DOI https://doi.org/10.15407/csc.2023.01.018
UDC 004.8 + 004.032.26

O.O. HOLTSEV, PhD Student, International Research and Training Centre for Information Technologies and Systems of the NAS and MES of Ukraine, Acad. Glushkov ave., 40, Kiev, 03187, Ukraine,
ORCID: https://orcid.org/0000-0002-1846-6648,
rcwolf@adg.kiev.ua

V.I. GRITSENKO, Corresponding Member of the Ukrainian Academy of Sciences, Director, International Research and Training Centre for Information Technologies and Systems of the NAS and MES of Ukraine, Scopus ID: 7101892671, Acad. Glushkov ave., 40, Kiev, 03187, Ukraine,
ORCID: https://orcid.org/0000-0002-6250-3987,
vig@irtc.org.ua

A SHORT OVERVIEW OF THE MAIN CONCEPTS OF ARTIFICIAL NEURAL NETWORKS

A significant increase in computer performance, the accumulation of a large amount of data necessary for training deep neural networks, the development of training methods for neural networks that allow you to quickly and efficiently train networks consisting of a hundred or more layers, has led to significant progress in training deep neural networks. This allowed deep neural networks to take a leading position among machine learning methods. In this work, neural network paradigms (and their methods of training and functioning) considers, such as Rosenblatt perceptron, multilayer perceptrons, radial basis function network, Kohonen network, Hopfield network, Boltzmann machine, and deep neural networks. As a result of comparative consideration of these paradigms, it can be concluded that they all successfully solve the tasks set before them, but now, deep neural networks are the most effective mechanism for solving intellectual practical tasks.

Keywords*: artificial intelligence, artificial neural networks, machine learning methods, deep neural networks.*

Introduction

The problem of artificial intelligence (AI) is of particular importance today. At the same time, artificial neural networks are one of the main tools for building AI. Therefore, the development of the theory of neural networks directly determines the progress in the creation of AI.

The capabilities of modern computers allow one to perform calculations at a speed that exceeds the capabilities of the human brain by many times. However, tasks that are trivial for humans, not related to calculations, remain extremely difficult for computers. Human abilities for associative storage of information, learning, generalization, processing of information taking into account the context remain unattainable even for modern supercomputers. The purpose of constructing artificial neural networks is to build a computing algorithm that works according to the principles and mechanisms of the human brain. The following properties of neural networks can be attributed to such principles.

1. Neural networks, by analogy with the human and animal brain, are built (composed) of many simple elements that perform relatively simple calculations. These simple elements (neurons) are connected to each other by various connections.

2. Neural networks are able to improve their work (learn and/or adapt) using examples.

3. Solving problems with neural networks does not require the developer to devise a problem solving algorithm and program it. At the same time, the neural network is able to detect hidden patterns in the problem, initially unknown to the developer.

This review briefly describes the main types of artificial neural networks and the basic principles of their functioning.

General Concepts of Artificial Neural Networks. History

Basic Concepts

An artificial neural network (ANN) is a mathematical model, as well as its software and/or hardware implementation, built according to the principle of organization and functioning of biological neural networks networks of nerve cells (neurons) of a living organism. The term ANN arose while modeling the structural organization and processes occurring in the brain.

ANN is a system of connected and interacting simple processors (artificial neurons). Each such neuron, unlike processors of computers, performs a fairly simple operation it calculates the value of a mathematical function, the arguments of which are the values of signals sent to this neuron by other neurons of the network. Being composed in a large complex network, such "neural processors", acting together, are able to perform quite complex tasks, including those that classical computer systems are often unable to perform.

The basic structural units of ANNs are a neuron and a connection between neurons. From a mathematical point of view, ANN is a graph in which neurons are vertices (nodes), and connections between them are edges (see Fig. 1).

In biological neural networks, neurons are connected to each other by means of synapses: the more the number of synapses between two neurons, the *stronger* the connection between them. In ANN, the *value* of the connection between two neurons can be expressed as a coefficient: the larger it is, the stronger the connection. Such a coefficient is called the "weight" of the connection. The signal, going from one neuron to another, is multiplied by the weight of the connection between

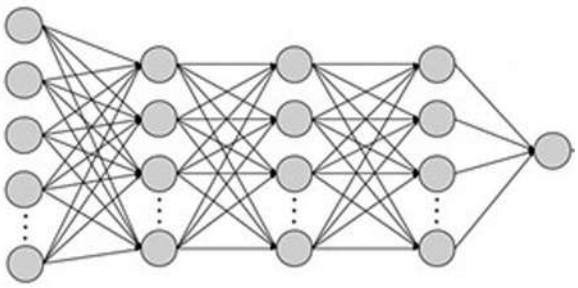

Fig. 1. An example of a neural network

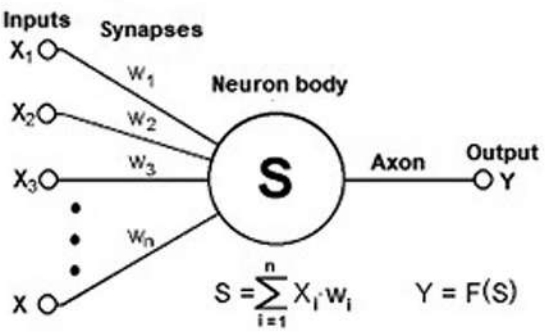

Fig. 2. The general structure of the neural network

them. Since one neuron can receive signals from many other neurons, these signals (in ANNs) are usually summed. Thus, we can say that each neuron receives a "weighted sum" of signals from other neurons of the network connected to it. Fig. 2 illustrates this description:

Rosenblatt's Perceptron

One of the first artificial neural networks ever designed is called “perceptron” (from the Latin perceptio perception) [1]. It uses the model of information perception by the brain and was proposed by Frank Rosenblatt in 1958.

In general, Rosenblatt's perceptron consists of three layers (and, correspondingly, of three types of elements (neurons): sensory, associative and reactive ones:

- Sensory elements (S-elements) are actually the sensors that receive signals from the outside world and transmit them to associative elements. Biological analogues can be, for example, eye retinal receptors.

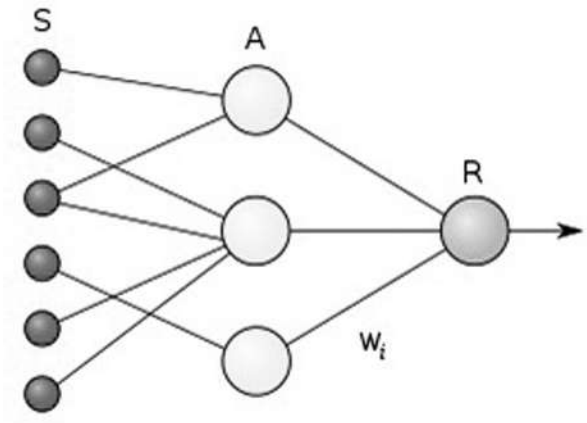

Fig. 3. Rosenblatt's perceptron

- Associative elements (A-elements) are the neurons that process signals and calculate the "association" between the input signals and the required reaction at the output.

