Investigation of adversarial attacks on pattern recognition neural networks

Denis Vladimirovich Kotlyarov, student of the North Caucasian Federal University, den14kotlyarov@yandex.ru
Gleb Dmitrievich Dyudyun, student of the North Caucasian Federal University, gleb.dudun@gmail.com
Natalya Vitalievna Rzhevskaya, student of the North Caucasian Federal University, natalia070901@gmail.com
Maria Anatolyevna Lapina, Associate Professor, Department of Information Security of Automated Systems, mlapina@ncfu.ru
Mikhail Vladimirovich Babenko, Head of the Department of Computational Mathematics and Cybernetics

North Caucasian Federal University

Abstract. This article discusses the algorithm for creating a neural network based on pattern recognition. Several types of attacks on neural networks are considered, the main features of such attacks are described. An analysis of the Adversarial attack was carried out. The results of experimental testing of the proposed attack are presented. Confirmation of the hypothesis about the decrease in the accuracy of recognition of the neural network during the implementation of the attack by an attacker was obtained.

Keywords: Neural network, machine learning, pattern recognition, artificial intelligence, attack algorithm, information security, Adversarial attack, malicious machine learning.

Introduction

In modern times, such concepts as "neural network", "artificial intelligence" and other end-to-end technologies are used in various fields, such technologies are tightly integrated into our lives and are often used everywhere. Now nobody is surprised by the use of search algorithms from various advanced companies. Even the older generation now often uses voice assistants, without thinking that these are yet another “brainchild” of a neural network. In fact, a neural network can now do quite a lot: generate scientific reports, write poetry or a song, draw a picture that is not much different from the real one, the main thing is to train it correctly.

But such a network can serve both good and evil, depending on the purpose of the developer. So, for example, when recognizing images, an attacker can purposefully introduce errors into the recognition process, trying to force the system to incorrectly recognize the image being processed [1]. As a result, so-called spoofing attacks appear. Often, such attacks can be used in cases where an attacker seeks to disguise himself as another person and thereby commit illegal actions.

Literature review

At present, it is difficult to unambiguously define neural networks. After analyzing the study of several authors, we can say that a neural network or ANN is a learning system, which is a certain mathematical model built on the principle of human neurons, as well as its software implementation [3, 4]. Ivanyuk V.A. claims that artificial neural networks can be used to create intelligent decision-making systems, simulation modeling, expert systems [3].

A neural network is a mathematical model made up of interconnected nodes that work together to solve a problem. The nodes are arranged in layers, and each node performs a simple mathematical operation on the input to produce the output [5].

Kachagina K.S. in her research gives a range of applications of a neural network in everyday life. So in this study, we can highlight that the ANN is already used in security organizations, law enforcement agencies, at various factories and much more [4]. Therefore, it is very important to organize the security of these systems, since further the scope of neural systems will only grow.

It should be noted that the main tasks of neural networks are reduced to:

- Classification, that is, the separation of a certain object with a certain attribute from others.
- Prediction, this task often serves the interests of the financial world.
- Recognition, which will help to simplify the work, for example, for law enforcement agencies.
- Solving problems without a teacher.

In recent years, there has been an introduction into information and telecommunication systems as a means of identification, and often authentication of users [1]. According to experts, the introduction of such technologies often brings with it massive discontent from the outside. The reason for this was that the neural network is imperfect. Such a system has a number of vulnerabilities that will be used to disable it [4].

Since each person is unique, by spoofing biometric data, attackers can describe such data mathematically and use it as input to machine learning algorithms in order to automate the recognition process, and then use such ANN to replace their identity.

There are many attacks on neural networks that prevent the system from working properly. An attacker can carry out large-scale attacks without being noticed. For example, in biometric systems, an attacker can intentionally introduce errors into the process of recognizing biometric data. Ensuring the security of such systems is an important issue.

Article [10] describes how adversarial attacks work by exploiting vulnerabilities in neural networks that can be easily fooled by small noises or modifications to the input data that are imperceptible to humans, but can cause the network to misclassify the input data.

Consider some types of attacks on biometric systems that disrupt the recognition process [1]:

Fast Gradient Sign Method - an attack with noise overlay on the image with each new iteration. This attack is quite effective when constantly analyzing the image. This type of attack is practically unrealizable in the absence of direct access to data.

Using Infrared LEDs to Change Human Facial Features

Overlaying black or white stickers on the image for incorrect recognition

The use of devices that allow you to identify a person for another.

Hostile attacks are a growing concern in the field of artificial intelligence because they can be used to trick neural networks into misclassifying inputs.

One of the first studies on Adversarial Attacks was carried out by Szegedy [15], who showed that neural networks can be fooled by small noise inputs. Since then, a large amount of research has been done on this topic, including the development of new attack methods and defense strategies.

One of the most common types of contention attacks is the Fast Gradient Sign Method (FGSM), which was introduced by Goodfellow [13]. This method involves calculating the gradient of the loss function with respect to the input data and then modifying the data in the direction of the gradient to maximize the loss. Many subsequent studies have relied on this method, including the iterative FGSM (IFGSM) attack presented by Kurakin [14].

Article [10] also explains various types of hostile attacks, such as targeted and non-targeted attacks, and provides examples of real-life applications of hostile attacks, such as manipulating the systems of unmanned vehicles.

Other types of attacks that have been developed include the Jacobian-based saliency map attack [17], which uses the Jacobian matrix to determine the most sensitive input features, and the deep fool attack (Moosavi-Dezfooli et al., 2016), which generates small perturbations, which minimize the distance between the original input and the misclassified output.

Various strategies have been proposed to protect against these attacks. Adversarial learning involves augmenting the training data with adversarial examples to make the neural network more robust [16], while defensive distillation involves training a separate network to detect adversarial examples [17]. Other protection strategies include randomization, input transformation, and gradient masking [12].

Despite these defense strategies, adversary attacks remain a major threat to machine learning systems. As noted by Akhtar and Mian [11], attacks by the adversary can have serious consequences in the real world, such as causing self-driving cars to misinterpret road signs or medical systems to misdiagnose diseases.

In addition, article [10] discusses some of the techniques that have been developed to defend against adversary attacks, including adversary training and defensive distillation.

In recent years, researchers have also studied the impact of adversarial attacks on object detection systems [18], semantic segmentation (Xie et al., 2017), and generative models (Samangouei et al., 2018). These studies have shown that enemy attacks are effective against these systems and have proposed new defense methods to improve their reliability.

Despite significant progress in the development of adversarial attacks and defenses, there are still open issues that require further research. One of these tasks is the development of effective and reliable methods of protection. Another challenge is understanding the vulnerabilities of deep learning models to attack by malicious actors and finding ways to fix them.

Several studies [13, 15, 18] have examined the effectiveness of adversarial attacks on various types of machine learning models, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and autoencoders. For example, Xu et al. (2020) have shown that adversarial attacks can be effective against RNN-based text classifiers even if the attacks are generated using a different language model.

To protect against enemy attacks, researchers have proposed various defense strategies. One common defense strategy is adversarial learning, which involves training a model on adversarial examples in addition to regular training data [16]. Other security strategies include gradient masking, which involves hiding gradients from an attacker [17], and feature compression, which involves preprocessing input data to remove redundant features [19].

Several recent studies have also explored the use of generative models such as Generative Adversarial Networks (GANs) to generate adversarial examples [18]. These models can be used to create more realistic examples of competitive actions that are harder to detect and more difficult to defend against.

Biometric authentication systems, which use physiological or behavioral characteristics to verify people's identities, have become increasingly popular in recent years. However, these systems are not immune from attacks, and neural networks are used to carry out attacks on biometric data [19].

