View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Portal de Revistas de Nicaragua

Nexo Revista Científica

Vol. 33, No. 02, pp. 623-637/Diciembre 2020

ISSN-E 1995-9516
Universidad Nacional de Ingeniería
COPYRIGHT © (UNI). TODOS LOS DERECHOS RESERVADOS
http://revistas.uni.edu.ni/index.php/Nexo
https://doi.org/10.5377/nexo.v33i02.10798

Neural Network for Handwriting Recognition

Red neuronal para el reconocimiento de escritura a mano

Mikhail M. Butaev[1], Mikhail Yu. Babich[1], Igor I. Salnikov[2], Alexey I. Martyshkin[3], Dmitry V. Pashchenko[4], Dmitry A. Trokoz[5]

[1] Dr. of Technical Sciences, Professor, Scientific Secretary, JSC Research and Production Enterprise «Rubin», 440000, Russia, Penza, Baydukova St, 2,
[2] Doctor of Technical Sciences, Professor, Head of Department of Computational Machines and Systems, Penza State Technological University, (440039, Russia, Penza, 1/11 Baydukova proyezd/Gagarina ul., 1/11
[3] Candidate of Technical Sciences, Associate Professor, Department of Computational Machines and Systems, Penza State Technological University, 440039, Russia, Penza, 1/11 Baydukova proyezd/Gagarina ul., 1/11
[4] Dr. of Technical Sciences, Professor, Rector, Penza State Technological University (440039, Russia, Penza, 1/11 Baydukova proyezd/Gagarina ul., 1/11
[5] Candidate of Technical Sciences, Associate Professor, Vice-Rector for Research, Penza State Technological University (440039, Russia, Penza, 1/11 Baydukova proyezd/Gagarina ul., 1/11

Corresponding autor email: nts@npp-rubin.ru

(recibido/received: 16-July-2020; aceptado/accepted: 19-September-2020)

ABSTRACT

Today, in the digital age, the problem of pattern recognition is very relevant. In particular, the task of text recognition is important in banking, for the automatic reading of documents and their control; in video control systems, for example, to identify the license plate of a car that violated traffic rules; in security systems, for example, to check banknotes at an ATM and in many other areas. A large number of methods are known for solving the problem of pattern recognition, but the main advantage of neural networks over other methods is their learning ability. It is this feature that makes neural networks attractive to study. The article proposes a basic neural network model. The main algorithms are considered and a programming model is implemented in the Python programming language. In the course of research, the following shortcomings of the basic model were revealed: low learning rate (the number of correctly recognized digits in the first epochs of learning); retraining - the network has not learned to generalize the knowledge gained; low probability of recognition - 95.13%.To solve the above disadvantages, various techniques were used that increase the accuracy and speed of work, as well as reduce the effect of network retraining.
Keywords: neural network, pattern recognition, neural network algorithms, accuracy, network training, network retraining.

RESUMEN

Hoy, en la era digital, el problema del reconocimiento de patrones es muy relevante. En particular, la tarea de reconocimiento de texto es importante en la banca, para la lectura automática de documentos y su

control; en sistemas de control de video, por ejemplo, para identificar la matrícula de un automóvil que violó las reglas de tránsito; en los sistemas de seguridad, por ejemplo, para verificar los billetes en un cajero automático y en muchas otras áreas. Se conocen una gran cantidad de métodos para resolver el problema del reconocimiento de patrones, pero la principal ventaja de las redes neuronales sobre otros métodos es su capacidad de aprendizaje. Es esta característica la que hace que las redes neuronales sean atractivas para estudiar. El artículo propone un modelo básico de red neuronal. Se consideran los principales algoritmos y se implementa un modelo de programación en el lenguaje de programación Python. En el curso de la investigación, se revelaron las siguientes deficiencias del modelo básico: baja tasa de aprendizaje (el número de dígitos reconocidos correctamente en las primeras épocas de aprendizaje); reentrenamiento: la red no ha aprendido a generalizar los conocimientos adquiridos; baja probabilidad de reconocimiento: 95,13%. Para resolver las desventajas anteriores, se utilizaron varias técnicas que aumentan la precisión y la velocidad del trabajo, así como también reducen el efecto del reentrenamiento de la red.
Palabras clave: red neuronal, reconocimiento de patrones, algoritmos de redes neuronales, precisión, entrenamiento de redes, reentrenamiento de redes.

1. INTRODUCTION

This article is devoted to the recognition of handwritten characters using neural network algorithms (Haykin, 2006; Kruglov, 2002; Rachkovskij & Kussul, 2001).
The chosen topic of work is conditioned by a number of reasons (Kaplan, 2001): recognition of printed characters using a computer is a well-researched field today, while the problem of recognizing handwritten characters is not trivial and, undoubtedly, a big breakthrough in this area is still ahead. In electronic document management systems, the input data can be both printed and handwritten documents; in addition, there are, for example, handwritten texts.
The problem of recognizing handwritten characters has been known since the 1970-ies, but there are still some problem areas, both theoretical and practical. Today the problem of recognition is being dealt with both in Russia and abroad: the University of Massachusetts Amherst (USA), Moscow State University (RF) and others (Kleyko et al., 2016).
There are two types of handwritten character recognition (Zaentsev, 1999): 1. In real time, when recognition occurs immediately after the character is entered; such systems include touch input systems. A large number of algorithms have been developed that effectively solve this problem, the accuracy of which reaches 98%. These algorithms use the trajectory of the input device; 2. Recognition of already written documents. The systems that solve this problem have a low accuracy of 80-85%. Such systems process the images obtained, for example, from a scanner.
The purpose of the article is to create and improve a neural network to solve the problem of recognizing handwritten characters. To achieve this goal, it is necessary to solve a number of tasks. Create a basic neural network model. Explore the problems that arise in the base model and find an approach to their solution. Implement an experimental handwritten character recognition system based on an advanced neural network.