- Reactive elements (R-elements) are the neurons that receive signals (weighted sum) from A-elements and are activated when a specific threshold is exceeded.

Thus, topologically, the network consists of 3 layers: input (the layer of sensory elements) S, hidden (the layer of associative elements) A and output (the layer of reactive elements) R. Fig. 3 illustrates this description.

According to the modern terminology, the Rosenblatt perceptron can be classified as an artificial neural network:

- with one hidden layer;
- with a threshold transfer function;
- with direct signal propagation.

In order for the perceptron to correctly perform its functions (pattern recognition; classification), it must be trained. ANN training is basically a process of tuning the network connection weights in such a way that the result (the value of the R-element of the output layer) maximally meets expectations in the context of a specific sample (signals) submitted to the input.

In his works, Rosenblatt describes different approaches to perceptron training, calling them reinforcement systems. In these reinforcement systems, Rosenblatt drew on D. Hebb's ideas about learning in biological neural networks, proposed in 1949. [2]. Mentioned Hebb's ideas can be expressed (in a simplified form) as follows:

- If two neurons on both sides of a synapse (connection between two neurons) are activated simultaneously, then the weight (strength) of this connection is growing.
- If two neurons on either side of a synapse fire asynchronously, the weight (bandwidth) of that synapse decreases.

The classic method of training a perceptron is the *method of error correction*. It is a type of unsupervised learning, in which the weight of the connection does not change as long as the current response of the perceptron remains correct. Once an incorrect reaction emerges, the weight of the connection decreases. (Rosenblatt, however, only permitted A – R connection weight changes, while S – A connections were assigned only once (at the beginning of the learning procedure), a random value, and did not participate in learning process).

Perceptron convergence theorem proves that a perceptron trained using such an algorithm *always* comes to a solution in a finite time.

In 1969, a book by Marvin Minsky and Seymour Papert [3] was published. It showed the fundamental limitations of simple perceptron. This led to decline of interest in ANNs among many researchers. Later, interest in neural networks, and Rosenblatt's ideas, in particular, revived. In 2001 a group of researchers led by Ernst Kussul conducted experiments on training a version of Rosenblatt perceptrons with different numbers of elements in the A layer (up to 512,000) to recognize handwritten digits of the MNIST database [4]. A recognition accuracy of 99.2% was achieved, which is comparable to the best results of the beginning of the 21st century. Later, the same group of researchers significantly improved the recognition accuracy of their models based on Rosenblatt perceptron [5, 6].

Multilayer Perceptrons

After the decline of interest in perceptrons caused by the publication of Minsky [3], their "re-invention" by D. Rumelhart [7] took place in 1986. Although Rosenblatt's perceptrons are actually multi-layered, there was (and perhaps still is) a

misconception that it was Rumelhart who invented and first proposed the use of multi-layer neural networks with one or more hidden layers.

In a collection of articles published in 1986, Rumelhart suggested changing (correcting) connection weights not only of the output layer, but also of the hidden layer. The activation function was proposed to make non-linear. It's usually a sigmoidal function: $S(x) = \frac{1}{1+e^{-x}}$.

As a result, training a multilayer ANN ceases to be a trivial task. Rumelhart, along with other researchers, proposed a method of error backpropagation based on the principle of gradient descent along the error surface [8–10]. This method will be considered separately below.

Differences from Traditional Systems

Computing systems based on neural networks have a number of properties that classical computing systems lack (yet present in nervous systems of animals and humans). The main of them are:

1. The process of creating a neural network a learning process rather than a programming (or design) process.

2. The learning ability of a neural network gives a developer an opportunity to solve tasks with unknown dependencies between input and output data, which allows one to work with incomplete data.

3. Resistance to noise in the input parameters – a neural network can independently determine the parameters that are irrelevant to analysis purpose and filter them out.

4. Adaptation to environmental changes – neural networks can be re-trained in new conditions caused by slight fluctuations in the parameters of the environment.

5. The potential fault tolerance of neural networks is a consequence of the distributed way of storing information in a neural network, due to which only considerable damage to the structure of the network significantly affects its performance.

6. Neural networks make it possible to create efficient software for highly parallelized computers. On the basis of neural networks, it is possible to solve the problem of the efficiency of simultaneous processing of multiple tasks for a wide class of tasks.

Types of Artificial Neural Networks

Classification According to Different Criteria

ANNs can be categorized according to various criteria. Below is a list of the main ones, along with a brief description of them.

By type of input information:

- Analog – use information in the form of real numbers.
- Binary – operate with information presented in a binary form.

By the number of hidden layers:

- Single layer.
- Multi-layered.
- Including deep (more than two hidden layers).

By the type of learning procedure:

- Training with a supervisor – the expected output results of the network are known.
- Learning without a supervisor – the neural network builds its weighted connection structure (connection matrix) basing only on input signals (influences); such networks are called self-organizing networks.
- Reinforcement learning is a system of assigning penalties and incentives from the environment.
- Mixed learning.

By signal transmission time:

- Asynchronous – the neuron activating function depends not only on the connection weights, but also on the time of transmission of the signals through the communication channels.
- Synchronous (does not depend on the transmission time).

By types of connections and activation functions:

- The networks of direct propagation – all connections are directed strictly from input neurons to output neurons.
- With linear/non-linear activation function.
- Recurrent neural networks – the signals from the output neurons or neurons of the hidden layer are partially transmitted back to the inputs of the neurons of the input layer or layers of a lower level.

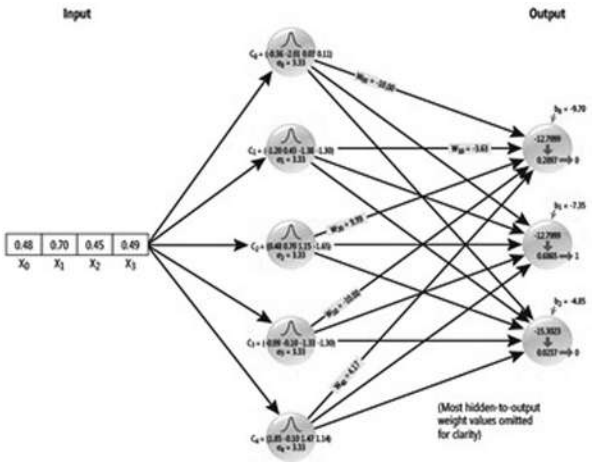

Fig. 4. Radial base network (RBF)

▪ Probabilistic neural networks – activation of a neuron occurs with a probability equal to its output signal (a real number from 0 to 1).

