_2 (Fig. 8).

Let us examine one more case when we use the signal activation function in some layers of neuron elements. Then as it is shown in Fig. 9, when solving the classification problem formulated above the perceptron with one hidden layer and the signal activation function of the output neuron is enough. Here the neurons of the hidden layer generate discriminant straight lines S_1 and S_2, and the output neuron element with the signal activation function forms the final solution of the problem.

So up to 2006 the main paradigm of learning of multilayer neural networks was as follows: the perceptron with a maximum of one or two hidden layers was sufficient for solution of different problems. Using of the perceptron with more than two hidden layers made little sense.

3. DEEP NEURAL NETWORKS

In the general case, deep neural networks are neural networks with multiple layers of neuron elements [3–6, 14–21]. There are the following deep neural networks:

- deep belief neural network
- deep perceptron
- deep convolutional neural network
- deep recurrent neural network
- deep autoencoder
- deep recurrent convolution neural network (deep R-CNN).

Historically, the first were the deep belief neural networks and the deep perceptron. In the general case, they are a multilayer perceptron with more than two hidden layers. The main difference of the deep perceptron from the deep belief neural network is that in general the deep belief neural network is not a feed forward neural network. Up to 2006 in the scientific community priority was the paradigm according which a multilayer perceptron with a maximum of one or two hidden layers was more effective for a nonlinear transformation of the input space of patterns into the output one comparing with a perceptron with a larger number of hidden layers. It was believed that it is no sense to use a perceptron with more than two hidden layers. This paradigm based on the theorem about the perceptron with one hidden layer being the

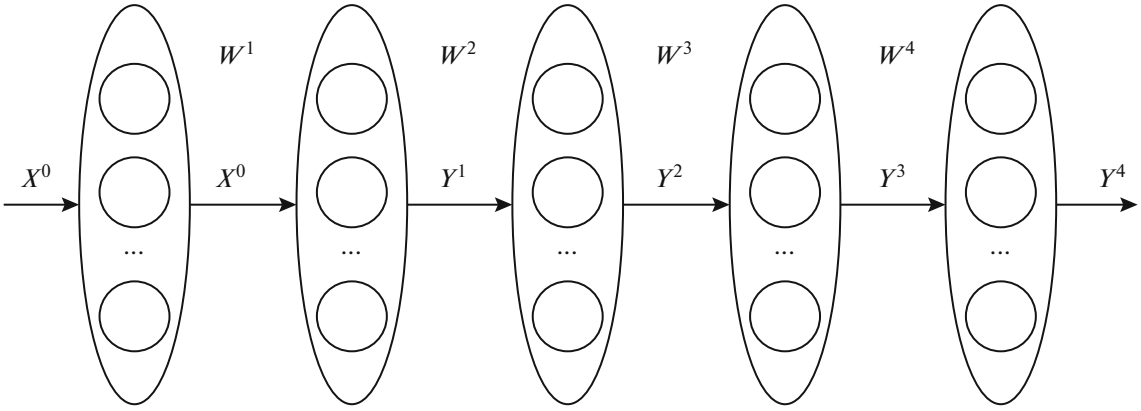


Fig. 10. Deep neural network.

universal approximator. The other aspect of this problem was that no attempts to use the backpropagation algorithm when training a perceptron with three or more hidden layers resulted in an improvement of the problem's solution. The matter is that when a sigmoidal activation function is used for training of a perceptron with three or more hidden layers the backpropagation algorithm is not effective. This happens because of the vanishing gradient problem. For example, the maximal value of the derivative of a sigmoid activation function is $F'(S_j)$=0.25. Then the use of equations (13) and (14) to train a perceptron with large number of hidden layers leads to vanishing of the gradient when the signal propagates from the last to the first layer. In 2006, Hinton proposed a greedy layer-wise algorithm [3], which became an effective method for training of deep neural networks, which as it was mentioned above, was a perceptron with large number of hidden layers. It was shown that comparing with the traditional perceptron the deep neural network performs nonlinear transformations and presentation of data more effectively. Such a network performs a deep hierarchical transformation of the input space of patterns. As a result, the first hidden layer extracts a low-level space of features of input data; the second layer detects a space of features of higher level of abstraction and so on.

As we mentioned above, a deep neural network contains a great number of hidden layers of neuron elements (Fig. 10) and performs a deep hierarchical transformation of the input space of patterns.