One common type of attack on biometrics is presentation attacks, also known as spoofing attacks, where an attacker uses a fake or artificial biometric to impersonate a legitimate user. Neural networks have been used to create attacks with a realistic representation of various biometric modalities, including fingerprints, face recognition, and voice recognition.

For example, Wang et al. (2019) used a convolutional neural network (CNN) to generate realistic fingerprints that can be used to fake fingerprint recognition systems. Similarly, Nguyen [21] used a generative adversarial network (GAN) to generate synthetic facial images that can be used to fake facial recognition systems.

Other research has focused on using neural networks to launch attacks on biometric data by exploiting vulnerabilities in the biometric system. For example, Li et al. (2019) proposed a method for generating adversarial examples for fingerprint recognition systems by distorting the input fingerprint image using a gradient-based optimization method. The resulting fingerprint of the attacker can be used to avoid detection by the biometric system.

To protect against attacks on biometric data, researchers have proposed various defense strategies, including the use of liveness detection techniques to detect attacks on presentations and the use of deep neural networks to increase the resilience of biometric systems to attacks. For example, Tan et al. [22] proposed a deep neural network liveliness detection method for detecting presentational attacks in face recognition systems.

Materials and research methods

Using the MNIST dataset as an example, we can consider the principle of building a certain neural network, and then explore its vulnerability. Based on the research of V.A. Ivanyuk, any neural network is mathematically a superposition of regression functions that describe the relationship between the values of the inputs and outputs of the network [3].

The input layer receives data in the form of features or input variables, which are then passed to the first hidden layer. Each node in the hidden layer performs a linear transformation of the input using weights and biases and then applies a non-linear activation function to produce a non-linear output. This output is then fed as input to the next level, and the process is repeated until the final output is obtained.

Starting the study, it is worth saying that the basic element of such systems is the so-called neuron. It is necessary in order to create a programming model. A neuron in an ANN is an artificial analogue of a real neuron, only represented as a simple mathematical function that determines the rules for generating an output signal based on input data [3]. In the work of K.S. Kachagina stated that a neuron is an imaginary black object with several input and one output hole [4].

A neural network can be represented mathematically as a function f(x; θ), where x is the input, θ are the network parameters (weights and biases), and f(x; θ) is the output of the network.

The weights and biases in the neural network are adjusted during training to optimize network performance. This is done by minimizing the cost function, which measures the difference between projected output and actual output. The backpropagation algorithm is used to update the weights and biases in such a way as to reduce the loss function.

A neural network consists of layers of interconnected nodes, and the calculation of the output of each node can be represented mathematically as (1):

$$z = w \cdot x + b \tag{1}$$

where z – weighted sum of input parameters x, w – scale vector, b – displacement vector.

The output of a node is calculated by applying a non-linear activation function to a weighted sum, which can be represented mathematically according to (2):

$$a = g(z) \tag{2}$$

where g is the activation function.

The node layer output is calculated as follows (3):

$$a^1 = g(z^1) \tag{3}$$

where a^1 – exit of the first layer, z^1– weighted sum of first layer inputs, g is the activation function.

The output of the last layer is the output of the neural network, which can be used for prediction or classification.

During training, the weights and biases of the neural network are adjusted to minimize the cost function, which measures the difference between the predicted output and the actual output. This is done using gradient descent, where the gradient of the cost function with respect to weights and biases is computed, and the weights and biases are updated accordingly.

The backpropagation algorithm is used to efficiently compute the gradient of the cost function with respect to the weights and biases of the network.

Thus, the mathematical logic of a neural network includes calculating the weighted sum of the input data for each node, applying a non-linear activation function, and propagating the output through the layers of the network to obtain the final output. The network weights and biases are optimized during training to minimize the cost function using gradient descent and backpropagation.

It is worth deciding on the neural network training algorithm:

1. Import libraries
2. Data checking
   2.1 Checking for Lost Values
   2.22.2 Data normalization
3. Modeling
4. Getting a result

The neural network analyzes the biometric data by learning patterns and features from the input data during the training process. It can then use these learned features to predict new biometrics. For example, a facial recognition neural network can learn to recognize certain facial features and patterns, such as eye position or mouth shape, and use that information to identify people in new images. Similarly, a fingerprint recognition neural network can learn to recognize the unique ridges and patterns on a fingerprint and use that information to verify a person's identity.

Let's analyze this algorithm on a specific example.

It is necessary to create and train a neural network based on the MNIST database that will recognize handwritten numbers.

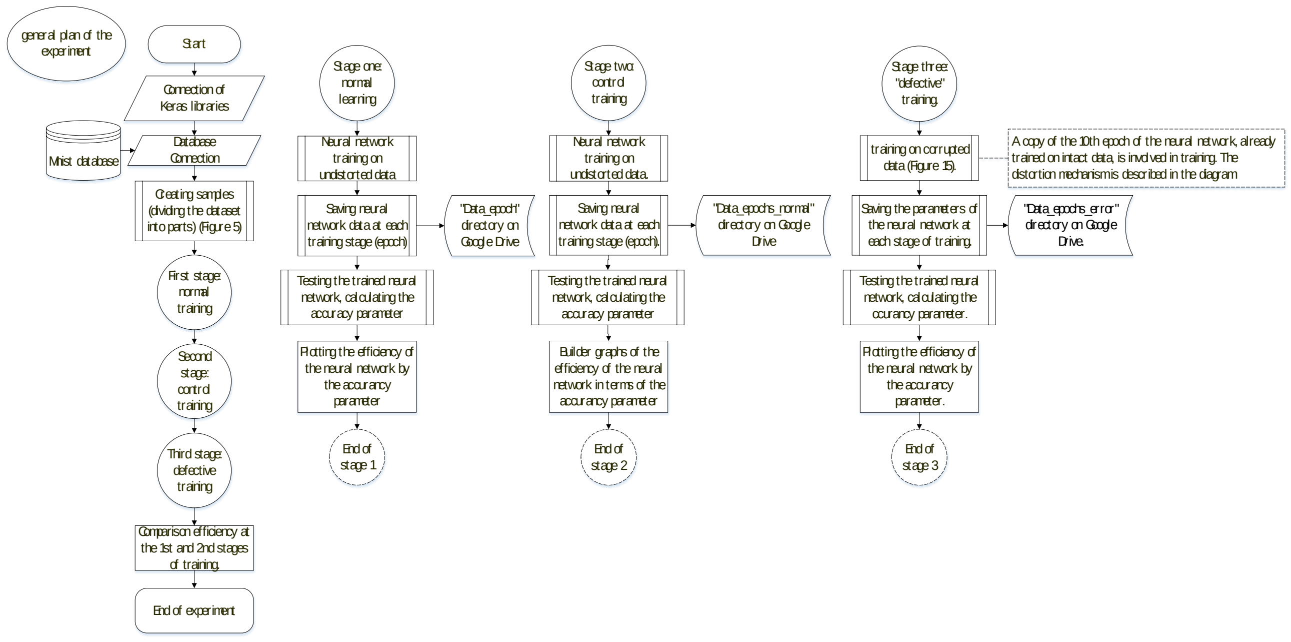

general plan of the experiment
Start
Connection of Keras libraries
Mnist database
Database Connection
Creating samples (dividing the dataset into parts) (Figure 5)
First stage: normal training
Second stage: control training
Third stage: defective training
Comparison efficiency at the 1st and 2nd stages of training.
End of experiment
Stage one: normal learning
Neural network training on undistorted data
Saving neural network data at each training stage (epoch)
"Data_epoch" directory on Google Drive
Testing the trained neural network, calculating the accuracy parameter
Plotting the efficiency of the neural network by the accurancy parameter
End of stage 1
Stage two: control training
Neural network training on undistorted data.
Saving neural network data at each training stage (epoch).
"Data_epochs_normal" directory on Google Drive
Testing the trained neural network, calculating the accuracy parameter
Builder graphs of the efficiency of the neural network in terms of the accurancy parameter
End of stage 2
Stage three: "defective" training.
training on corrupted data (Figure 15).
A copy of the 10th epoch of the neural network, already trained on intact data, is involved in training. The distortion mechanismis described in the diagram
Saving the parameters of the neural network at each stage of training.
"Data_epochs_error" directory on Google Drive.
Testing the trained neural network, calculating the ccurancy parameter.
Plotting the efficiency of the neural network by the accurancy parameter.
End of stage 3

Fig 1. General structure of the experiment algorithm

It is necessary to create and train a neural network based on the MNIST database that will distinguish between handwritten numbers.