▪ Networks of radial basis functions – as activation functions radial basis functions are used. Also called RBF networks.

▪ Self-organizing maps – competitive neural networks trained without a supervisor, performing visualization tasks and clustering.

The most famous and significant types of neural networks are described below in more details.

Network of Radial Basis Functions

The radial basis network (RBF) [11, 12] is characterized by the following features:

1. The only hidden layer.

2. The weights of the connections going from the receptors of the input layer to the neurons of the hidden layer are one valued.

3. Only neurons of the hidden layer have a nonlinear activation function - radial-basis.

Usually, the Gaussian "bell" is chosen as the activation function:

$$h_i(x) = \exp\left[\frac{\left(\|x - x_i\|\right)^2}{2\sigma_i^2}\right],$$

where x is the input vector, x_i – the reference point (the centre of the radial function) or the i-th image of the training sequence, σ_i – is the "*bell*" width parameter.

As a metric $||x-xi||$ the Euclidean distance is usually used:

$$\|x - x_i\| = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_n - y_b)^2}.$$

Each element of the hidden layer may correspond to an instance of the training sample (in this case, the center of the radial activation function is equal to the corresponding instance – the support vector method), or, if there are too many such instances, the k-means method can be applied (if the training data represents a continuous function, the initial values of the specified cluster centers are placed at the minimum or maximum points), and, accordingly, k elements of the hidden layer with the centers of the radial functions at the mass centers of each of the k selected clusters are set.

The values of the hidden elements' activation radial basis functions widths are chosen so that they are large enough, but the functions do not overlap each other significantly in the input data space. The K nearest neighbors algorithm can be applied:

$$\sigma_i = \sqrt{\frac{1}{K}\sum_{k=1}^{K}\|c_i - c_k\|^2}.$$

(In practice, the value of K is usually taken from the range from 3 to 5).

The elements of the output layer form a linear combination of the outputs of the neurons of the hidden layer:

$$f_i(x) = y_j = \sum_{i=1}^{k} w_{ij} h_i(x).$$

To find the weights of the output layer, the method of gradient descent can be applied (the Widrow-Hoff delta rule, since the activation functions of the output elements are linear). Alternatively, the method of pseudo-inverse matrices can be applied:

The interpolation *matrix H* is found:

$$H = \begin{pmatrix} h_1(\overline{x}_1) & h_2(\overline{x}_1) & \ldots & h_m(\overline{x}_1) \\ h_1(\overline{x}_2) & h_2(\overline{x}_2) & \ldots & h_m(\overline{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ h_1(\overline{x}_p) & h_2(\overline{x}_p) & \ldots & h_m(\overline{x}_p) \end{pmatrix}$$

the inversion of the product of the matrix H on the transposed matrix H is calculated:

$$A^{-1} = \left(H^T H\right)^{-1},$$

weight vectors are calculated according to the formula

$$\underline{W} = A^{-1} H^T \underline{y},$$

where *y* is the target vector of expected values of the output element.

It's also worth mentioning the mixed learning method: first, the weights are calculated using the pseudo-inverse matrix method described above, and then the output layer is trained using the error backpropagation method.

RBF networks are good function approximators, and they are also used for prediction and classification.

Kohonen's Network

Self-organizing feature map (SOFM) by Kohonen [13] – has two layers of neurons (elements): input and output. Such a network is trained without a supervisor and is most often used for clustering and data visualization tasks. SOFM is a method of projecting a multidimensional space into a lower dimensional space.

The input elements of the feature map are only intended to distribute the data of the input vector between the output elements of the network. The output elements are called cluster elements. Each input element is associated with each *cluster element*. It is often convenient to interpret the weight values of a cluster element as coordinates describing the position of the cluster in the input data space [14]. SOFM is a competitive neural network as a result of its work, only one output element is activated according to the WTA (winner-takes-all) rule.

Network training is performed as follows. Initially, the values of the weights of the output elements are set to random values from a limited range. Next, for each training vector, the proximity metric of this vector in the cluster element is calculated. To calculate the metric, the weighted values of the connections going to the cluster element are used. The simplest type of such a metric, for example, can be the Euclidean distance:

$$d_j = \sum_j \left(w_{ij} - x_i\right)^2.$$

The element with the smallest distance wins, and its connection weights are modified to make the element "closer" to the input vector:

$$w_{ij}(n) = w_{ij}(n-1) + \eta(n-1)\left[x_i - w_{ij}(n-1)\right]$$

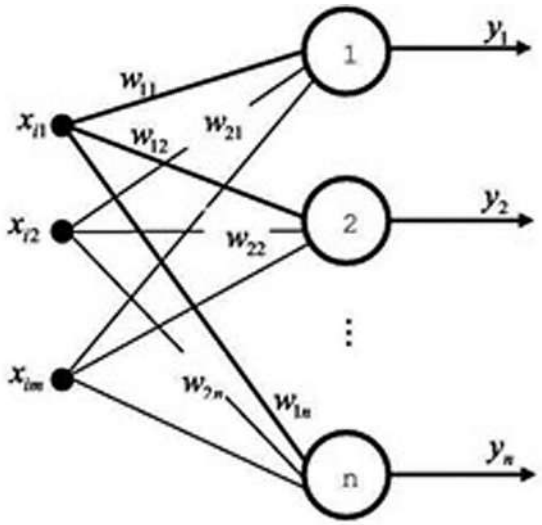

Fig. 5. Kohonen Network

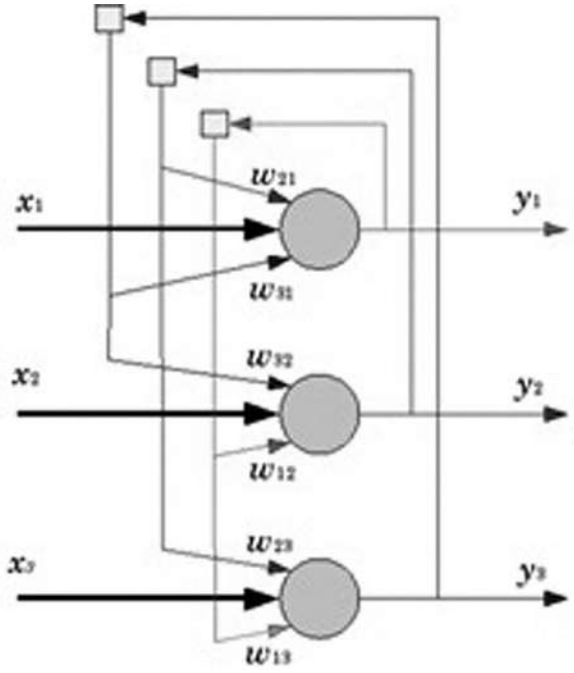

Fig. 6. Hopfield network

here n is the training epoch number, η – the training rate function – a special training coefficient of the network that is gradually reduced each new epoch.