2. MATERIALS AND METHODS

The problem of handwritten letters and symbol recognition can be divided into parts (Rasmussen & Eliasmith, 2011; Osovsky, 2002). First, it is necessary to split the images containing many numbers into a sequence of separate images, each of which contains one number. Let's say you want to split an image (Figure 1) into six separate images (Figure 2).

Figure 1: An example of an image of a handwritten number

Figure 2: Examples of number images

People solve the segmentation problem easily, but it is difficult for a computer program. When the image is segmented, the program must classify each individual digit (Mohamad, 2003; Martyshkin & Martens-Atyushev, 2019; Martyshkin et al., 2018). The segmentation problem is not difficult to solve if there is a good classifier of individual numbers. There are many approaches to solve the segmentation problem. One of the approaches used is splitting the image in different ways and, using the classifier, determination of the best segmentation performed. The best segmentation scores a high score if the classifier confidently classifies all segments, and a low score if the classifier is difficult on multiple segments. The idea is that the classifier becomes difficult if the segmentation was not done correctly. This approach and others can be used to solve the segmentation problem. So we will focus on the development of a neural network in the article that can solve a more interesting and complex problem - the recognition of individual handwritten digits (Bryukhomitsky, 2005; Medvedev & Potemkin, 2002; Wasserman, 1992). A three-layer neural network was designed to recognize individual digits.

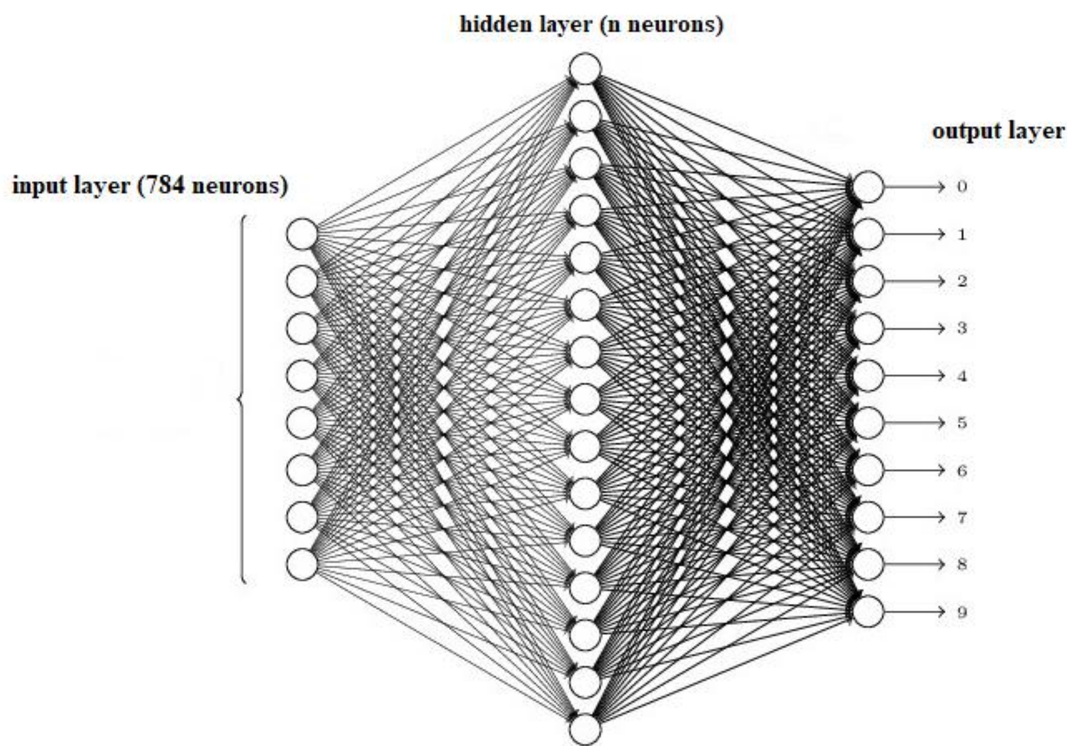

Figure 3. Network architecture

The input layer contains the neurons that receive pixels for input. The training data for the neural network consists of a set of 28 x 28 pixel images, the scanned images of handwritten digits, so the input layer contains 784 = 28 x 28 neurons. For simplicity, most of the input neurons have been omitted in the picture. Pixels are grayscale, where 0 represents white, 1 is black, and the values between 0 and 1 represent grayscale.
In the second layer of the network (hidden layer), the number of neurons is declared equal to n. Later in the article we will experiment with this number.
The output layer contains 10 neurons. If the first neuron fires, i.e. its output ≈ 1, then this indicates that the network classified the figure on the image as 0. If the second neuron fires, it means that the network decided that the figure shows the number 2, etc. Neurons are numbered starting at 0.
During the design process, we asked ourselves why we use exactly 10 output neurons? After all, we only need 4 neurons, the value of each is binary, these four neurons are enough for us to encode the response. Let's consider the first case with 10 output neurons. The first output neuron, the one that decides whether the picture is 0 or not, does so by weighing the evidence from the hidden layer of neurons. What are neurons doing in the hidden layer? For example, suppose the first neuron in the hidden layer determines whether the picture is similar to the one shown on Figure 4.