The following libraries and modules have been used:

Numpy is a library for working with multidimensional arrays and matrices [27].

Keras is a framework for building and training neural networks. It is part of the Tensorflow library and allows you to create neural network models using a set of high-level abstractions.

Import the NumPy library and give it the alias "np". We import the MNIST dataset from the Keras library. MNIST contains images of numbers from 0 to 9, which will be used to train the neural network [28].

Sequential is a neural network model in which layers are added sequentially one after the other. Dense and Flatten layers from the Keras library. Adam is an optimizer from the Keras library. Optimizers are used to adjust the neural network weights during training. SparseCategoricalCrossentropy is a loss function from the Keras library. The loss function is a metric that evaluates how well a neural network performs on a classification task.

Matplotlib is used for data visualization. ModelCheckpoint is a class that allows you to save model weights during training. The load_model function from the Keras library is used to load a saved model from a file [28].

Next, we connect Google Drive to Google Colab. Google Drive is used to save the model and other files.

The shutil module in Python provides a high-level interface for working with files and directories. It contains functions for copying, moving, renaming and deleting files and directories.

```
import numpy as np
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import SparseCategoricalCrossentropy
import matplotlib.pyplot as plt
from tensorflow.keras.callbacks import ModelCheckpoint
from keras.models import load_model
from google.colab import drive
drive.mount('/content/gdrive')
import shutil
```

We load the MNIST dataset and split it into training and test data, where `x_train` and `x_test` – numpy arrays containing images of handwritten digits. Each image is a 28x28 matrix of pixels. Each pixel corresponds to a value from 0 to 255 (various shades of black, white and gray)

`y_train` and `y_test` – numpy arrays containing labels for the corresponding images in the training and test sets. The label is an integer from 0 to 9 that corresponds to the handwritten digit on the corresponding image.

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

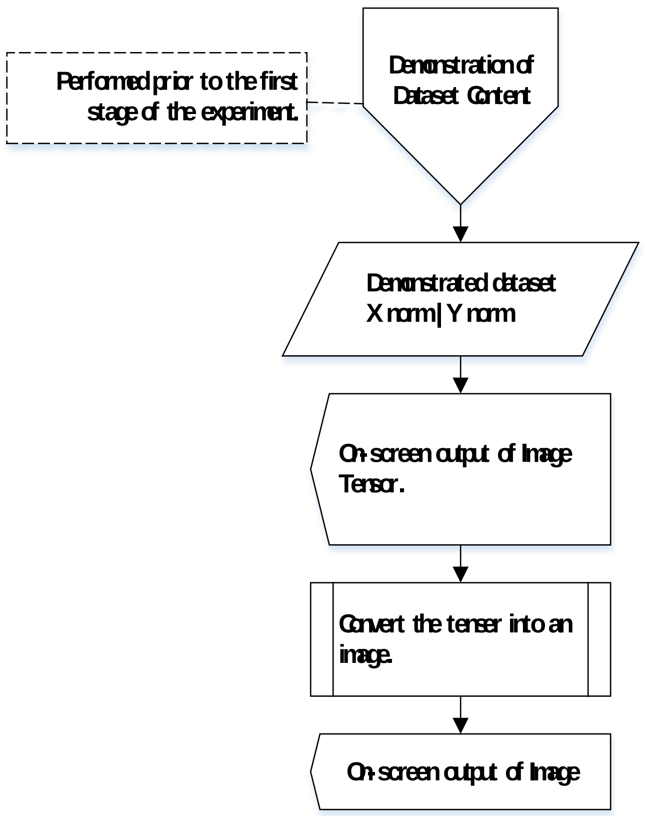
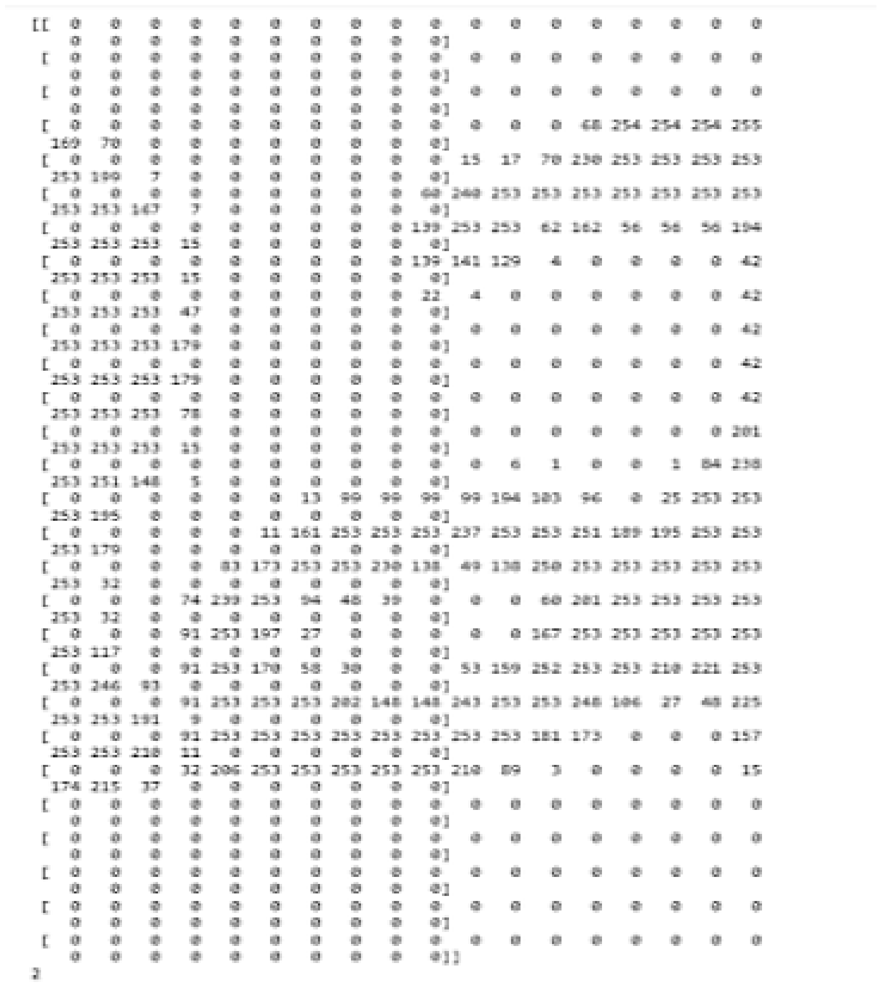

Performed prior to the first stage of the experiment.
Demonstration of Dataset Content
Demonstrated dataset X norm | Y norm
On-screen output of Image Tensor.
Convert the tenser into an image.
On-screen output of Image

Fig 2.The structure of the algorithm of the module for demonstrating the contents of the Mnist dataset