Generally, not only winner element's weights are modified, but, also, the weights of other ele-

ments close to the input vector. The weights of these elements can be changed provided that they are close enough to the winning element (that is, some "radius" is set, which gradually decreases as the network is trained). Also, the weights of these elements may change depending on the degree of proximity to the winning element.

Hopfield's Network

Hopfield's neural network is an autoassociative network that behaves like a memory, which can restore a stored image, having received a distorted version of any stored sample as an input [15]. This recurrent network consists of one layer of elements connected to each other (except for themselves). Each element of the Hopfield network receives as an input, (and also outputs) a value from the set {-1, 1} (the same way as for the neurons with threshold activation function).

Network training is performed in one step. The following formula is used to calculate the matrix of weighted values of connections:

$$W = \sum_i X_i^T X_i.$$

After calculating the matrix, its diagonal is zeroed (because the elements have no connection with themselves).

The network functions as follows. The output signals of the elements are set equal to the elements of the input vector X. One of the elements is selected in a pseudo-random (uniform) manner and the weighted sum of the signals from all the remaining elements is calculated. If the value of the sum is greater than zero, the state of the element is set to 1. If the value of the sum is less than zero, the state becomes equal to 0. If the value of the sum is 0, the state remains unchanged. The operation is repeated until the network reaches a stable state. In a number of works, there are estimates of the number of images that can be stored in such a network. One such general estimate is the following:

$$p_{\max} = \frac{N}{2 \ln N},$$

where N – the number of network elements.

In [15], Hopfield proved that the network should converge to a stable set of activity values having considered the *energy* function of the system:

$$E = \frac{-1}{2} \sum_j \sum_i x_j x_i w_{ij}.$$

If element j changes its state by the value Δ_{sj}, then the change of energy will be equal to

$$\Delta E = -\Delta s_j \sum_i s_j w_{ij} \Delta.$$

Boltzmann Machine

Boltzmann machine is a stochastic recurrent neural network suggested by Geoffrey Hinton and Terry Seynovsky in 1985 [16]. Boltzmann machine may be considered as a stochastic generative version of Hopfield's network [15]. This kind of neural networks utilizes the concept of so-called "simulated annealing" method for the network state refresh process.

The reason for using this method is the disadvantage of the Hopfield's network, which is a high chance of the network to converge to a local minimum instead of the desired global one.

The algorithm of Boltzmann machine functioning is based on the simulation of the physical process that occurs during the crystallization of a substance, such as the annealing of metals. It is assumed that the atoms of the substance are already almost built into a crystal lattice, but transitions of individual atoms from one cell to another are still possible. The higher the temperature, the higher the activity of atoms. The temperature is gradually reduced, which leads to the fact that the probability of transitions to states with higher energy decreases. A stable crystal lattice corresponds to the minimum energy of atoms, so the atom either moves to a state with a lower energy level, or remains unchanged. (This algorithm is also called the algorithm of *N*. Metropolis (after its author [17]). The main procedure of this algorithm is the random selection of a part of the system to change (in case of applying this algorithm to find the minimum of the real function of a binary vector X, this will be a change of a bit of the vector X). The change is always accepted if the global energy of the system decreases, and if an increase in energy is observed, then the changes are accepted according to the probability:

$$p = e^{\frac{-\Delta E}{T}},$$

where T stands for temperature.

After every M changes of the system state (M is a predefined number of cycles), that is, changes of any element state from "on" to "off" or vice versa, the temperature value T is forcibly reduced to a specified minimum $T_0 (T_0 \geq 1)$.

With respect to the Boltzmann machine, the value of the probability that an element should be activated is calculated using the logistic function:

$$p_{i=on} = \frac{1}{1 + e^{\frac{-\Delta E}{T}}}.$$

Similarly to the Hopfield network, the energy of the Boltzmann machine is defined as:

$$E = -\sum_{i<j} x_i w_{ij} x_j - \sum_{i<j} \theta_i x_i,$$

where θ_i is the activation threshold of the *i-th* element.

The elements of the Boltzmann machine are divided into two subsets: visible and hidden. The learning process of such a network consists of two phases: the fixation phase, during which the visible elements are “fixed” with the values of the training vectors, and the free execution phase, during which only the input elements are fixed (in case the visible elements are divided into inputs and outputs), or the states of the elements are not fixed at all. During the first phase for each sample the visible elements of the network are switched to the states corresponding to the binary values of the input vector of the sample, and then the network is allowed to stabilize by gradually reducing the “temperature” value. (At the same time, a hidden element is randomly selected; the probability of its switching is calculated; it gets either switched or not; next random hidden item is selected, etc.).

After the stabilization of the network, the state of all elements is fixed and the transition to the next sample is performed. During the second phase of the process, only the state of the input elements is fixed (if the visible ones are divided into inputs and outputs), or nothing is fixed, and the system is allowed to stabilize a certain (large) number of times, each time gradually lowering the "temperature", as well as collecting statistics of element states when the network gets stable. Next, having the statistics