The output value of the j-th neuron of the k-the layer is defined as

$$y_j^k = F(S_j^k), \tag{18}$$

$$S_j^k = \sum_{i=1} w_{ij}^k y_i^{k-1} + T_j^k, \tag{19}$$

where F is the activation function of the neuron element, S_j^k is the weighted sum of the j-th neuron of the k-the layer, w_{ij}^k is the weight coefficient between the i-th neuron of the $(k-1)$-th layer and the j-th neuron of the k-th layer, T_j^k is the threshold of the j-th neuron of the k-th layer.

For the first (distributing) layer

$$y_i^0 = x_i. \tag{20}$$

In the matrix form the output vector of the k-th layer is

$$Y^k = F(S^k) = F(W^k Y^{k-1} + T^k), \tag{21}$$

where W is the matrix of the weight coefficients, Y^{k-1} is the output vector of the $(k-1)$-th layer, T^k is the vector of the threshold values of neurons of the k-th level.

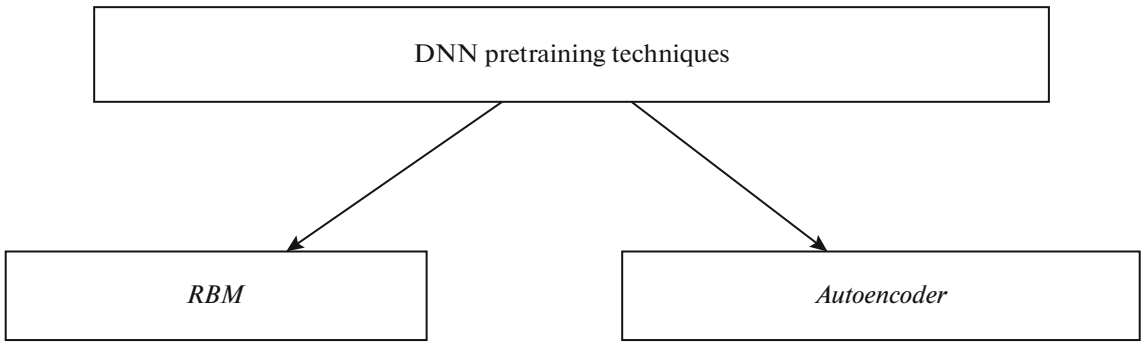


Fig. 11. Methods of pre-training of deep neural networks.

If the deep neural network is used for classification of patterns, the output values of the neural network are frequently determined basing on a softmax activation function:

$$y_j^F = \text{softmax}(S_j) = \frac{e^{S_j}}{\sum_l e^{S_l}}. \quad (22)$$

In spite of architectural differences of deep neural networks, the principles of their training are identical.

4. LEARNING OF DEEP NEURAL NETWORKS

Let us discuss learning of deep neural networks. There are two principal methods of learning [14–18]:

1. Method with pre-training that consists of two stages:

Unsupervised pre-training of a neural network by means of the layer-by-layer method of training starting with the first layer.

Fine-tuning of all the synaptic connections of the neural network with the aid of the backpropagation algorithm or the wake-sleep algorithm.

2. The stochastic gradient method (SGD) with the Rectified Linear Unit (ReLU) activation function of neural elements [18].

An important stage of learning of a deep belief neural network is pre-training of its layers. There are two principle approaches to pre-training of the layers of such neural networks (Fig. 11).

The first one is the autoencoder approach. It is based on the representation of each layer in the form of an autoassociative neural network. The second approach is based on the representation of each layer of the neural network in the form of the restricted Boltzmann machine (RBM).

5. AUTOENCODER METHOD OF LEARNING

In this case, with the purpose of minimization of the total mean-square error when reconstruction the information we at first train the first layer as the autoassociative neural network. Then the second layer is trained and so on. To train each layer the backpropagation algorithm can be used. After that, we perform the fine-tuning of all the synaptic connections of the neural network using the backpropagation algorithm.

Let us examine a perceptron with three hidden layers (Fig. 12). Then in accordance with the autoencoder method, we first take two first layers of the neural network (1 and 2) and on their basis, we generate the autoassociative neural network (1-2-1) (the PCA neural network). Thus, the reconstructing layer is added (Fig. 13). Next, to minimize the mean-square error of reconstruction of information we train such a network with the aid, for example, of the backpropagation algorithm. The duration of the training is not more than 100 epochs.