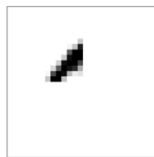

Figure 4. Part of the digit 0 image.

It can do this by heavily weighing the input pixels that overlap the image and easily weighing everything else. Likewise, suppose the second, the third, and the fourth neurons in the hidden layer determine whether the pictures shown on Figure 5 are similar.

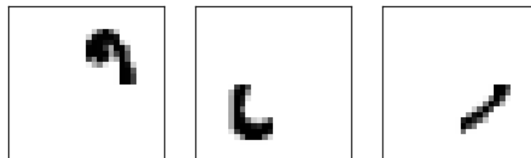

Figure 5. The parts of digit 0 image.

The images shown on Figures 4-5 together make up an image of the number 0. If all of these hidden neurons are firing, then we can conclude that the figure 0 is represented in the image. Assuming that the neural network functions in this way, we can explain, why is it better to have 10 output neurons than 4. If we had 4 output neurons, then the first neuron will try to decide which was the most significant bit. There is no easy way to link the most significant bit in a simple form like the ones above. It is difficult to imagine that there is any reason to associate the parts of a digit shape with, for example, the most significant bit.

Network training by gradient descent

Now, when the network is designed, you need to train it to recognize numbers. The first thing we need is a training dataset, the so-called test (training) dataset. In this article, we will use the MNIST database, which contains tens of thousands of scanned images of handwritten numbers

along with their correct classification. MNIST is a modified subset of two databases compiled by NIST (United States' National Institute of Standards and Technology). Here are some samples of the images from MNIST (Figure 6).

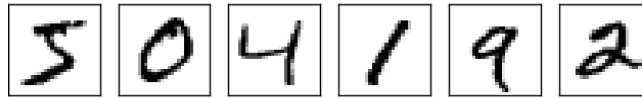
Figure 6. The examples of number images from MNIST

The MNIST database has two parts. The first part contains 60,000 images used as training data. These are scanned images of handwritten numbers written by 250 people, half of whom worked for the US Census Bureau and the other half were students. These images are 28 by 28 pixels in grayscale. The second part of the MNIST database consists of 10,000 images that will be used as verification data. These are also 28 x 28 pixel images. We will use test data to assess how well the network has learned to recognize numbers. The verification data comes from another set, from another 250 people. This will give us confidence that our system can recognize the numbers written by people who are not in the training data.
We declare the notation x to denote the training input image. It will be convenient to assume that each input image x is a $28 \times 28 = 784$-dimensional vector. Each element in the vector represents a single pixel grayscale value. Let's declare the corresponding desired output $y=y(x)$, where y – 10-dimensional vector. For example, if the number 6 is shown on the training image x, then y(x) = (0,0,0,0,0,0,1,0,0,0) T is the desired output value. We need an algorithm that will help us find weights and biases so that the output value of the network y(x) is approximated for all training images x. To determine how successfully we achieved this goal, we define the cost function:

$$C(w,b) \equiv \frac{1}{2n} \sum_x \| y(x) - a \|^2 \tag{1}$$

Here w denotes the collection of all weights in the network, b - all biases, n is the number of input patterns, a is the output vector of the network, when x is the input vector, the summation takes place over all input patterns. The notation "v" denotes the norm for the vector v. We will call C the squared loss function, also known as the mean square deviation (MSE). From the formula we see that C, w, b is not negative, since each term in the sum is not negative. Also, C w, $b \approx 0$ when y (x) is roughly equal to the output a for all training images x. So, our learning algorithm will work well if it can find weights and biases such that C w, $b \approx 0$. Otherwise, if C w, b is high, then our learning algorithm is not doing its job well. The goal of the learning algorithm is to minimize the loss function C w, b. In other words, we want to find a set of weights and biases that make the loss function as small as possible. We will do this using an algorithm known as gradient descent.
The number of correctly classified images is not a smooth function of network weights and biases. In most cases, small changes in weights and biases will not improve the classification performance of the network. This makes it difficult to understand how weights and biases need to be changed to achieve network performance improvements. If instead we use a loss function, for example, a squared loss function, then it turns out to be easy to understand what changes need to be made to reduce the loss. In summary, the goal of training a neural network is to find weights and biases that minimize the loss function C w, b. This is a well-posed task, but now it contains many subtasks - interpretation of w and b as weights and biases, σ function, choice of network

architecture, MNIST, etc. It will be clear much more if we abstract from all the details and concentrate on minimization. First, let's explore the gradient descent technique that will help solve the minimization problem, then return to the function we want to minimize. We minimize the function C (v) - it can be any real function of a set of variables, v = v1, v2,…. Replace w with v to emphasize that this can be any function. For simplicity, we represent the function C as the function of two variables v1, v2. We need to find the global minimum of C. Of course, we can do this just by looking at the graph, but in general, the function C can be a complex function of many variables, and usually it is impossible to just look at the graph and find a solution. We can imagine our function as a valley, and the ball rolling along the slope of the valley; the ball will eventually reach the very bottom (Figure 7). This idea will help us find the minimum of the function C.