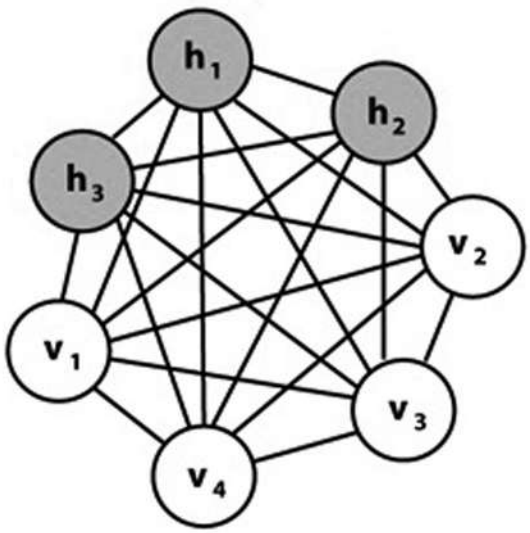

Fig. 7. Boltzmann machine with hidden and visible elements

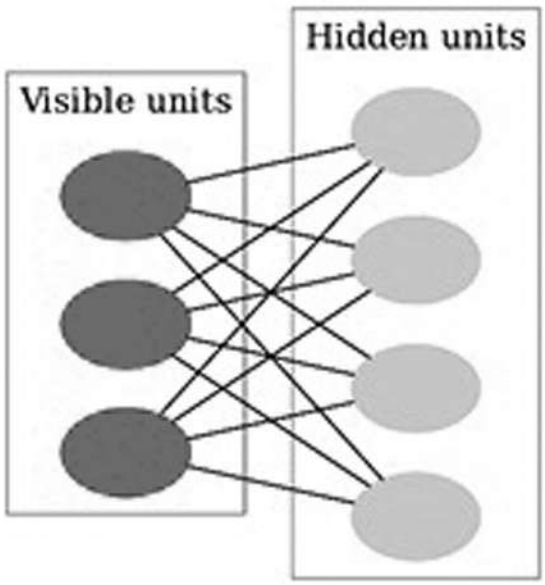

Fig. 8. Restricted Boltzmann Machine

of the first and second phases, for each phase the probability of each pair of (*i, j*) elements to be simultaneously “on” is calculated. Finally, for each connection of two elements, the change of weight is calculated as follows:

$$w = \eta\left(p_{ij}^{+} - p_{ij}^{-}\right),$$

where p_{ij}^{+} is the probability of elements (*i, j*) being simultaneously in the switched-on state in the first

phase and p_{ij}^{-} – in the second phase, where η – network training rate.

After modifying the weights, this set of operations is repeated until the network converges (that is, until the weights stop changing). Such an algorithm is extremely inefficient and can take an extremely long time, as a consequence the fully-connected Boltzmann machine is rarely used in practice. However, its "restricted" version, which has a relatively fast learning algorithm and has proven itself well in the area of deep neural networks, is much more popular.

A Restricted Boltzmann machine (RBM), is a type of generative stochastic neural network that determines a probability distribution over input data samples.

RBM got its name as a modification of the fully-connected Boltzmann machine. In RBM the neurons are divided into visible and hidden, and connections are allowed only between neurons of different types. Also, unlike the "unrestricted" version, the method of simulated annealing is not used to train this type of network.

Deep Neural Networks

In simple terms, deep neural networks (DNNs) are neural networks with a large number of hidden layers, as well as a large number of neurons (elements) in each layer. ANNs, in general, are neural networks with direct connections, in which data is transmitted from the input layer to the output layer without feedback. One of the reasons of successful usage of deep neural networks is that such a network can automatically extract meaningful features necessary for solving problems from the input data. In the alternative machine learning algorithms, these features have to be extracted manually by humans. There is a special direction of research feature engineering. However, when processing large volumes of data, the neural network copes with feature selection much better than a human [18].

In the 20th century, deep neural networks were rarely considered by researchers due to the problematic nature of training such networks: firstly, the performance level of computational technologies did not allow high-quality experiments of training large neural networks in those years, secondly, the training methods had their drawbacks and limitations that are discussed in more detail in section “Deep Learning. Deep Belief Network Pre-Training”.

In 1980 Kunihiko Fukushima suggested a neural network architecture, which he called neocognitron [19]. The architecture of the network claimed to be analogous to complex and simple cells in the cat's visual cortex [20]. Simple cells activate in response to simple visual stimulus, such as an object boundaries orientation. Complex cells are less dependent on the spatial structure of signals and focus on its more general features. In neocognitron, convolutional layers correspond to simple cells, and subsampling layers correspond to complex cells. Despite the fact that neocognitron is a deep neural network, deep learning is not used in it.

In 1987 Dana Ballard proposed an approach to training neural networks without a supervisor based on an autoencoder [21]. This approach is considered in more detail below.

In 1990 the error backpropagation algorithm was applied to train a neural network for handwritten digit recognition [22].

In the late 1980s, it became clear that one error backpropagation algorithm would not be sufficient for effective deep learning. An explanation for this was presented in 1991 in [23, 24] this is the vanishing gradient problem.

As a result, the vanishing gradient problem was proposed to be solved in various ways: evolutionary methods, Hessian-free optimization, very deep learning, long-short-term memory network, RelU function, deep belief networks, autoencoders. The last two methods are considered below.

Training Methods

Main Ideas

The general task of neural network training is to select the values of the connection weights in such a way that the network works in the best way. As mentioned earlier, there are two different types of neural network training: supervised and unsupervised. The unsupervised learning method was discussed above in the context of Hopfield networks, partly Boltzmann machines, and partly RBF networks.

When training a network with a supervisor, for each input vector of the training sample, the correct output vector is known in advance. The essence of the learning process is correcting the connection weights of the network neurons, if the network produces a result different from the desired one.

For a forward-propagational neural network with a linear activation function, it is easy to compare the output of the network with the desired output and calculate the errors for each neuron:

$$\delta_j = t_j - o_j,$$

where t_j is the desired value, a o_j – actual.

The change of the neuron's connection weight, in this case, is calculated by the Widrow-Hoff rule, called the delta rule

$$\Delta w_{ij} = \eta \delta_j x_i,$$

where η is the learning rate of the network (constant), and x_i – the signal sent to the i-th neuron. The errors of the neurons of the hidden layers are also calculated quite simply.

The problem arises when the activation function of the neurons of the network is non-linear. Let it be a sigmoidal function

$$\sigma = \frac{1}{1+e^{-x}}.$$

In this case, the task is to calculate the change in the weight of a neuron in proportion to the "contribution" of this neuron to the resulting error of the network. This can be achieved by applying the method of the errors backpropagation.

The Method of Error Backpropagation

Let's make some definitions (Table 1).

Let $E(\ldots, w_{ij}, \ldots)$ – the error function of the network that depends on the connection weight of the i-th and j-th neuron. Our task is to minimize the error. To do this, it is suggested to use the gradient descent method, according to which the argument must be changed in the direction of the antigradient of the function:

$$w_{ij}(n) = w_{ij}(n-1) - \eta \frac{\partial E}{\partial w_{ij}}. \quad (1)$$

So:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}.$$

The derivative of the network error from the weight of a specific connection can be decomposed according to the chain rule:

Table 1. **The some definitions**

Definitions	Explanation
η	rate (speed) of network training
x_i	input signal to neuron i
o_j	output signal value of neuron j
t_j	desired signal value of neuron j
$net_j = \sum_i o_i w_{ij}$	total input signal to neuron j from all neurons connected to it (weighted sum)
$E = \frac{1}{2}\sum_j \left(t_j - o_j\right)^2$	The total error of the network is the error of all neurons of the output layer

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_{ij}}{\partial w_{ij}}.$$

We apply the chain rule again

$$\frac{\partial o_{ij}}{\partial w_{ij}} = \frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}.$$

As a result, we have

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}. \quad (2)$$

Let us first note

$$\frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j} = \frac{\partial E}{\partial net_j} = \delta_j. \quad (3)$$

Note that

$$\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial\left(\sum_i o_i w_{ij}\right)}{\partial w_{ij}} = o_i.$$

We also note that $\frac{\partial o_j}{\partial net_j}$ is the derivative of the output with respect to the input, and it corresponds to the derivative of the activation function of the neuron. Let's designate it $f'(net_j)$. Then formula (2) takes the next form:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \cdot f'(net_j) \cdot o_i. \quad (4)$$

Let's proceed to the calculation of the derivative $\frac{\partial E}{\partial o_j}$. To do this, consider two cases: neuron j is in the output layer, and neuron j is not in the output layer.