After that, we reject the reconstructing (the last) layer, fix the weights of the hidden layer and generate the autoassociative neural network from the next two layers of the neural network (2-3-2). The autoassociative neural network is trained basing on data received from the previous (the second) layer. The process continues until the last or the next to the last layer as it schematically shown in Fig. 13. The result of the

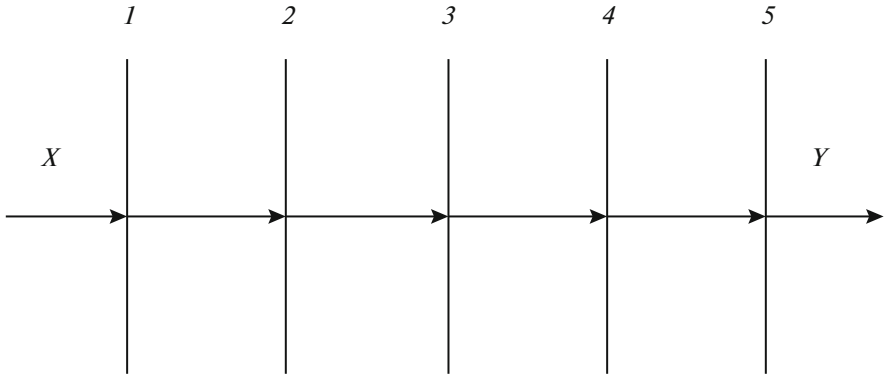


Fig. 12. Perceptron with three hidden layers.

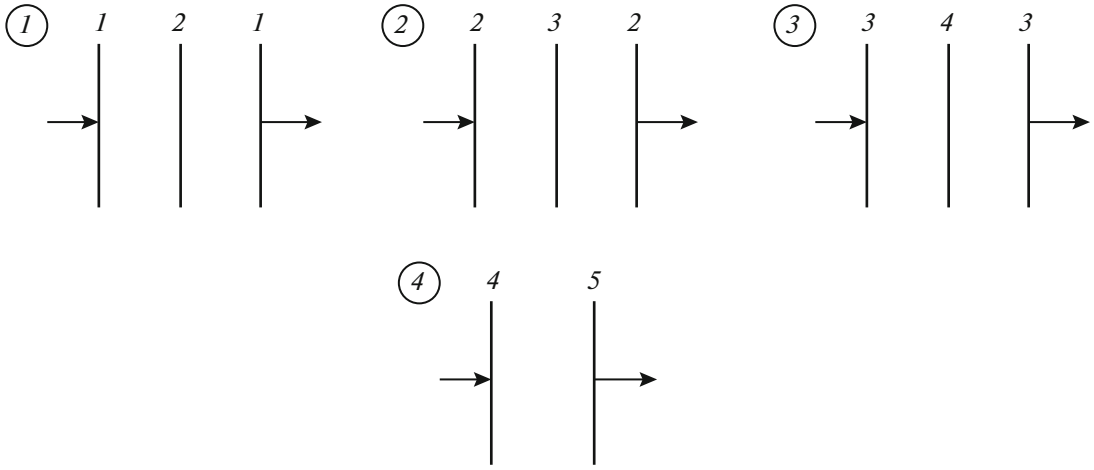


Fig. 13. Autoencoder method of training.

layer-by-layer training is a pre-trained neural network. Further fine-tuning is carried out by means of, for example, the supervised backpropagation algorithm.

This process can be pictured as the following algorithm:

(1) We generate the autoassociative neural network with the output layer X, the hidden layer Y and the output layer X.

(2) The autoassociative neural network is trained with the aid of, for example, the backpropagation algorithm (as a rule, not more than 100 epochs are necessary); and the synaptic connections W^1 of the first layer are fixed.

(3) We take the next layer; and in much the same way, we generate the autoassociative neural network.

(4) Using the tuned synaptic connections W^1 of the previous layer we feed the input data to the second autoassociative neural network and train it similarly. As a result, we obtain the weight coefficients W^2 of the second layer.

(5) The process continues until the last layer of the neural network.

(6) The neural network as a whole is trained for the fine-tuning of the parameters by means of the backpropagation algorithm.

6. TRAINING OF DEEP NEURAL NETWORKS BASED ON RBM

As we mentioned above under this approach we present each layer of the neural network as the restricted Boltzmann machine (RBM) [3–6]. The restricted Boltzmann machine consists of two layers of stochastic binary neuron elements that are interconnected by bidirectional symmetric connections