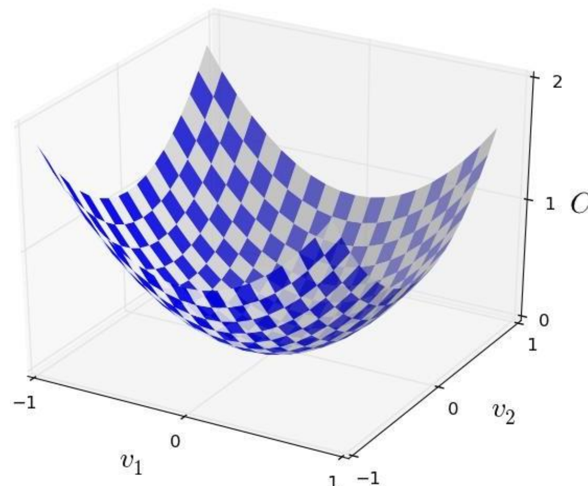

Figure 7. Visualization of a function of two variables v1, v2

When we move the ball by Δv_1 in the direction of v_1 and by Δv_2 in the direction of v_2, then C changes according to the following expression:

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2. \tag{2}$$

Now we need to choose Δv_1 and Δv_2 such that ΔC is negative, i.e. the ball should roll down the valley. To understand how to do this, we define Δv as the vector of changes in v, i.e. $\Delta v \equiv (\Delta v_1, \Delta v_2)\ T$. We will also define the gradient C as the vector of partial derivatives $\left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)^T$.

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)^T. \tag{3}$$

Now we can rewrite the expression for ΔC.

$$\Delta C \approx \nabla C \cdot \Delta v \tag{4}$$

This equation explains why ∇C is called the gradient vector: ∇C maps changes in v to the changes in C, which is what we wanted. But we can also figure out from the equation how we can choose Δv so that ΔC is negative. Let's say we chose

$$\Delta v = -\eta \nabla C, \tag{5}$$

where η is a small, positive parameter (known as the **learning factor**). Then from the equation (4) we obtain the following:

$$\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta \| \nabla C \|^2 \tag{6}$$

Since ∇C $2 \geq 0$, then $\Delta C \leq 0$, i.e. C will be constantly decreased, not increased, if we change v in accordance with the equation (5). This is what we were striving for. So, let's take the equation (5) to determine the law of motion for the ball in the gradient descent algorithm. We will use the equation (5) to calculate the value of Δv, and then move the ball by the indicated amount:

$$v \to v' = v - \eta \nabla C. \tag{7}$$

Then we apply the same rule for the next move. If we continue to move the ball according to this rule over and over again, then we will reach the global minimum of the function C. To summarize, the gradient descent algorithm repeatedly calculates the gradient ∇C, and then it moves in the opposite direction, descending the valley slope (Fig. 8). Visually, you can imagine it like this (Figure 8).

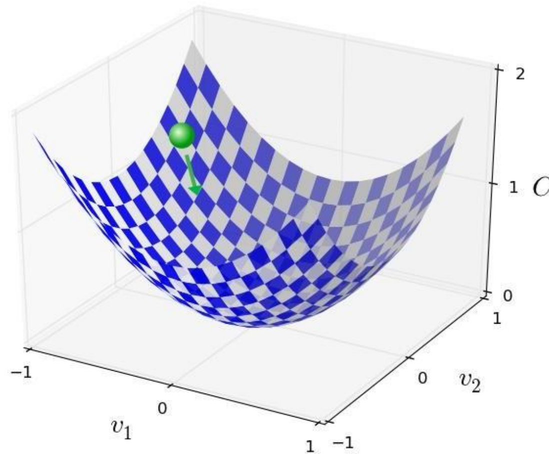

Figure 8. Visualization of the gradient descent algorithm

To make gradient descent work correctly, you need to choose a learning factor η small enough for the equation (4) to be a correct approximation. If this condition is not met, it may happen that $\Delta C > 0$. If we make η very small, then the algorithm will work very slowly. In practice, η is varied so that the equation (4) is met and the algorithm is fast enough. The function C can be the function of many variables. Suppose that C is the function of m variables $v_1,\ldots, vm$. Then the change of ΔC in C caused by a small change of $\Delta v = (\Delta v_1,\ldots, \Delta vm)$ T is equal to the following:

$$\Delta C \approx \nabla C \cdot \Delta v, \tag{8}$$

where the gradient ∇C vector

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \ldots, \frac{\partial C}{\partial v_m} \right)^T . \tag{9}$$

Just like in the case of two variables, we can choose the following:

$$\Delta v = -\eta \nabla C. \quad (10)$$

Thus, we guarantee that the expression (8) for ΔC will be negative. This makes it possible to find the minimum, even if the function C is the function of many variables:

$$v \to v' = v - \eta \nabla C. \quad (11)$$

How can you apply gradient descent to train a neural network? The idea is to use gradient descent to find weights wk and biases bl that minimize the loss function. Let's rewrite the update rule with weights and biases instead of vj. Now we have the components wk and bl, and the gradient vector ∇C has the corresponding components $\frac{\partial C}{\partial w_k}$ and $\frac{\partial C}{\partial b_l}$. Having rewritten the update rule for our components, we have the following:

$$w \to w_k^{'} = w_k - \eta \frac{\partial C}{\partial w_k} \quad (12)$$

$$b \to b_l^{'} = b_l - \eta \frac{\partial C}{\partial b_l} \quad (13)$$

Applying this rule many times, you can go down the hill and achieve the minimum of the loss function. In other words, it is a rule that can be used to train the network.

But there is one problem. Let's look at the quadratic loss function (1). More generally, the loss function looks like this: $C = \frac{1}{n} \sum_x C_x$ – the arithmetic mean of all functions $C_x \equiv \frac{\| y(x) - a \|^2}{2}$ for each training example. To compute the gradient ∇C, we have to compute the gradient ∇Cx for each input pattern x, and then compute the arithmetic mean. For a large number of training examples, this can take a long time, thus training is slow. Stochastic gradient descent can be used to speed up learning. The idea is to estimate the gradient ∇C by computing ∇Cx for a small set of randomly selected training examples.