Let's start with the first case:

$$\frac{\partial E}{\partial o_j} = \frac{\partial\left(\frac{1}{2}(\ldots + (t_j^2 - 2t_j o_j + o_j^2)\right)}{\partial o_j} =$$

$$= \frac{1}{2}(-2t_j + 2o_j) = -(t_j - o_j) = -\delta_j .$$

Now, for the second case, let's present E as a complex function that depends on all the weighted sums of signals coming from the neurons of this layer (not the last) to the next one – $E\left(net_u(o_j), net_v(o_j), \ldots, net_z(o_j)\right)$.

By the rule of finding the derivative of a complex function of multiple arguments

$$z = f\left(x(t), y(t)\right)$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial x}\cdot\frac{\partial x}{\partial t} + \frac{\partial z}{\partial y}\cdot\frac{\partial y}{\partial t}$$

$$\frac{\partial E}{\partial o_j} = \frac{\partial E\left(net_u(o_j), net_v(o_j), \ldots, net_z(o_j)\right)}{\partial o_j} =$$

$$\frac{\partial E}{\partial net_u}\cdot\frac{\partial net_u(o_j)}{\partial o_j} + \frac{\partial E}{\partial net_v}\cdot\frac{\partial net_v(o_j)}{\partial o_j} +$$

$$+\ldots + \frac{\partial E}{\partial net_z}\cdot\frac{\partial net_z(o_j)}{\partial o_j}.$$

Since the weighted sum of the signals of all neurons of this layer (L), applied to the input of the neuron of the next layer k, is determined by the equation $net_k = \sum_{\ell\in L} o_\ell w_{\ell k} = o_{\ell_1} w_{jk} + \ldots o_j w_{jk} + \ldots o_{\ell_1} w_{jk}$, then all are derivatives $\frac{\partial net_k(o_j)}{\partial o_j}$ will be equal wjk.

So, $\frac{\partial E}{\partial o_j} = \sum_k \frac{\partial E}{\partial net_k}\cdot w_{jk}$.

Using formula (2), rewrite the last expression as t_j.

As a result, we formulate the expression (3) for both cases (the last and not the last layer) (See Table 2.)

Value δ is first calculated for the neurons of the output layer, then the previous layer, etc. Next, the weights are modified according to formula (1). In practice, each weight is also additionally modified by a small fraction of the weight change of the previous step. This helps reduce the chance of weight change fluctuation.

Autoencoders

An autoencoder is a special architecture of artificial neural networks, which allows to apply unsupervised training [25] when using the method of error backpropagation. The simplest autoencoder architecture is a forward-propagational network, without feedback, most similar to a perceptron and containing an input layer, an intermediate layer, and an output layer. Unlike a perceptron, the output layer of an autoencoder must contain as many neurons as the input layer.

The main principle of operation and training of autoencoder networks is getting the network to produce output layer result as close to the input as possible. To make sure the solution does not turn out to be trivial, a restriction is imposed on the intermediate layer of the autoencoder: the intermediate layer must be smaller in size than the input and output layers. This limitation forces the neural network to look for generalizations and correlations in the input data, to "compress" it. Thus, the neural network automatically learns to extract common features from the input data, which are encoded as the values of the neural network connection weights. Hence, while training the network on a set of different input images, the neural network can independently learn to recognize lines and stripes at different angles.

Most often, autoencoders use cascading to train deep (multilayer) networks. Autoencoders are used for pre-training a deep network without a supervisor. To achieve this, the layers are trained in turns, one by one, starting from the first ones. An addi-

Table 2. **Last layer and not-last layer connection weight modifiers**

i is in the previous layer, *j* is in the last layer	*i* is in the previous layer, *j* is not in the last layer, *k* is in the next layer
$\frac{\partial E}{\partial w_{ij}} = \delta_j o_i$ where $\delta_j = -(t_j - o_j)\cdot f'(net_j)$	$\frac{\partial E}{\partial w_{ij}} = \delta_j o_i$ where $\delta_j = \sum_k \delta_k w_{jk}\cdot f'(net_j)$

tional output layer is connected to each new untrained layer prior to training. It complements the networks according to autoencoder architecture.

Next, a set of training samples is "fed" to the input of the network. The weights of the untrained layer and the additional layer of the autoencoder are tuned using the error backpropagation method. The encoder layer is then disabled and a new one corresponding to the next untrained network layer is created. The same set of training data is fed to the input of the network again, but this time the layers of the network that have already been trained remain unchanged and work as an input for the next layer that is being trained. Likewise, training continues for all layers of the network, except for the last ones. The final layers of the network are usually trained without using an autoencoder, instead, using the same error backpropagation method and involving labeled data (supervised training).

Deep Learning. Deep Belief Network Pre-training

As already mentioned in section "Deep Neural Networks", at the end of the 20th century it became clear that the method of error backpropagation was not effective enough when the training objects were neural networks with a nonlinear activation function and the number of hidden layers of elements exceeded two (such networks, in fact, are considered "deep").

With the development of computing technology and it's performance growth, it became possible to build large-scale neural networks. In practice, it turned out that one of two problems would often arise during training of deep neural networks by the method of error backpropagation: the problems of a fading (vanishing) or exploding gradient. The essence of these problems is that for layers distant from the output layer by 2 or more, the value of the gradient becomes insignificantly small (which reduces the effect of such learning to zero), or vice versa - extremely large, which leads to incorrect learning, not to mention the problem of processing such large numbers (values of weights) using the processors of modern computers.