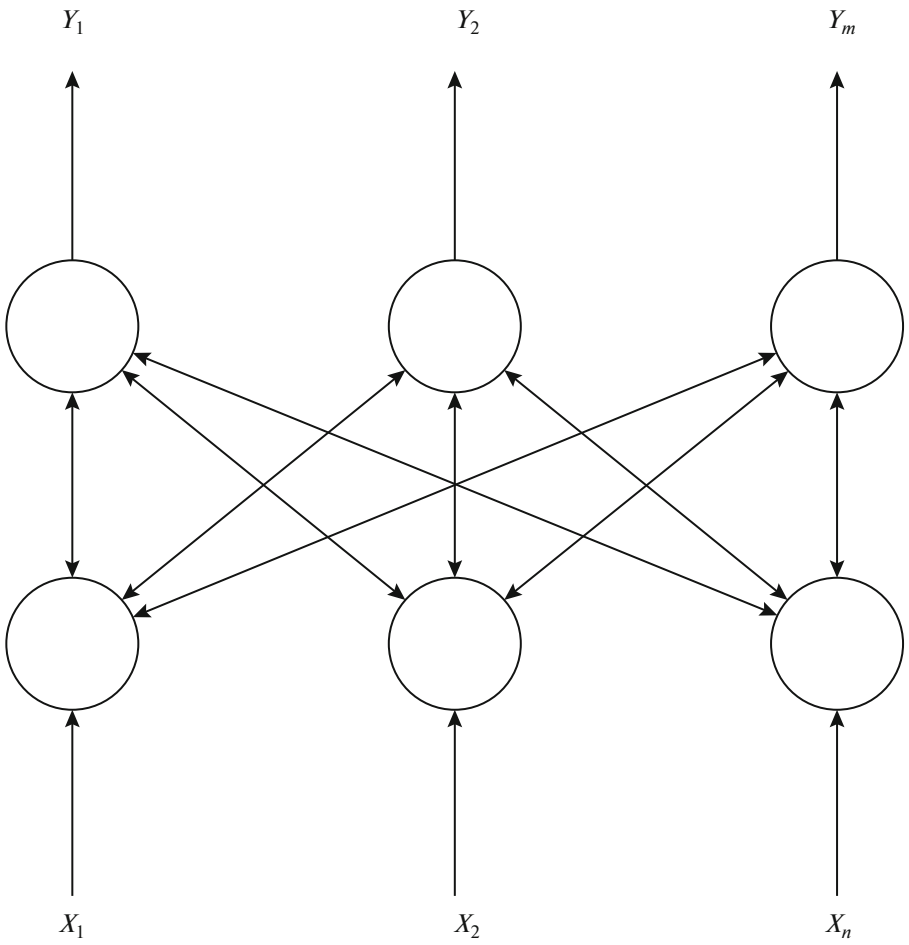


Fig. 14. Restricted Boltzmann machine.

(Fig. 14). The input layer of the neuron elements is called a visible layer (the layer X) and the second layer is called a hidden layer (the layer Y). The deep neural network can be formed of a set of the restricted Boltzmann machines. When we use a sufficient number of the neurons of the hidden layer, the restricted Boltzmann machine can model any discrete distribution [14].

The given neural network is a stochastic neural network where the sates of the visible and hidden neurons change according to a probabilistic version of the sigmoid activation function:

$$p(y_j|x) = \frac{1}{1+e^{-S_j}}, \quad S_j = \sum_{i=1}^{n} w_{ij}x_i + T_j, \tag{23}$$

$$p(x_i|x) = \frac{1}{1+e^{-S_i}}, \quad S_i = \sum_{j=1}^{m} w_{ij}y_j + T_i. \tag{24}$$

The states of the visible and hidden neuron elements are supposed to be independent:

$$P(x|y) = \prod_{i=1}^{n} P(x_i|y),$$

$$P(y|x) = \prod_{j=1}^{m} P(y_j|x).$$

Thus, the states of all the neuron elements of the restricted Boltzmann machine are defined in terms of the probability distribution. In RBM the neurons of the hidden layer are detectors of features that store the characteristics of the input data. The main idea of the training is to reproduce the distribution of the input data using the states of the hidden units as closely as possible. This is equivalent to maximizing the likelihood of the input data distribution P(x) by the modification of synaptic weights using the gradient of the log probability of the input data. Let us consider this in more detail.