For clarity, we derive one of the elements `x_train and y_train` in the form in which they are stored in tuples. Take element number 277.

```
print(x_train[277])

print(y_train[277])
```



```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  68 254 254 254 255
  169  70   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  15  17  70 230 253 253 253 253
  253 199   7   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  60 240 253 253 253 253 253 253 253
  253 253 167   7   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0 139 253 253  62 162  56  56  56 194
  253 253 253  15   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0 139 141 129   4   0   0   0   0  42
  253 253 253  15   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  22   4   0   0   0   0   0   0  42
  253 253 253  47   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  42
  253 253 253 179   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  42
  253 253 253 179   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  42
  253 253 253  78   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 201
  253 253 253  15   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   6   1   0   0   1  84 238
  253 251 148   5   0   0   0   0   0   0]
 [  0   0   0   0   0   0  13  99  99  99  99 194 183  96   0  25 253 253
  253 195   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0  11 161 253 253 253 237 253 253 251 189 195 253 253
  253 179   0   0   0   0   0   0   0   0]
 [  0   0   0   0  83 173 253 253 230 138  49 138 250 253 253 253 253 253
  253  32   0   0   0   0   0   0   0   0]
 [  0   0   0  74 239 253  94  48  39   0   0   0  60 201 253 253 253 253
  253  32   0   0   0   0   0   0   0   0]
 [  0   0   0  91 253 197  27   0   0   0   0   0 167 253 253 253 253 253
  253 117   0   0   0   0   0   0   0   0]
 [  0   0   0  91 253 170  58  30   0   0  53 159 252 253 253 210 221 253
  253 246  93   0   0   0   0   0   0   0]
 [  0   0   0  91 253 253 253 202 148 148 243 253 253 248 106  27  48 225
  253 253 191   9   0   0   0   0   0   0]
 [  0   0   0  91 253 253 253 253 253 253 253 253 181 173   0   0   0 157
  253 253 210  11   0   0   0   0   0   0]
 [  0   0   0  32 206 253 253 253 253 253 210  89   3   0   0   0   0  15
  174 215  37   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]]
2
```

Fig 3.Output of element 277

Display the image of the element `x_train[277]` in black and white (white is 0 and black is 255).

`plt.axis('off')` – this is a function that hides the coordinate axes on the image.

```
plt.imshow(x_train[277], cmap='binary')
plt.axis('off')
```

```
(-0.5, 27.5, 27.5, -0.5)
```

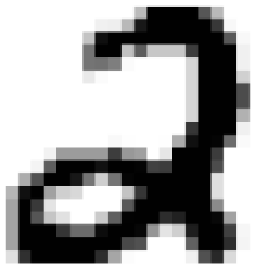


Fig 4.Image of element 277

Let's normalize the pixel values to bring the values from 0 to 255 to the range from 0 to 1. This is done by dividing each pixel value by 255. Pixel normalization improves the performance of the model, as it facilitates training and reduces the time required for processing data.

```
x_train = x_train / 255.0

x_test = x_test / 255.0
```

At the next stage, we split the training and test datasets into two equal parts. The first part will be used for the first training of the neural network, and the second part will be distorted and used in the second training.

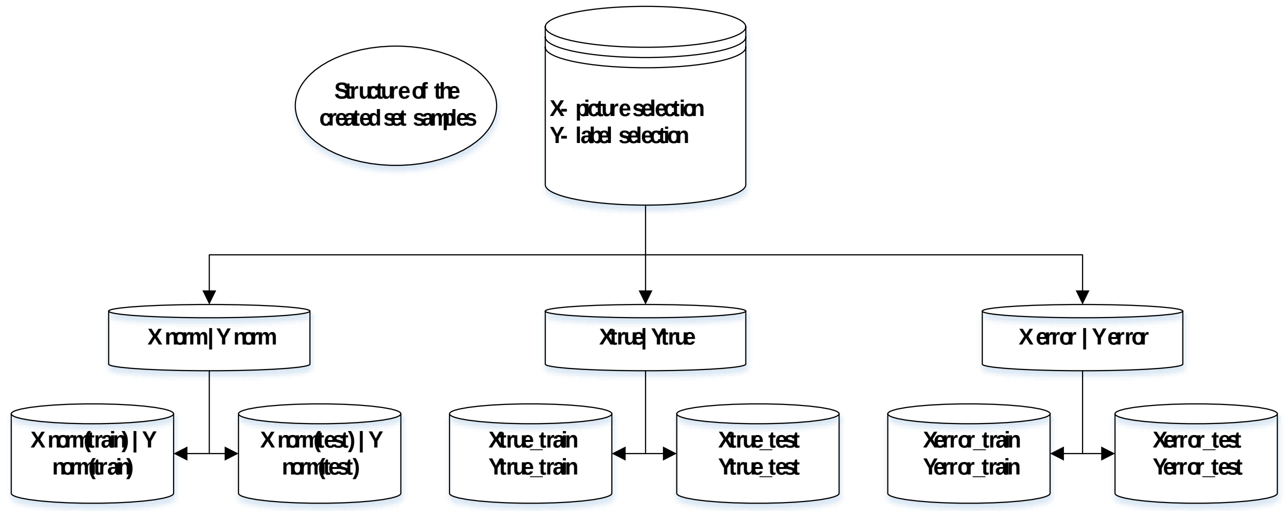


Fig 5.Scheme of splitting the dataset into samples

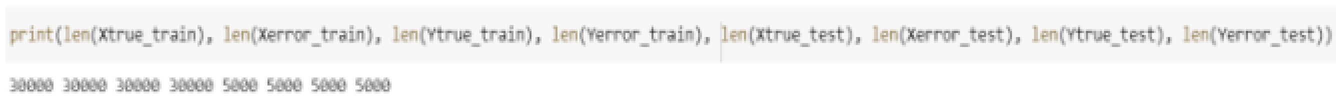
```
Xtrue_train, Xerror_train = np.split(x_train, 2)

Ytrue_train, Yerror_train = np.split(y_train, 2)

Xtrue_test, Xerror_test = np.split(x_test, 2)

Ytrue_test, Yerror_test = np.split(y_test, 2)
```

For clarity, we display the number of elements in each data set



```
print(len(Xtrue_train), len(Xerror_train), len(Ytrue_train), len(Yerror_train), len(Xtrue_test), len(Xerror_test), len(Ytrue_test), len(Yerror_test))

30000 30000 30000 30000 5000 5000 5000 5000
```

Fig 6.Displaying the lengths of tuples

We import the os module, which provides functionality for interacting with the operating system, such as creating, deleting and moving files and directories, getting information about file paths, and much more.

Create the necessary folders on Google Drive if none exist.

In `data_epochs` epoch files will be saved so that later you can analyze the work of the neural network.

```
import os

if not os.path.exists('/content/gdrive/My Drive/neyro'):

    os.makedirs('/content/gdrive/My Drive/neyro')

if not os.path.exists('/content/gdrive/My Drive/neyro/data_epochs'):
```

```
os.makedirs('/content/gdrive/My Drive/neyro/data_epochs')
```

Define the architecture of the model.