A neural network's susceptibility to exploding or vanishing gradient problems depends largely on the

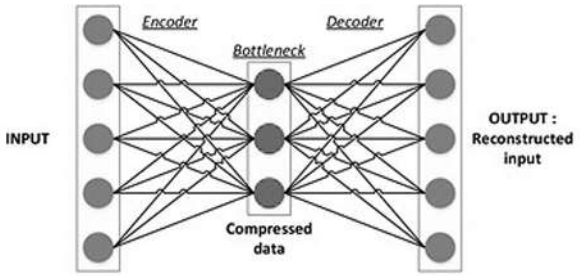

Fig. 9. Autoencoder

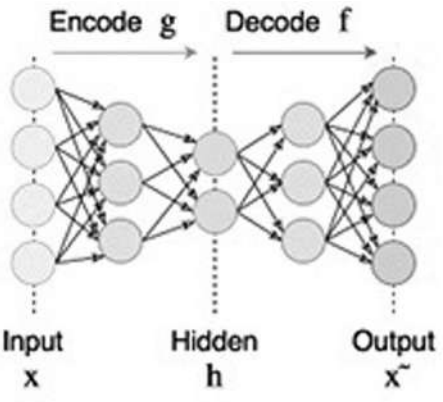

Fig. 10. Cascade of autoencoders

properties of the activation functions used. Therefore, their correct selection is important to prevent the described problems. Many solutions have been proposed to solve the problems of gradient explosion and vanish. They all have their drawbacks. The simplest method of combating these problems is the selection of the activation function (instead of the classical sigmoidal one). The most famous alternative activation functions are:

- ReLU: $h(x) = \max(0, x)$. The function is easy to calculate and has a derivative equal to either 1 or 0. It is also believed that this function is used in biological neural networks. At the same time, the function doesn't saturate in positive region, which makes the gradient more sensitive to distant layers. The disadvantage of the function is the absence of a derivative at zero, which can be eliminated by additionally defining the derivative at zero on the left or right. There are also modifications of ReLU.
- Softplus: $h(x) = \ln(1 + e^x)$. The smooth, everywhere differentiable analogue of the ReLU func-

tion therefore inherits all its advantages. However, this function is more difficult to calculate.

One of the alternative methods of combating the shortcomings of the error backpropagation method is pre-training the layers of the network separately and, then, assambling them into a single deep network. At the same time, it is usually necessary to supplement the networks with two or three layers of neurons with randomly generated connections and re-train these two or three layers by the method of error backpropagation. For "pre-training" you can use the autoencoders described in section “Autoencoders”. The essence of such pre-training is to build a cascade network from layers of neurons that are trained to extract features from input data. Each new layer implies the extraction of more and more "deep" features from the features, presented to it by the previous layer. A network consisting of such layers is usually called "a deep belief network".

One of the popular methods for pre-training a deep neural network is pre-training a network based on restricted Boltzmann machines. In this case, the deep belief network consists of the layers of hidden elements taken from restricted Boltzmann machines.

To build a deep belief network, it is necessary to train a series of restricted Boltzmann machines in cascade: the first network is trained using the original input data, and all subsequent ones are trained using the signals of the hidden elements of the previous (already trained) layers. The most optimal learning algorithm at the moment is the contrastive divergence algorithm proposed by D. Hinton [26]. The algorithm uses the Gibbs sampling method to organize the gradient descent procedure. The connection weights of the trained RBMs are assigned to the corresponding layers of the deep belief network. By adding two or three additional layers of neurons with random values of connection weights to the deep belief network, and training these layers by the method of error backpropagation, we get a complete deep neural network.

Conclusion

In this work, the main paradigms of neural networks (and methods of their training) are considered. As a result of comparative consideration of these paradigms, it can be concluded that all of them successfully solve the tasks set before them, but today deep neural networks are the most effective mechanism for solving intellectual practical tasks.

The growing popularity of deep neural networks in recent years can be explained by three factors.

Firstly, there has been a significant increase in computer performance, including GPU (Graphics Processing Unit) calculation accelerators, which makes it possible to train deep neural networks much faster and with higher accuracy.

Secondly, a large amount of data has been accumulated, which is a requirement for the deep neural networks training.

Thirdly, such neural network training methods have been developed that allow fast and high-quality training of networks consisting of a hundred or more layers [27], which was previously impossible due to the problem of vanishing gradient and overtraining.

The combination of these three factors has led to significant progress in deep neural networks training and their practical use, which allows deep neural networks to take a leading position among machine learning methods.

Neural networks have been successfully applied in a wide variety of areas - business, medicine, technology, geology, physics, etc.

The contribution value of neural network methods in medicine is getting more and more significant. One of the areas where neural networks (especially deep neural networks) perform best – is the classification task area, which includes the diagnosis of diseases, forecasting the dynamics of pathologies development, etc.

Recently, neural networks and, especially, deep neural networks have decidedly become an integral part of our lives and are widely used for solving different tasks and are actively used where conventional algorithmic solutions are ineffective or even impossible. Among the tasks, the solution of which can be attributed to the competence of neural networks, are the following: speech and text recognition, text analysis, semantic search, contextual advertising on the Internet, spam filtering, verification of suspicious bank card transactions,

security systems and video surveillance, expert systems, decision support systems and, even, the prediction of stock market prices. The emergence of a new promising direction in neural network approach, namely, their combination with hybrid intelligence is expected.