The probability of a joint configuration over both visible and hidden units (x, y) is defined by the Gibbs distribution:

$$P(x,y) = \frac{e^{-E(x,y)}}{Z},$$

where $E(x, y)$ is the energy of the system in the state (x, y), Z is the normalization factor, which ensures that the total probability is equal to one. This factor is defined as

$$Z = \sum_{x,y} e^{-E(x,y)}.$$

The probability to find the visible neurons in the given state is equal to the sum of probabilities of the configurations $P(x, y)$ over the states of the hidden neurons:

$$P(x) = \sum_{y} P(x,y) = \sum_{y} \frac{e^{-E(x,y)}}{Z} = \frac{\sum\limits_{y} e^{-E(x,y)}}{\sum\limits_{x,y} e^{-E(x,y)}}.$$

To estimate the maximum likelihood of the data distribution $P(x)$, the RBM should train by taking the gradient of the log probability of the input data with respect to the weights and thresholds. The log-likelihood of the input data distribution is defined as follows:

$$\ln P(x) = \ln \sum_{y} e^{-E(x,y)} - \ln \sum_{x,y} e^{-E(x,y)}.$$

Then the log-likelihood gradient can be written as

$$\frac{\partial \ln(P(x))}{\partial \omega_{ij}} = \frac{\partial}{\partial \omega_{ij}} \left(\ln \sum_{y} e^{-E(x,y)} \right) - \frac{\partial}{\partial \omega_{ij}} \left(\ln \sum_{x,y} e^{-E(x,y)} \right).$$

Transforming the last expression, we obtain

$$\frac{\partial \ln P(x)}{\partial \omega_{ij}} = -\frac{1}{\sum\limits_{y} e^{-E(x,y)}} \sum_{y} e^{-E(x,y)} \frac{\partial E(x,y)}{\partial \omega_{ij}} + \frac{1}{\sum\limits_{x,y} e^{-E(x,y)}} \sum_{x,y} e^{-E(x,y)} \frac{\partial E(x,y)}{\partial \omega_{ij}}.$$

Since

$$P(x,y) = P(y|x)P(x),$$

then

$$P(y|x) = \frac{P(x,y)}{P(x)} = \frac{\frac{1}{Z} e^{-E(x,y)}}{\frac{1}{Z} \sum\limits_{y} e^{-E(x,y)}} = \frac{e^{-E(x,y)}}{\sum\limits_{y} e^{-E(x,y)}}.$$

As a result, we obtain the following expression:

$$\frac{\partial \ln P(x)}{\partial \omega_{ij}} = -\sum_{y} P(y|x) \frac{\partial E(x,y)}{\partial \omega_{ij}} + \sum_{x,y} P(x,y) \frac{\partial E(x,y)}{\partial \omega_{ij}}.$$

In this expression, the first summand defines a positive phase of the Boltzmann machine training when the neural network works basing on the patterns from the training set. The second summand characterize a negative phase of the training, when the neural network runs freely independently of the external data.

Let us examine the energy of the RBM neural network. In the view of the energy of the system, the problem of the learning of the neural network is to find the configuration of the output data with the minimal energy basing on the input data. As a result, at the training set the energy of the neural network will be less comparing with other states. The energy of the binary state (x,y) is defined the same as for the Hopfield model:

$$E(x,y) = -\sum_i x_i T_i - \sum_j y_j T_j - \sum_{i,j} x_i y_j \omega_{ij}. \tag{25}$$

In this case

$$\frac{\partial E(x,y)}{\partial \omega_{ij}} = -x_i y_j,$$

and

$$\frac{\partial \ln P(x)}{\partial \omega_{ij}} = \sum_y P(y|x) x_i y_j - \sum_{x,y} P(x,y) x_i y_j.$$

Since the mathematical expectation is equal to

$$E(x) = \sum_i x_i P_i,$$

then

$$\frac{\partial \ln P(x)}{\partial \omega_{ij}} = E\left[x_i y_j\right]_{\text{data}} - E\left[x_i y_j\right]_{\text{model}}.$$

The gradients for the thresholds can be obtained similarly:

$$\frac{\partial \ln P(x)}{\partial T_i} = E\left[x_i\right]_{\text{data}} - E\left[x_i\right]_{\text{model}},$$

$$\frac{\partial \ln P(x)}{\partial T_j} = E\left[y_j\right]_{\text{data}} - E\left[y_j\right]_{\text{model}}.$$

From these expressions, it follows that the first term characterizes the run of the neural network basing on the data from the training set; the second term characterizes the run of the neural network basing on the data of the model (the data generated by the neural network) that is the free run independently of the external data.

Since the calculation of the mathematical expectation basing on the RBM neural network is very complex, G. Hinton suggested an approximation of these summands, which he called the contrastive divergence (CD) [3].