It turns out that by averaging a small sample, you can quickly estimate the true gradient ∇C, which in turn makes learning faster.

Stochastic gradient descent selects a small number of m random training patterns. We'll call these training images X_1, X_2,…, X_m and refer to them as a mini-batch. The sample size m is sufficient for the mean value ∇C_{Xj} to be approximately equal to the mean value over all data ∇C_x

$$\frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C, \quad (14)$$

from which follows the following:

$$\nabla C \approx \frac{1}{m} \sum_{j=1}^{m} \nabla C_{X_j}, \quad (15)$$

Thus, we confirm that we can estimate the gradient over a randomly selected minigroup. In our case, stochastic gradient descent selects a random mini-group of training images and trains by this group.

$$w \rightarrow w_k^{'} = w_k - \frac{\eta}{m} \frac{\partial C_{X_j}}{\partial w_k} \quad (16)$$

$$b \rightarrow b_l^{'} = b_l - \frac{\eta}{m} \frac{\partial C_{X_j}}{\partial b_l} \quad (17)$$

where all the training examples Xj in the current mini-group are summed up, then we randomly choose another mini-group and train on it, and so on, until the training images run out, which tells us about the end of the learning era (Golovko, 2002). At this point, a new era of learning begins. It is worth noting that there are no clear rules about recalculating the loss function and using mini-groups to update weights and biases. In equation (1), we use the factor 1/n. Sometimes this factor is omitted and all loss functions for each training example are added instead of getting an average value. This is useful when the total number of training examples is not known in advance. This is possible if training examples are generated in real time. Also, the update rules by mini-groups (16) and (17) exclude the coefficient 1/m, this is equivalent to the change in the learning coefficient η.
Thus, stochastic gradient descent speeds up the network training process many times, for example, if we have the size of the training set n = 60,000, as in MNIST, and the size of the mini-group is m = 10. We get a 6000x speedup during the gradient calculation. Of course, this estimate will not be exact, but it does not have to be exact, all we need is to minimize the function C.

3. RESULTS AND DISCUSSION

Weight initialization

When they create a network, you need to set initial values for weights and biases. So far I have used the Gaussian distribution, with an expectation of 0 and a standard deviation of 1. This approach is viable, but using a different path can make our neural network faster. Suppose there is a network with 1000 input neurons, the training image x is fed to the input, after which half of the input neurons takes the value 1, and the other half becomes 0. The weighted sum of inputs for the hidden neuron is $z = \sum_j w_j x_j + b$. Half of the sum members are equal to 0, so 501 random variables are summed up. The quantity z is Gaussian with the expectation 0 and the standard deviation $\sqrt{501} \approx 22.4$.

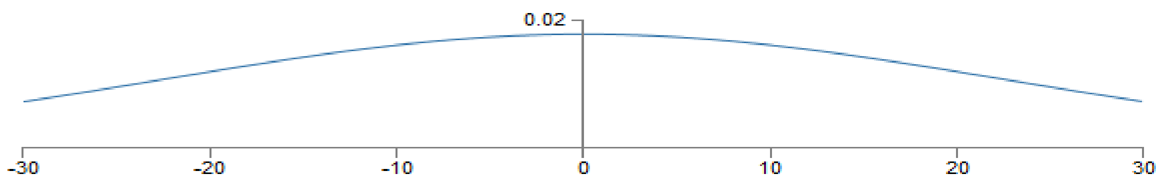

Figure 9. Distribution of a random variable

The graph (Figure 9) shows that with a high probability $z \gg 1$ or $z \ll -1$. In this case, the output value σ (z) of the hidden neuron will be saturated, and when the neuron is saturated, small

changes in weights will have no effect on the activation of the hidden neuron. This will cause the weights to adjust very slowly. To get rid of this problem, I propose to initialize the weights with Gaussian random variables with an expectation of 0 and the standard deviation $1/\sqrt{n_{in}}$, where n_{in} – is the number of input weights. Then z is distributed over a Gaussian with the mathematical expectation of 0 and a standard deviation of $\sqrt{3/2} \approx 1.22$. Figure 10 shows the distribution graph.

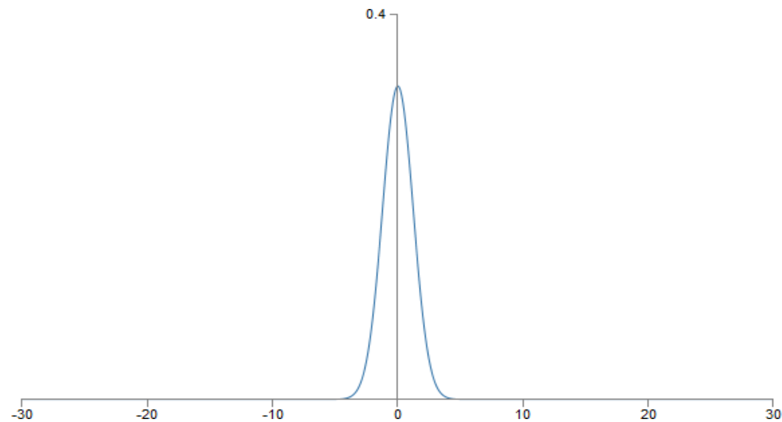

Figure 10. Distribution of a random variable

By initializing a neuron in this way, it is much less susceptible to saturation. Now let's do an experiment comparing different approaches to weight initialization.