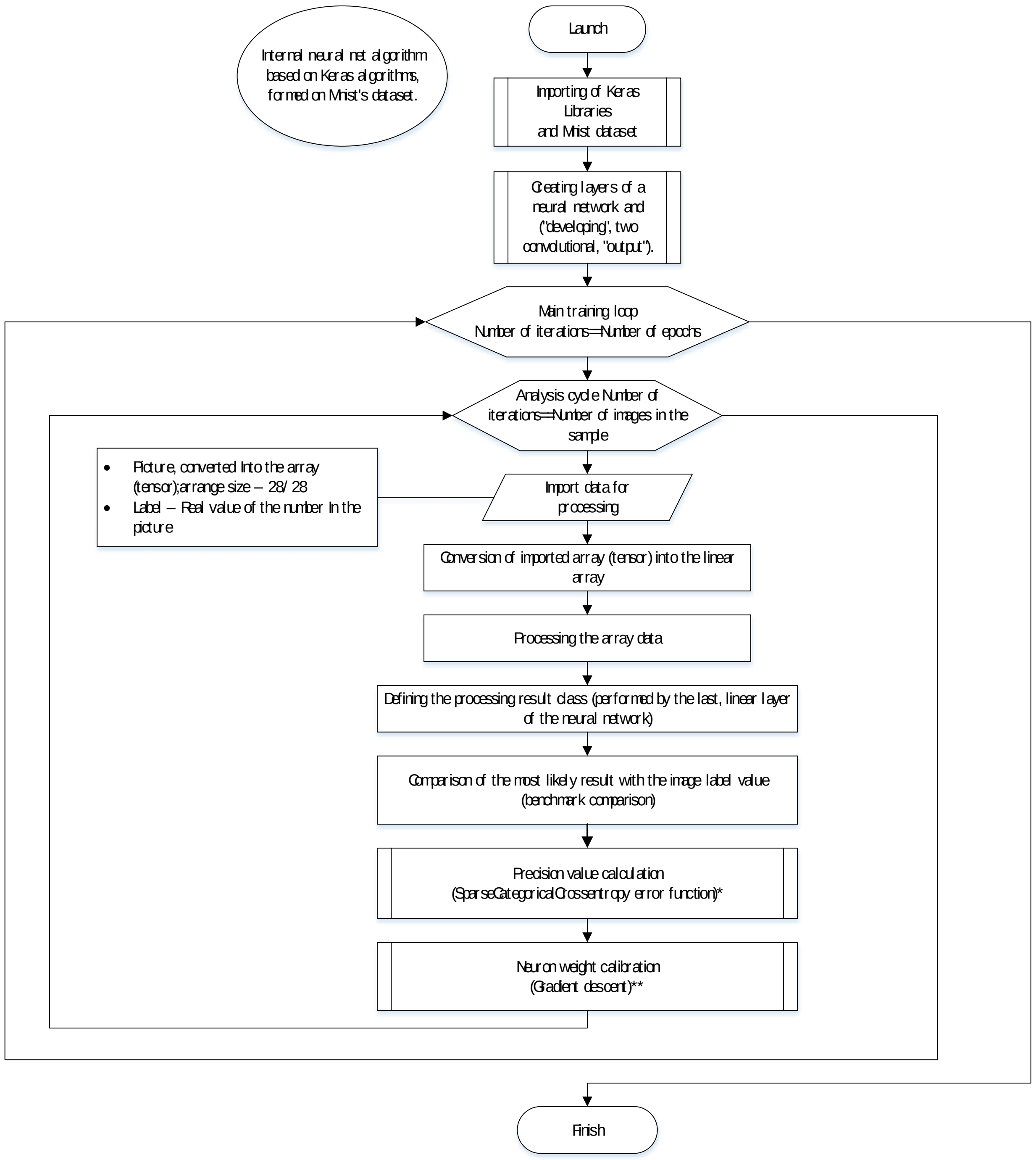

Internal neural net algorithm based on Keras algorithms, formed on Mnist's dataset.
Launch
Importing of Keras Libraries and Mnist dataset
Creating layers of a neural network and ("developing", two convolutional, "output").
Main training loop Number of iterations=Number of epochs
Analysis cycle Number of iterations=Number of images in the sample
Picture, converted Into the array (tensor);arrange size – 28/ 28
Label – Real value of the number In the picture
Import data for processing
Conversion of imported array (tensor) into the linear array
Processing the array data
Defining the processing result class (performed by the last, linear layer of the neural network)
Comparison of the most likely result with the image label value (benchmark comparison)
Precision value calculation (SparseCategoricalCrossentropy error function)*
Neuron weight calibration (Gradient descent)**
Finish

Fig 7. The structure of the algorithm of the internal mechanism of the neural network

Using the method of experiment, the optimal structure of the neural network and the parameters of its training were selected.

The first layer - Flatten converts a two-dimensional array (28, 28) into a one-dimensional array (dimension 784) so that it can be fed to the input of the neural network.

The second layer - Dense with 100 neurons and the ReLU (Rectified Linear Unit) activation function uses a linear operation followed by a non-linear activation function. The ReLU activation function returns 0 for negative values and the value itself for positive ones.

The third layer is Dense with 50 neurons and the ReLU activation function.

The fourth layer is Dense with 10 neurons and softmax activation function. softmax converts neuron values into probabilities summing up to 1.0 and is used for multi-class classification [29].

```
model = Sequential([
  Flatten(input_shape=(28, 28)),
  Dense(100, activation='relu'),
  Dense(50, activation='relu'),
  Dense(10, activation='softmax')
])
```

We compile the model with the necessary parameters.

The first optimizer parameter defines the optimization method that will be used to train the model. In this case, the optimizer `Adam` is used at the rate of learning (`learning_rate`) equal to 0.01.

Second parameter `loss` defines the loss function to be used during model training. Here we use categorical cross entropy (`SparseCategoricalCrossentropy`).

Third parameter `metrics` defines the metrics that will be used to evaluate the model. In this case, we will use only the accuracy metric (`accuracy`).

```
model.compile(optimizer=Adam(learning_rate=0.01),
              loss=SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```

Create a checkpoint object that is used to save the state of the model as a file in a folder `data_epochs` after each learning epoch.