The approximation is based on the Gibbs sampling. In this case, the first summands in the expressions for the gradients characterize the distribution of data at the moment of time $t = 0$ and the second summands characterize the restored states or the states generated by the model at the moment of time $t = k$. With that in mind the CD-k procedure can be presented as

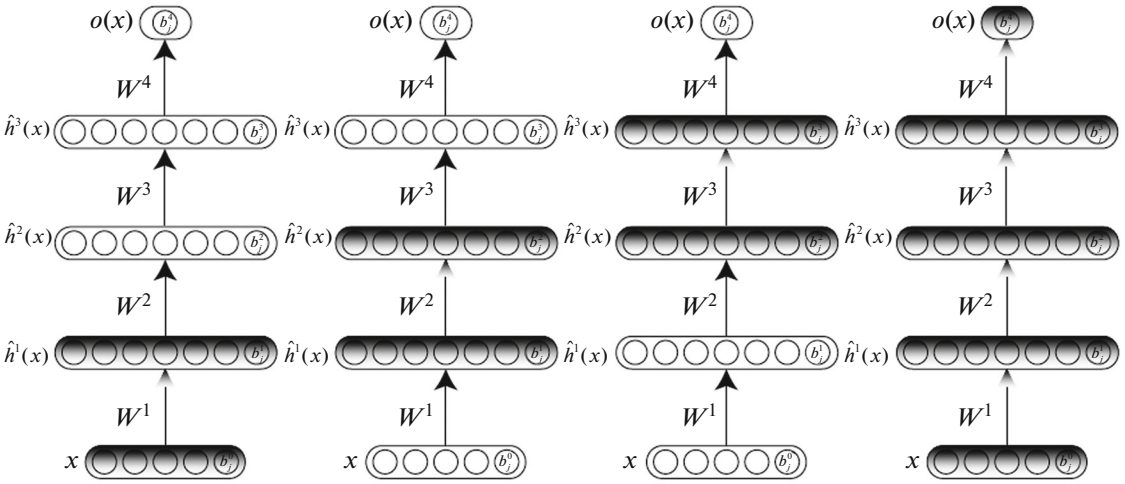
$$x(0) \to y(0) \to x(1) \to y(1) \to \ldots \to x(k) \to y(k). \tag{26}$$

As a result, we can obtain the following learning rules for the RBM neural network. When using CD-1, $k = 1$, and taking into account that in accordance with the method of the gradient descent

$$\omega_{ij}(t+1) = \omega_{ij}(t) + \alpha \frac{\partial Ln P(v)}{\partial \omega_{ij}(t)},$$

for the online training we obtain

$$\begin{aligned} \omega_{ij}(t+1) &= \omega_{ij}(t) + \alpha(x_i(0) y_j(0) - x_i(1) y_j(1)), \\ T_i(t+1) &= T_i(t) + \alpha(x_i(0) - x_i(1)), \\ T_j(t+1) &= T_j(t) + \alpha(y_j(0) - y_j(1)). \end{aligned} \tag{27}$$



(a) First hidden layer pretraining (b) Second hidden layer pretraining (c) Third hidden layer pretraining (d) Fine-tuning of whole network

Fig. 15. Greedy layer-wise algorithm [Journal of Machine Learning Research 1 (2009) 1–40].

The same for the CD-*k* algorithm

$$\begin{aligned} \omega_{ij}(t+1) &= \omega_{ij}(t) + \alpha(x_i(0)y_j(0) - x_i(k)y_j(k)), \\ T_i(t+1) &= T_i(t) + \alpha(x_i(0) - x_i(k)), \\ T_j(t+1) &= T_j(t) + \alpha(y_j(0) - y_j(k)). \end{aligned} \qquad (28)$$

In the case of the batch training and the CD-*k* procedure

$$\omega_{ij}(t+1) = \omega_{ij}(t) + \alpha\sum_{l=1}^{L}(x_i^l(0)y_j^l(0) - x_i^l(k)y_j^l(k)),$$

$$T_j(t+1) = T_j(t) + \alpha\sum_{l=1}^{L}(y_j^l(0) - y_j^l(k)), \qquad (29)$$

$$T_i(t+1) = T_i(t) + \alpha\sum_{l=1}^{L}(x_i^l(0) - x_i^l(k)).$$

From the last expressions, it is evident that the learning rules of the restricted Boltzmann machine minimize the difference between the input data and the data generated by the model. The data generated by the model are obtained by means of the Gibbs sampling [14].