Table 1. Initial data of the experiment 1

n	η	λ	x	e
0-50000	0.1	5.0	30	30

The results of the experiment are shown on Figure 11.

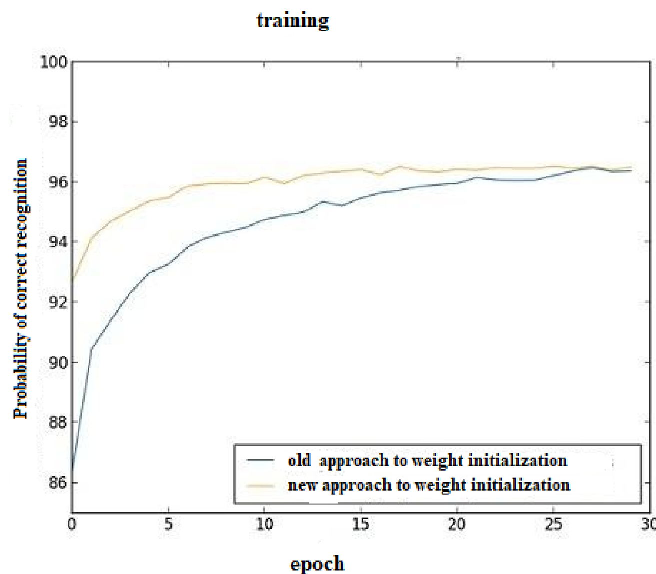

Figure 11. Probability of recognition with different approaches to weight initialization

In both cases, the recognition accuracy is about 96%. But the new approach to initialization is much faster. When they use the old approach at the end of the first epoch, we get the recognition accuracy of 87%, when using the new one - 93%. Thus, using a new approach to initializing weights, we get good results faster.

The number of neurons in the hidden layer

Let's return to the choice of the network complexity, i.e. the choice of the number of neurons in the hidden layer. Previous research and experiments have been conducted on a network with 30 or 100 neurons in the hidden layer. This was done in order to reduce training time. At the moment, there are no rules or formulas that determine the required number of neurons in the hidden layer of the neural network. If there are few neurons, then the network will be poorly trained. On the contrary, with a large number of the network, it will be subject to overfitting. The number of neurons is selected experimentally for each task, so we will experiment with the dependence of the recognition accuracy on the number of neurons in the hidden layer.

Table 2. Initial data of the experiment 2

n	η	λ	x	e	C
50000	0.1	5.0	30-100-300-600-800	60	Энтр.

The experiment results are presented on the figure 12.

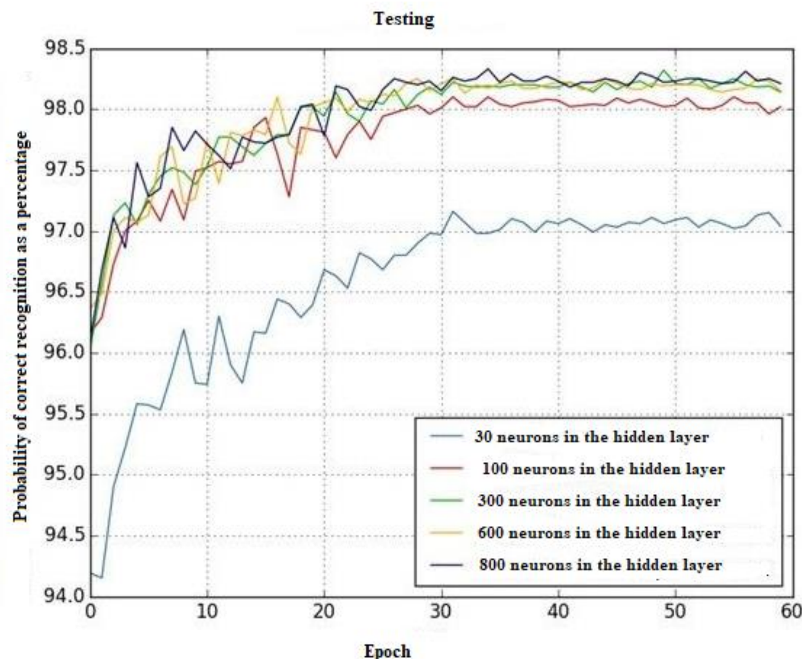

Figure 12. Recognition probability depending on the number of neurons in the hidden layer

As can be seen from the experiment, the networks with 300, 600, and 800 neurons in the hidden layer produce approximately the same recognition accuracy, but a network with 800 neurons is ahead by the epoch 33. On the other hand, the more neurons, the slower the learning takes place, so in our study we settled on 800 neurons in the hidden layer.

Learning rate

Choosing a learning rate is also not an easy task. If the value is too high, the network will give bad results as it can slip minimum. If the coefficient is too low, training can take a long time. Let's conduct an experiment to identify the dependence of the loss function value on the learning rate.

Table 3. Initial data of the experiment 3

n	η	λ	x	e	C
50000	0.025-0.25-2.5	5.0	100	30	Энтр.

The experimental results are shown on Figure 13.