```
checkpoint = ModelCheckpoint('/content/gdrive/My Drive/neyro/data_epoch
s/epoch_{epoch:02d}.h5')
```

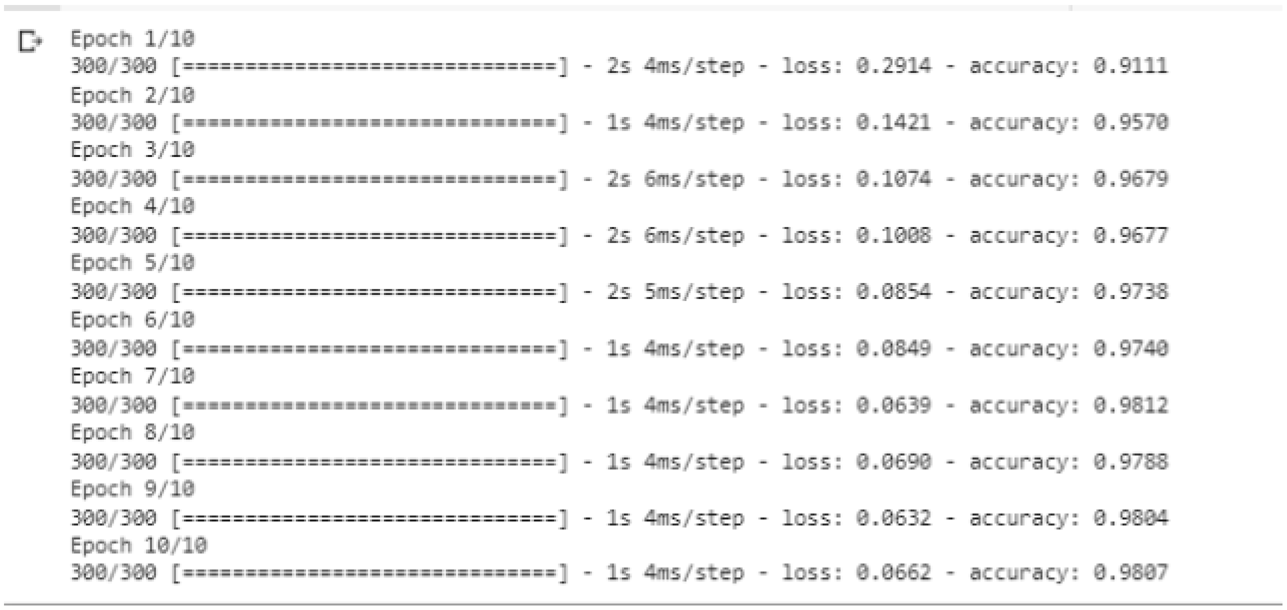
Train the model on training data `Xtrue_train` with appropriate labels `Ytrue_train`. Specify the number of epochs 10. For one training iteration, we take the batch size 100.

`shuffle=True` specifies that the training dataset will be shuffled before each epoch to avoid the possibility that the model might remember the order of the training examples.

`callbacks=[checkpoint]` indicates that the object `checkpoint` will be used as a callback to save the state of the model after each epoch.

Learning outcomes are saved to an object `history`. After training the model, history will contain information about the change in the loss function and accuracy metrics during model training.

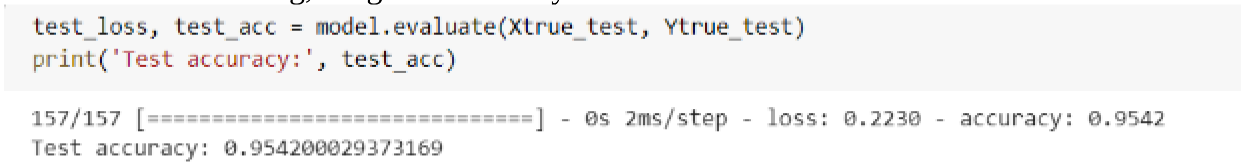
```
epochs = 10
history = model.fit(Xtrue_train, Ytrue_train, epochs=epochs, batch_size
=100, shuffle=True, callbacks=[checkpoint])
```



```
Epoch 1/10
300/300 [==============================] - 2s 4ms/step - loss: 0.2914 - accuracy: 0.9111
Epoch 2/10
300/300 [==============================] - 1s 4ms/step - loss: 0.1421 - accuracy: 0.9570
Epoch 3/10
300/300 [==============================] - 2s 6ms/step - loss: 0.1074 - accuracy: 0.9679
Epoch 4/10
300/300 [==============================] - 2s 6ms/step - loss: 0.1008 - accuracy: 0.9677
Epoch 5/10
300/300 [==============================] - 2s 5ms/step - loss: 0.0854 - accuracy: 0.9738
Epoch 6/10
300/300 [==============================] - 1s 4ms/step - loss: 0.0849 - accuracy: 0.9740
Epoch 7/10
300/300 [==============================] - 1s 4ms/step - loss: 0.0639 - accuracy: 0.9812
Epoch 8/10
300/300 [==============================] - 1s 4ms/step - loss: 0.0690 - accuracy: 0.9788
Epoch 9/10
300/300 [==============================] - 1s 4ms/step - loss: 0.0632 - accuracy: 0.9804
Epoch 10/10
300/300 [==============================] - 1s 4ms/step - loss: 0.0662 - accuracy: 0.9807
```

Fig 8. Data output after the first training

Let's evaluate the model on test data
After the first training, we got an accuracy of 0.9542 on test data



```
test_loss, test_acc = model.evaluate(Xtrue_test, Ytrue_test)
print('Test accuracy:', test_acc)
```

```
157/157 [==============================] - 0s 2ms/step - loss: 0.2230 - accuracy: 0.9542
Test accuracy: 0.954200029373169
```

Fig 9. Accuracy of training on test data

Derive the graphs of the first training