The training of the deep belief neural network is based on the greedy layer-wise algorithm. According with this method, at the beginning the first layer of the neural network is trained as the RBM. For that, the input data are fed to the layer of the visible neuron elements and using the CD-*k* procedure we calculate the states $p(y|x)$ and $p(x|y)$ of the visible and hidden neurons, respectively. When performing this procedure (at most 100 epochs) the weight coefficients and the thresholds values of the RBM neural network change and then they are fixed. Next, we take the second layer of the neural network and generate the new RBM. For it, the input data are the data from the previous layer. The training takes place and the process continues over all layers as it is shown in Fig. 15 [17]. As a result of this unsupervised learning, we can obtain a suitable starting initialization of the fitting parameters of the deep neural network. On the final step the fine-tuning of the parameters of the entire network by means of the backpropagation algorithm or the with the wake-sleep algorithm.

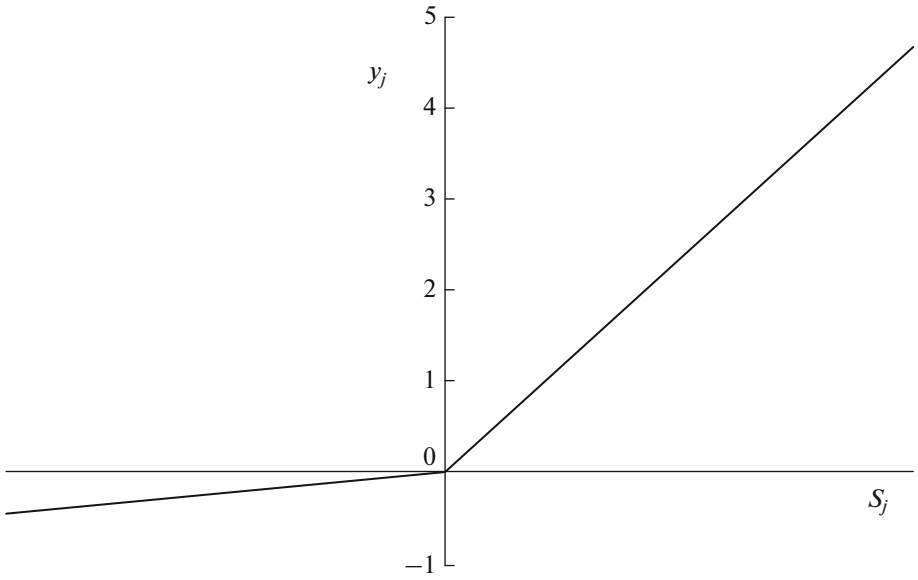


Fig. 16. ReLU activation function.

7. STOCHASTIC GRADIENT METHOD WITH ReLU

In this case, we use the Rectified Linear Unit (ReLU) activation function in all the layers of the deep neural network except the last one. When this activation function is used (Fig. 16), it is not necessary to use pre-training of the layers when learning the deep neural network and the standard backpropagation algorithm can be used. If online or batch learning with a small group of patterns (a mini-batch) is used the gradient descent method is called **the stochastic gradient descent method** (SGD).

When ReLU is used the output value of the neuron element is

$$y_j = F(S_j) = \begin{cases} S_j, & S_j > 0; \\ kS_j, & S_j \le 0, \end{cases} \tag{30}$$

where $k = 0$ or it takes a small value. For example, $k = 0.01$ or $k = 0.001$.

Then

$$\frac{\partial y_j}{\partial S_j} = F'(S_j) = \begin{cases} 1, & S_j > 0; \\ k, & S_j \le 0, \end{cases} \tag{31}$$

Using equations (16) and (17), we can perform the fine-tuning of the synaptic weights of the deep neural network. Why is the backpropagation algorithm effective for the training of the deep neural network when we use the ReLU activation function? As it follows from Eq. (31) in the region of the positive values of the weighted sum, the derivative of the rectifying function is equal to one and this allows us to eliminate the vanishing gradient problem.

8. Up-to-date PARADIGMS OF LEARNING

If the training set is large, that is its size is much larger than the number of the fitting parameters one uses the gradient descent method with the ReLU activation function. If the size of the set is comparable with the number of the fitting parameters, one uses the pre-training of the neural network based on the RBM and the backpropagation algorithm for the fine-tuning of the synaptic weights. As the authors of [19] noted: "For smaller data sets, unsupervised pre-training helps to prevent overfitting, leading to significantly better generalization when the number of labelled examples is small, or in a transfer setting where we have lots of examples for some 'source' tasks but very few for some 'target' tasks. Once deep learning had been rehabilitated, it turned out that the pre-training stage was only needed for small data sets".

CONCLUSIONS

In the present paper, we discussed and analyzed the main paradigms of learning of the multilayer neural networks starting with a single layer perceptron and going on to deep neural networks. We destroyed some myths about the abilities of multilayer neural networks and showed their groundlessness. We reasoned the transition to deep neural networks.