REFERENCES

1. *Rosenblatt, F.* (1962). Principles of Neurodynamics. Perceptrons and Theory of Brain Mechanisms. Washington, DC: Spartan Books.
2. *Hebb, D.O.* (1949). The Organization of Behavior. New York, USA: John Wiley & Sons Inc.
3. *Минский М., Пейперт С.* (1971). Персептроны, Мир, 261 с.
4. *Kussul, E, Baidyk, T., Kasatkina, L., Lukovich, V.* (2001). "Rosenblatt perceptrons for handwritten digit recognition". IJCNN'01. Proceedings of the International Joint Conference on Neural Networks., Vol. 2, pp. 1516-1520. doi: 10.1109/IJCNN.2001.939589.
5. *Kussul, E., Baidyk, T.* (2004). "Improved method of handwritten digit recognition tested on MNIST database". Image and Vision Computing, 22, pp. 971–981.
6. *Kussul E., Baidyk T.* (2006). "LIRA neural classifier for handwritten digit recognition and visual controlled microassembly". Neurocomputing, 69 (16–18), pp. 2227–2235.
7. Parallel Distributed Processing: Explorations in the Microstructures of Cognition (1986). Ed. by Rumelhart D. E. and McClelland J.L. Cambridge, MA: MIT Press.
8. *Galushkin, A.I.* (1974). Sintez Mnogosloynykh Sistem Raspoznavaniya Obrazov. M.: "Energiya", 1974 p. [Галушкин А.И. (1974). Синтез Многослойных Систем Распознавания Образов. М.: «Энергия», 1974 с.] (In Russian).
[9] *Werbos, P.J.* (1974). Beyond regression: New tools for prediction and analysis in the behavioral sciences. Ph.D. thesis, Harvard University, Cambridge, MA.
10. *Rumelhart, D.E., Hinton, G.E., Williams, R.J.* (1986). "Learning internal representations by error propagation". In: Parallel Distributed Processing, Vol. 1, Cambridge, MA, MIT Press. pp. 318 362.
11. *Broomhead, D.S., Lowe, D.* (1988). "Multivariable functional interpolation and adaptive networks". Complex Systems. 2, pp. 321–355.
12. *Schwenker, F., Kestler, H.A., Palm, G.* (2001). "Three learning phases for radial-basis-function networks". Neural Networks. 14 (4–5), pp, 439–458. doi:10.1016/s0893-6080(01)00027-2.
13. *Kohonen, T.* (2001). Self-Organizing Maps (Third Extended Edition), New York, 501 p. ISBN 3-540-67921-9.
14. *Callan, R.* (1999). The Essence of Neural Networks. Prentice Hall Europe, London. ISBN 13: 9780139087325.
15. *Hopfield, J.* (1984). "Neurons with graded response have collective computational properties like those of two-state neurons". Proceedings of the National Academy of Sciences of the United States of America. 81. pp. 3088-3092. DOI: 10.1073/pnas.81.10.3088.
16. *Ackley, D.H., Hinton, G.E., Sejnowski, T.J.* (1985). "A learning algorithm for Boltzmann machines". Cognitive Science. 9 (1), pp. 147 169.
17. *Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.* (1953). "Equations of state calculations by fast computing machines". Journal Chemical Physics, 21, pp. 1087 1091. DOI: doi.org/10.1063/1.1699114.
18] *Sozykin, A.V.* (2017). "An overview of methods for deep learning in neural networks". Vestnik Yuzhno-Ural'skogo Gosudarstvennogo Universiteta. Seriya" Vychislitelnaya Matematika i Informatika". 6 (3), pp. 28 59.
19. *Fukushima, K.* (1980). "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". Biological Cybernetics, 36, pp. 193 202.
20. *Wiesel, D.H., Hubel, T.N.* (1959). "Receptive fields of single neurones in the cat's striate cortex". The Journal of Physiology, 148 (3), pp. 574–591. DOI: 10.1113/jphysiol.1959.sp006308.
21. *Ballard, D.H.* (1987). "Modular learning in neural networks". Proceedings of the Sixth National Conference on Artificial Intelligence. Seattle, Washington, USA, July 13–17, 1987. Vol. 1, pp. 279–284.
22. *Le Cun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.* (1990). "Handwritten digit recognition with a back-propagation network". Advances in Neural Information Processing Systems 2. Morgan Kaufmann, pp. 396–404.
23. *Hochreiter, S.* (1991). Untersuchungen zu dynamischen neuronalen netzen. Diploma thesis. Institut fur Informatik, Lehrstuhl Prof. Brauer. Technische Universitat Munchen.
24. *Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J.* (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In S. C. Kremer & J. F. Kolen (ed.), A Field Guide to Dynamical Recurrent Neural Networks. Wiley-IEEE Press, pp. 237–243. DOI: 10.1109/9780470544037.ch14.

25. *Khurshudov, A.A.* (2014). "Obucheniye mnogosloynogo razrezhennogo avtokodirovshchika na izobrazheniyakh bol'shogo masshtaba". Vestnik komp'yuternykh i informatsionnykh tekhnologiy, 2, pp. 27-30. [Хуршудов А.А. (2014). Обучение многослойного разреженного автокодировщика на изображениях большого масштаба. Вестник компьютерных и информационных технологий, 2, С. 27 30]. DOI: 10.14489/vkit.2014.02 (In Russian).
26. *Hinton, G.E.* (2002). "Training products of experts by minimizing contrastive divergence". Neural Computation. 14 (8), pp. 1771 1800. DOI: 10.1162/089976602760128018.
27. *He, K., Zhang, X., Ren, S., et al.* (2016). "Deep residual learning for image recognition". IEEE Conference on Computer Vision and Pattern Recognition (Las Vegas, NV, USA, 27–30 June 2016), pp. 770–778. DOI: 10.1109/CVPR.2016.90.

Received 21.10.2022

О.О. Гольцев, аспірант, Міжнародний науково-навчальний центр інформаційних технологій і систем НАН та МОН України, ORCID: https://orcid.org/0000-0002-1846-6648,
03187, м. Київ, просп. Акад. Глушкова,40, Україна,
rcwolf@adg.kiev.ua
В.І. Грищенко, чл.-кор. НАН України, почесний директор, Міжнародний науково-навчальний центр інформаційних технологій і систем НАН та МОН України, Scopus ID: 7101892671,
ORCID: https://orcid.org/0000-0002-6250-3987,
03187, м. Київ, просп. Акад. Глушкова,40, Україна,
vig@irtc.org.ua

КОРОТКИЙ ОГЛЯД ОСНОВНИХ КОНЦЕПЦІЙ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ

Вступ. У роботі розглянуто такі парадигми нейронних мереж (і їх методи навчання та функціонування), як персептрон Розенблатта, багатошарові персептрони, мережа радіально-базових функцій, мережа Кохонена, мережа Хопфілда, машина Больцмана та глибокі нейронні мережі.

Мета. В результаті розгляду цих парадигм можна зробити висновок, що всі вони успішно вирішують поставлені перед ними завдання, але на сьогодні глибокі нейронні мережі є найефективнішим механізмом для вирішення інтелектуальних практичних завдань.

Результати. Зростання популярності глибоких нейронних мереж, що відбувається останніми роками, можна пояснити трьома чинниками. По-перше, відбулося суттєве збільшення продуктивності комп'ютерів, у тому числі прискорювачів обчислень *GPU* (*Graphics Processing Unit*), що дало змогу навчати глибокі нейронні мережі значно швидше і з вищою точністю.

По-друге, було накопичено великий обсяг даних, необхідний для навчання глибоких нейронних мереж.

По-третє, було розроблено методи навчання нейронних мереж, що дають змогу швидко та якісно навчати мережі, які складаються зі ста і більше шарів, що раніше було неможливо через проблему зникаючого градієнта та перенавчання.

Висновки. Поєднання цих трьох чинників спричинило суттєвий прогрес у навчанні глибоких нейронних мереж та практичного використання їх, що дало глибоким нейронним мережам змогу посісти позицію лідера серед методів машинного навчання.

Ключові слова: *штучний інтелект, штучні нейронні мережі, методи машинного навчання, глибокі нейронні мережі.*