```
plt.plot(history.history['loss'])
plt.title('Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()
plt.plot(history.history['accuracy'])
plt.title('Training Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.show()
```

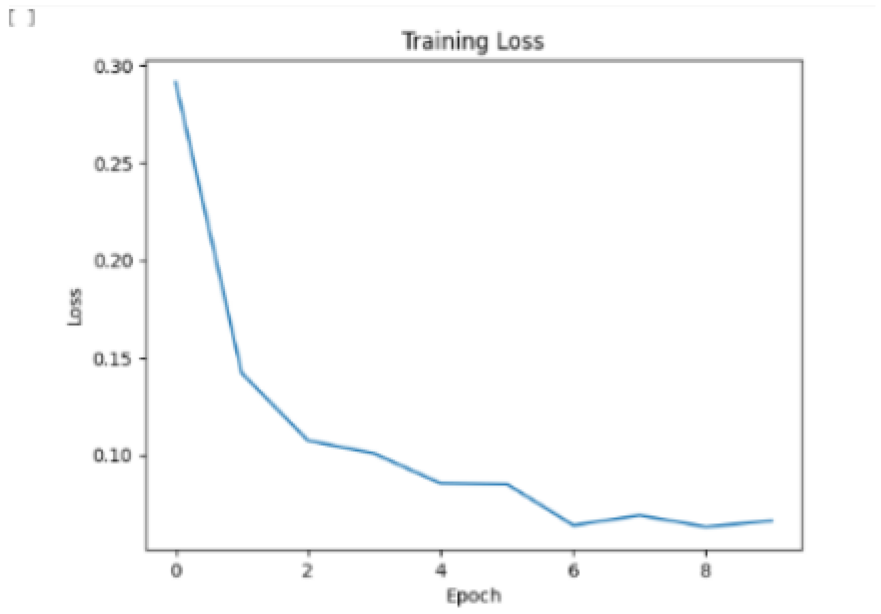


Fig 10. Graph of losses

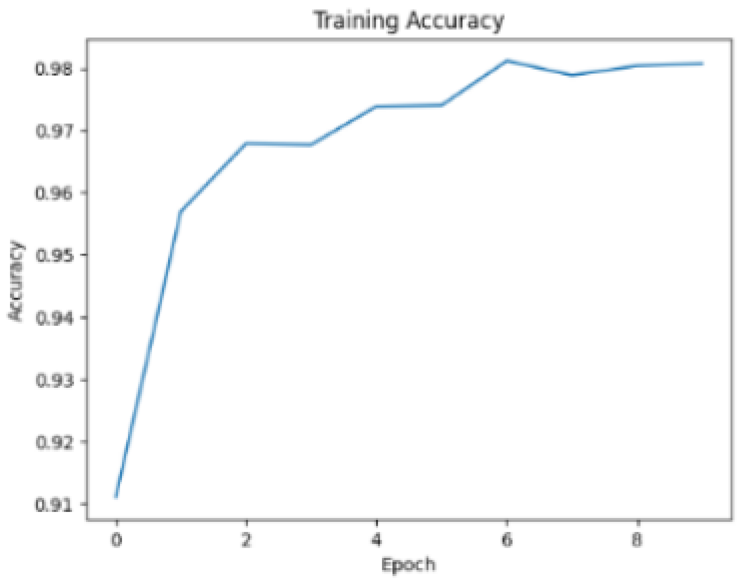


Fig 11. Graph of accuracy

Thus, the neural network was trained to recognize handwritten numbers on the MNIST dataset.

The next stage of work consists in distorting the dataset, that is, simulating the intervention of an attacker, and then analyzing the results after retraining the neural network.

The task is to find a way to calculate malicious interference in the operation of the neural network.

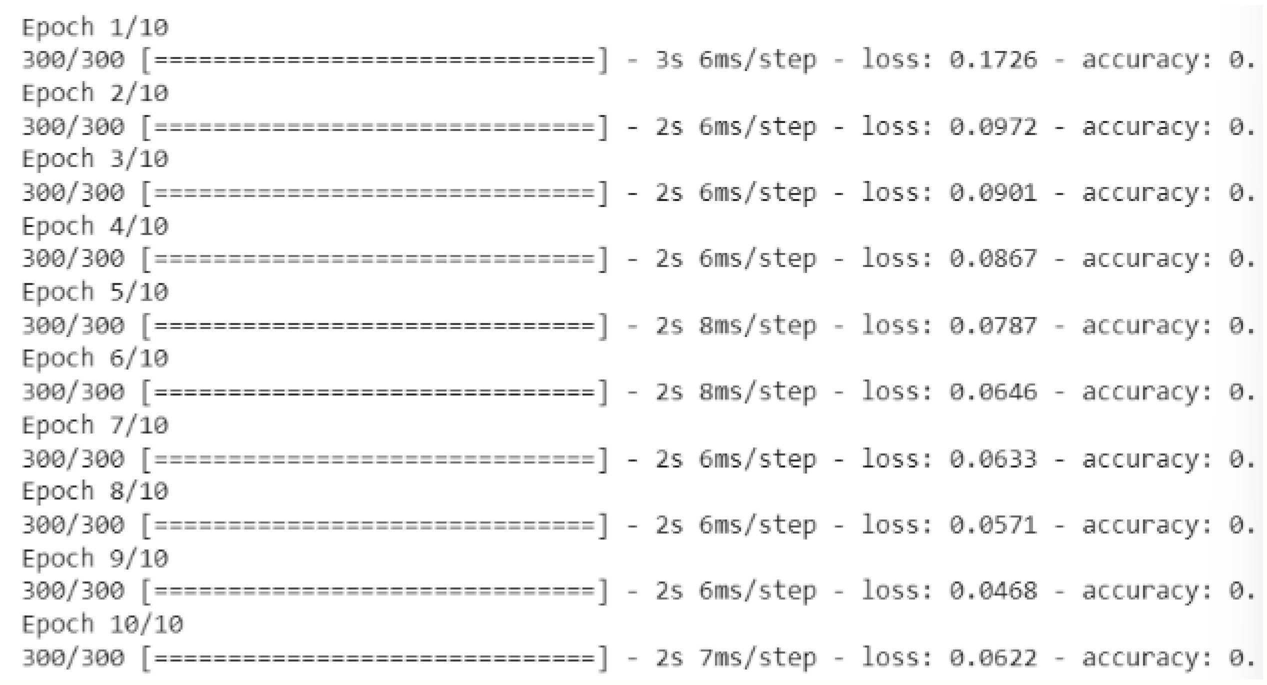
Let's put forward a hypothesis that one of the signs of interference in the neural network may be the difference in performance and accuracy to the trained neural network from the original one.

Let us test this hypothesis on a practical problem.

Before training the model on a distorted dataset, we will make a copy of the 10th epoch file and train the model loaded from it in exactly the same way, but on the second half of the dataset (`Xerror_train` и `Yerror_train`), but without changing anything in it.

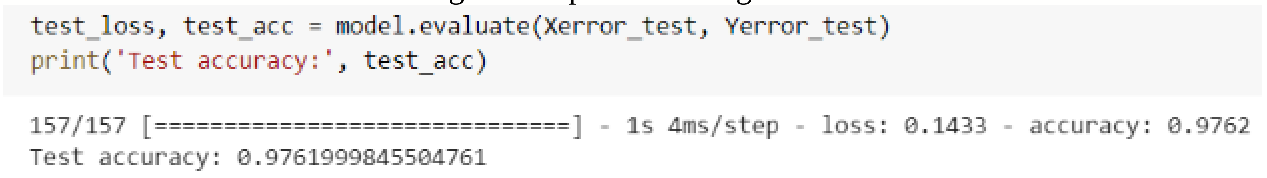
```
source_file = '/content/gdrive/My Drive/neyro/data_epochs/epoch_10.h5'
```

```
      destination_file = '/content/gdrive/My Drive/neyro/data_epochs/epoch_10
_copy1.h5'
      # Copying a file to a new path
      shutil.copyfile(source_file, destination_file)
      model = load_model('/content/gdrive/My Drive/neyro/data_epochs/epoch_10
_copy1.h5')
      #Model  compilation  with  Adam  optimizer,  SparseCategoricalCrossentropy
loss function and accuracy metric.
      history = model.compile(optimizer=Adam(learning_rate=0.01),
                    loss=SparseCategoricalCrossentropy(),
                    metrics=['accuracy'])
      # Creating a folder to save training data after each epoch
      import os
      if not os.path.exists('/content/gdrive/My Drive/neyro/data_epochs_conti
nue'):
          os.makedirs('/content/gdrive/My Drive/neyro/data_epochs_continue')
      # Saving the model after each epoch to folder data_epochs_error
      checkpoint = ModelCheckpoint('/content/gdrive/My Drive/neyro/data_epoch
s_continue/epoch_{epoch:02d}.h5')
      # Training the model on the second half (not yet distorted) of the dataset
      epochs = 10
      history = model.fit(Xerror_train, Yerror_train, epochs=epochs, batch_si
ze=100, shuffle=True, callbacks=[checkpoint])
```



```
Epoch 1/10
300/300 [==============================] - 3s 6ms/step - loss: 0.1726 - accuracy: 0.
Epoch 2/10
300/300 [==============================] - 2s 6ms/step - loss: 0.0972 - accuracy: 0.
Epoch 3/10
300/300 [==============================] - 2s 6ms/step - loss: 0.0901 - accuracy: 0.
Epoch 4/10
300/300 [==============================] - 2s 6ms/step - loss: 0.0867 - accuracy: 0.
Epoch 5/10
300/300 [==============================] - 2s 8ms/step - loss: 0.0787 - accuracy: 0.
Epoch 6/10
300/300 [==============================] - 2s 8ms/step - loss: 0.0646 - accuracy: 0.
Epoch 7/10
300/300 [==============================] - 2s 6ms/step - loss: 0.0633 - accuracy: 0.
Epoch 8/10
300/300 [==============================] - 2s 6ms/step - loss: 0.0571 - accuracy: 0.
Epoch 9/10
300/300 [==============================] - 2s 6ms/step - loss: 0.0468 - accuracy: 0.
Epoch 10/10
300/300 [==============================] - 2s 7ms/step - loss: 0.0622 - accuracy: 0.
```

Fig 12. Output of training results



```
test_loss, test_acc = model.evaluate(Xerror_test, Yerror_test)
print('Test accuracy:', test_acc)
```

```
157/157 [==============================] - 1s 4ms/step - loss: 0.1433 - accuracy: 0.9762
Test accuracy: 0.9761999845504761
```

Fig 13. Accuracy of training on test data

Accuracy after retraining increased from 0.9542 to 0.9762.

Let's create a copy of the epoch 10 file so that it can be used in the second training of the neural network and compare the received data.

```
source_file = '/content/gdrive/My Drive/neyro/data_epochs/epoch_10.h5'
destination_file = '/content/gdrive/My Drive/neyro/data_epochs/epoch_10_copy.h5'
shutil.copyfile(source_file, destination_file)
```

Load the neural network model from the file `epoch_10.h5`, stored in the data_epochs folder on Google Drive.

```
model = load_model('/content/gdrive/My Drive/neyro/data_epochs/epoch_10.h5')
```

Let's compile the model using the same parameters as the first time.