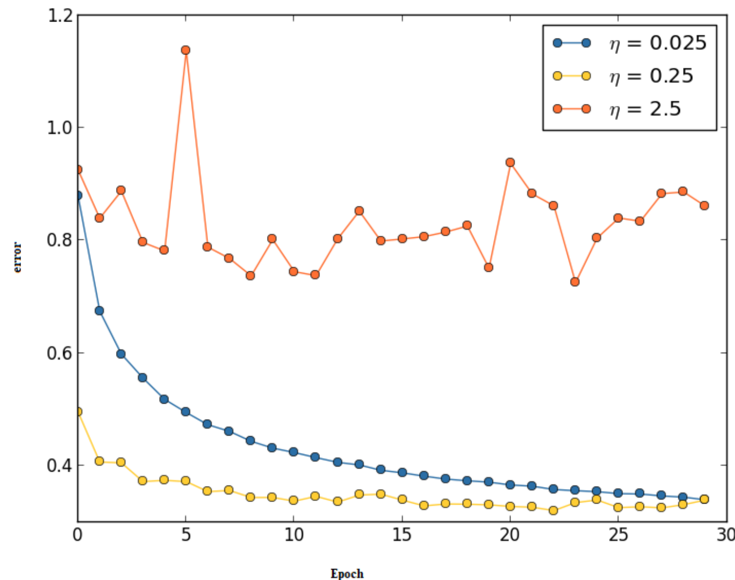

Figure 13. The graph of error dependence on learning rate

At η = 0.025, the error gradually decreases until the last epoch. At η = 0.25, the error decreases at first, but after the 20th epoch the network is saturated and the changes in the following epochs are small, rather, these are just random fluctuations. At η = 2.5, large fluctuations occur. So, for large η, there are correspondingly large steps, and we can skip the minimum. With η = 0.25, we get a smaller error value faster, but when we reach the minimum, we again face the problem of stepping over the minimum. In the latter case, everything is fine and the error gradually decreases. From the above, in order to reduce the error quickly, you can apply the dynamic reduction strategy η.

Creation of an improved neural network model. Advanced model configuration

Having studied the problems and methods for their solution arising in the basic model, relying on the techniques studied above and the experiments performed, it is possible to create an improved model of the basic neural network. The new network will consist of three neural layers. There will be 800 neurons in the hidden layer, a regularized cross-entropy function will be used as a loss function, instead of a quadratic one. A new approach to the initial initialization of weights will be applied to speed up learning. The learning factor will be changed dynamically, to speed up minimizing errors. Also, to achieve greater recognition accuracy, the network will be trained with 50,000 original images from MNIST, rotated images and the images subjected to elastic

distortion will be added. 250,000 images of the training set will be obtained in total. Below is an example of a rotated and deformed digit (Figure 14).

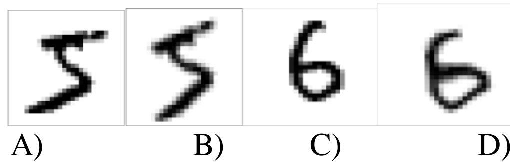

Figure 14. A - Original and B - rotated digit 5, C - original and D - deformed digit 6.

Let's experiment with the created network. In the first part of the experiment, 10,000 noise-free digits from the MNIST database will be used as a test sample, with which the network was not trained, and in the second part, noisy images of digits will be used as verification data. The images are the same as in the first part of the experiment, but noises have been added on images. Below is an example of noisy images that the network tried to recognize (Figure 15).

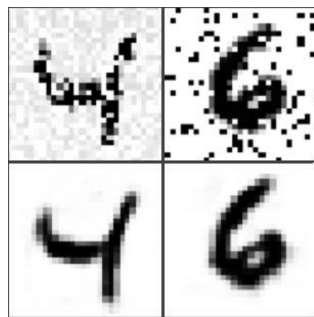

Figure 15. The example of noisy images of figures

Table 4. Initial data

n	η	λ	x	e	C
250000	0.8-0.025	5.0	800	60	Энтр.

The experimental results are shown on Figure 16.

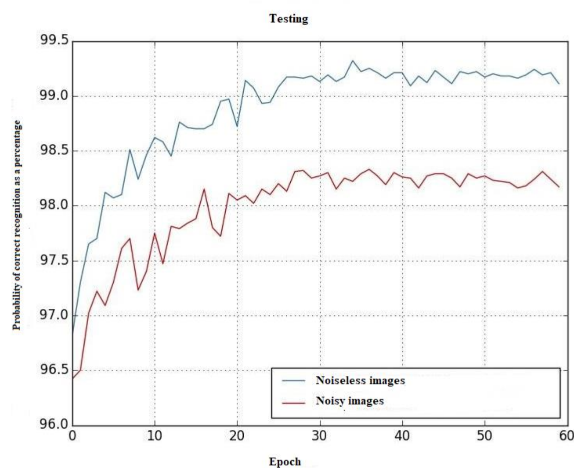

Figure 16. Experiment results

As can be seen from the graphs above, the network copes worse during noisy image recognition, but the difference is small - 1-2%. The best network performance for noiseless images is 99.32% (680 incorrectly recognized images), and 98.27% for noisy images.
It should be borne in mind that the test sample had the images of numbers that are difficult or even impossible to recognize for a person (Figure 17).