The main models of learning of deep neural networks are discussed. They are the models based on the restricted Boltzmann machine (RBM), the autoassociative approach and the gradient descent method.

REFERENCES

1. Rosenblatt, F., *Principles of Neurodynamics; Perceptrons and the Theory of Brain Mechanisms*, Washington: Spartan Books, 1962, p. 616.
2. Minsky, M. and Papert, S., *Perceptrons: An Introduction to Computational Geometry*, MIT Press, 1969.
3. Hinton, G.E., Osindero, S., and Teh, Y., A fast learning algorithm for deep belief nets, *Neural Computation*, 2006, vol. 18, pp. 1527–1554.
4. Hinton, G., Training products of experts by minimizing contrastive divergence, *Neural Computation*, 2002, vol. 14, pp. 1771–1800.
5. Hinton, G. and Salakhutdinov, R., Reducing the dimensionality of data with neural networks, *Science*, 2006, vol. 313, no. 5786, pp. 504–507.
6. Hinton, G.E., A practical guide to training restricted Boltzmann machines, Tech. Rep. 2010-000, Toronto: Machine Learning Group, University of Toronto, 2010.
7. Widrow, B. and Hoff, M., Adaptive switching circuits, in *1960 IRE WESCON Convention Record*, DUNNO, 1960, pp. 96–104.
8. Golovko, V., *Neural Networks: Training, Organization and Application*, Moscow: IPRZHR, 2001, p. 256 (in Russian).
9. Golovko, V., Technique of Learning Rate Estimation for Efficient Training of MLP, Golovko, V., Savitsky, Y., Laopoulos, T., Sachenko, A., and Grandinetti, L., *Proc. of the IEEE – INNS – ENNS Int. Joint Conf. on Neural Networks IJCNN'2000, Como, Italy, 2000*, Danvers: IEEE Computer Society, 2000, pp. 323–329.
10. Golovko, V., From multilayers perceptrons to deep belief neural networks: Training paradigms and application, *Lections on Neuroinformatics*, Golovko, V.A., Ed., Moscow: NRNU MEPhI, 2015, pp. 47–84 (in Russian).
11. Rumelhart, D., Hinton, G., and Williams, R., Learning representation by backpropagation errors, *Nature*, 1986, no. 323, pp. 533–536.
12. Lippmann, R.P., An introduction to computing with neural nets, *IEEEASSP Mag.*, 1987, vol. 4, no. 2, pp. 4–22.
13. Cybenko, G., Approximations by superpositions of a sigmoidal function, *Math. Control Signals, Syst.*, 1989, vol. 2, pp. 303–314.
14. Bengio, Y., Learning deep architectures for AI, *Foundations Trends Mach. Learning*, 2009, vol. 2, no. 1, pp. 1–127.
15. Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H., Greedy layer-wise training of deep networks, in *Advances in Neural Information Processing Systems*, Schölkopf, B., Platt, J.C., and Hoffman, T., Eds., *MA: MIT Press*, Cambridge, 2007, vol. 11, pp. 153–160.
16. Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. Why does unsupervised pretraining help deep learning?, *J. Mach. Learning Res.*, 2010, vol. 11, pp. 625–660.
17. Larochelle, H., Bengio, Y., Louradour, J., and Lamblin, P., Exploring strategies for training deep neural networks, *J. Mach. Learning Res.*, 2009, vol. 1, pp. 1–40.
18. Glorot, X., Bordes, A., and Bengio, Y., Deep sparse rectifier networks, in *Proc. of the 14th International Conference on Artificial Intelligence and Statistics*, JMLR W&CP, 2011, vol. 15, pp. 315–323.
19. LeCun, Y., Bengio, Y., and Hinton, G., *Deep Learning Nature*, 2015, vol. 521, no. 7553, p. 436.
20. Golovko, V., A learning technique for deep belief neural networks, *Neural Networks and Artificial Intelligence*, vol. 440: *Communication in Computer and Information Science*, Golovko, V., Kroshchanka, A., Rubanau, U., and Jankowski, S., Ed., Springer, 2014, pp. 136–146.
21. Golovko, V., *A New Technique for Restricted Boltzmann Machine Learning*, Kroshchanka, A., Turchenko, V., Jankowski, S., and Treadwell, D., *Proc. of the 8th IEEE International Conference IDAACS-2015, Warsaw 24–26 September 2015*, Warsaw, 2015, pp. 182–186.