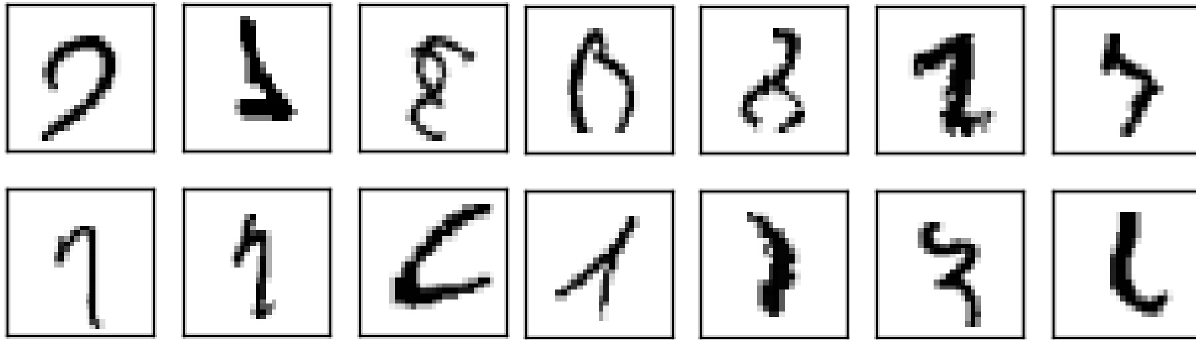

Figure 17. An example of hard-to-recognize images

According to the conducted experiments, it can be assumed that this method is suitable for recognizing handwritten characters. During the experiment, the network recognizes 9932 images of numbers from 10,000 and 9850 noisy images of numbers. These are good results if we consider that when there is noise, some of the numbers cannot be recognized even by a person. If we compare the results obtained with other methods for recognizing handwritten numbers from the MNIST database, then:
- SVM (Support Vector Machine) - 98.6% on noise-free images (LeCun et al., 2004).
- Convolutional neural network - 99.11% on noise-free images (Ranzato).
As can be seen from the above probabilities, the created network has a higher recognition probability, which makes it suitable for use in this area.

4. CONCLUSION

The article studied one of the models of neural networks - a multilayer neural network with forward propagation of a signal and back propagation of an error, which can be used for pattern recognition tasks. The experiments on the recognition of images of handwritten Arabic numerals have been carried out and the possibilities of this method have been revealed for pattern recognition.
The advantages of this network are the following: simplicity of architecture - consists of one hidden layer of neurons; the possibility of using various algorithms to improve the network; high percentage of recognition accuracy. Disadvantages of the network: the need to prepare a large amount of data for training; the network training takes a lot of time with a large training sample.
The first drawback of the network can be solved using the considered regularization method and by artificial increase of the training sample. The training time can be reduced by applying a different loss function. In this work they also considered the cross-entropy function due to the improved method of initializing the weights.
The paper presents the results of modeling, indicating that the created neural network can be successfully used to solve the problems of image recognition, in particular, handwritten characters.

5. ACKNOWLEDGMENTS

The reported study was funded by RFBR, project number 19-07-00516 A

6. REFERENCES

Bryukhomitsky, Yu.A. (2005). Neural network models for information security systems. Tutorial. - Taganrog: Publishing house of TRTU, - 160 p.

Golovko, V. (2002). Neural networks: training, organization and application.- IPRZhR,. - 256 p.

Haykin, S. (2006). Neural networks: a complete course, 2nd edition. Trans. from English. - M.: Publishing house "Williams", - 1104 p.

Kaplan, R. (2001). Basic concepts of neural networks. Trans. from English. - M.: Williams Publishing House, - 288 p.

Kleyko, D., Osipov, E., Senior, A., Asad, I., Khan, Ahmet, Y. (2016). Holographic Graph Neuron: A Bioinspired Architecture for Pattern Processing. In IEEE Transactions on Neural Networks and Learning Systems*, 99*: 1-13.

Kruglov, V.V. (2002). Artificial neural networks: Theory and practice / Kruglov V.V., Borisov V.V. - Moscow: Hot Line-Telecom, - 382 p.
LeCun, Y., Huang, F. J., & Bottou, L. (2004, June). Learning methods for generic object recognition with invariance to pose and lighting. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR. (Vol. 2, pp. II-104).

Martyshkin, A.I., & Martens-Atyushev, D.S. (2019). Mathematical modelling and evaluation of the characteristics of specialized reconfigurable systems based on a common bus at the stage of synthesis of the system configuration. Journal of Advanced Research in Dynamical and Control Systems*, 11*(8 Special Issue), 2852-2860

Martyshkin, A.I., Salnikov, I.I., Pashchenko, D.V., & Trokoz, D.A. (2018). Associative Co-processor on the Basis of Programmable Logical Integrated Circuits for Special Purpose Computer Systems Proceedings - Global Smart Industry Conference, GloSIC, article No. 8570067.

Medvedev, V.S., & Potemkin, V.G. (2002). Neural networks. MATLAB 6 / Ed. by Ph.D. in engineering V.G. Potemkin. - M.: DIALOG-MEPhI, - 496 p.

Mohamad, H.H. (2003). Fundamentals of Artificial Neural Networks. A Bradford Book, – 511 p.

Osovsky, S. (2002). Neural networks for information processing / Trans. from Polish by I.D. Rudinsky. - M.: Finance and Statistics, - 344 p.

Rachkovskij, D.A., & Kussul, E.M. (2001). Binding and normalization of binary sparse distributed representations by context-dependent thinning. Neural Computation*,* 13(2): 411-452.

Ranzato, M. A., Krizhevsky, A., & Hinton, G. (2010, March). Factored 3-way restricted boltzmann machines for modeling natural images. In Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 621-628).

Rasmussen, D., & Eliasmith, E. (2011). A neural model of rule generation in inductive reasoning. Topics Cognit. Sci.*, 3*(1): 140–153.

Wasserman, F. (1992). Neurocomputer technology: Theory and practice. Trans. from English, - 118 p.