```
history = model.compile(optimizer=Adam(learning_rate=0.01),
                        loss=SparseCategoricalCrossentropy(),
                        metrics=['accuracy'])
```

Let's create a folder `data_epochs_error`, where the files of the epochs of the second training will be saved

```
import os
if not os.path.exists('/content/gdrive/My Drive/neyro/data_epochs_error'):
```

```
os.makedirs('/content/gdrive/My Drive/neyro/data_epochs_error')
```

Let's create an object with which we can save epoch files.

```
checkpoint = ModelCheckpoint('/content/gdrive/My Drive/neyro/data_epoch
s_error/epoch_{epoch:02d}.h5')
```

Training and testing of the neural network is carried out on the second half of the MNIST dataset (with the prefix error), which was created at the beginning of the code.

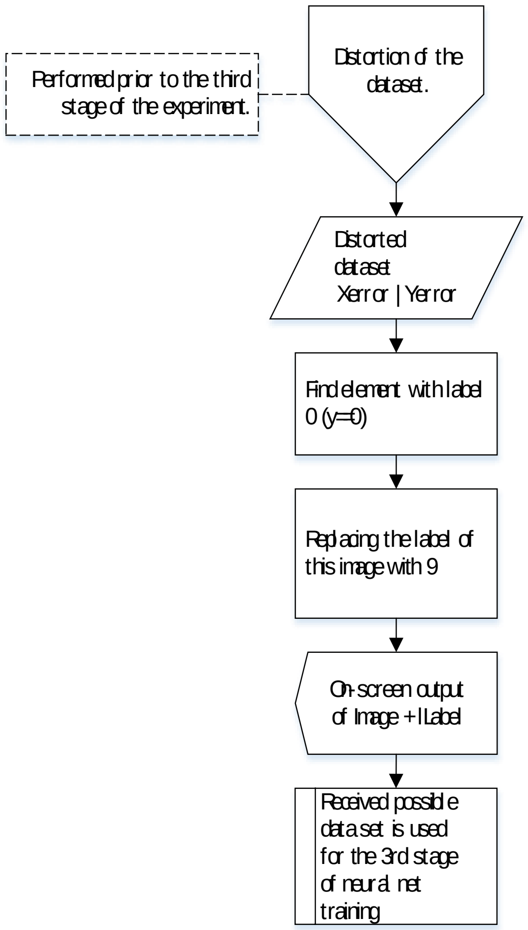
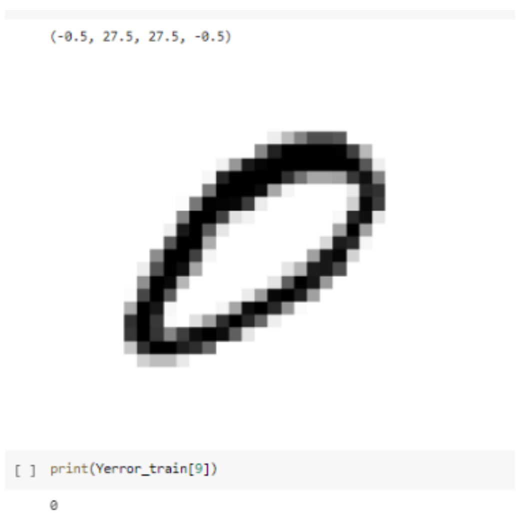

Performed prior to the third stage of the experiment.
Distortion of the dataset.
Distorted dataset Xerror | Yerror
Find element with label 0 (y=0)
Replacing the label of this image with 9
On-screen output of Image + lLabel
Received possible dataset is used for the 3rd stage of neural net training

Fig 14. The structure of the algorithm for direct distortion of the sample "Error"

Let's find an element from the data set whose value is 0 and display its image and its corresponding label.`plt.imshow(Xerror_train[9], cmap='binary')`

```
plt.axis('off')
```



```
(-0.5, 27.5, 27.5, -0.5)
```

```
[ ] print(Yerror_train[9])

0
```

Fig 15.Image of dataset element 0 and its label

Replace all label values in the training dataset `Yerror_train` with values 0 to 9.

This stage is an imitation of an attacker's action. This example can be projected onto real face recognition neural networks, that is, different dataset numbers are different faces of people.

One of the reasons for the emergence of the Adversarial attack is that machine learning methods were originally developed for stationary and secure environments, where the training and test samples were generated from the same statistical distribution. However, in practice, attackers can covertly manipulate input data to exploit vulnerabilities in ML algorithms and compromise the security of the entire machine learning system.

The type of attack in which an attacker seeks to gain access to the data and the learning process of an ML model in order to “poison” it (train it incorrectly) for subsequent inadequate work is called poisoning. It can be seen as a malicious infection of the training data. Thus, the “white box” strategy is used here, when the attacker has information about the victim - “malicious knowledge” (Adversarial Knowledge, AK): how the data for training are prepared and from what sources and what they are, what are the main functions of the attacked system , what algorithms it uses, what are the results, etc. Poison attacks involve insider information about the ML system and a fairly high level of attacker's competence in Data Science [26].

```
Yerror_train[Yerror_train == 0] = 9
```

Выведем элемент `Xerror_train[9]` с соответствующей ему меткой.

```
plt.imshow(Xerror_train[9], cmap='binary')

plt.axis('off')

print(Yerror_train[9])
```

```
9
```

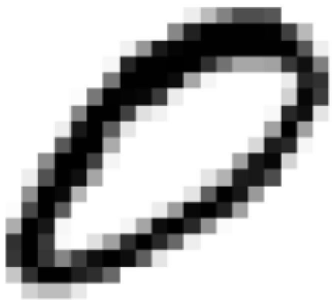


Fig 16. Image of dataset element 0 and its label after replacement

As a result, all images 0 correspond to labels with a value of 9.

Retrained the neural network on data sets `Xerror_train` and `Yerror_train`, using the same parameters as the first time.

```
epochs = 10
history = model.fit(Xerror_train, Yerror_train, epochs=epochs, batch_size=100, shuffle=True, callbacks=[checkpoint])
```

```
Epoch 1/10
300/300 [==============================] - 2s 4ms/step - loss: 0.2091 - accuracy: 0.9531
Epoch 2/10
300/300 [==============================] - 1s 4ms/step - loss: 0.1028 - accuracy: 0.9696
Epoch 3/10
300/300 [==============================] - 1s 4ms/step - loss: 0.0860 - accuracy: 0.9740
Epoch 4/10
300/300 [==============================] - 1s 4ms/step - loss: 0.0765 - accuracy: 0.9768
Epoch 5/10
300/300 [==============================] - 1s 5ms/step - loss: 0.0734 - accuracy: 0.9779
Epoch 6/10
300/300 [==============================] - 1s 4ms/step - loss: 0.0674 - accuracy: 0.9806
Epoch 7/10
300/300 [==============================] - 1s 5ms/step - loss: 0.0600 - accuracy: 0.9815
Epoch 8/10
300/300 [==============================] - 2s 6ms/step - loss: 0.0592 - accuracy: 0.9829
Epoch 9/10
300/300 [==============================] - 2s 7ms/step - loss: 0.0486 - accuracy: 0.9858
Epoch 10/10
300/300 [==============================] - 1s 4ms/step - loss: 0.0531 - accuracy: 0.9849
```

Fig 17. Data output after training

```
test_loss, test_acc = model.evaluate(Xerror_test, Yerror_test)
print('Test accuracy:', test_acc)
```

```
157/157 [==============================] - 0s 3ms/step - loss: 5.5475 - accuracy: 0.8758
Test accuracy: 0.8758000135421753
```

Fig 18. Accuracy on test data

```
plt.plot(history.history['loss'])
plt.title('Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()
plt.plot(history.history['accuracy'])
plt.title('Training Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.show()
```

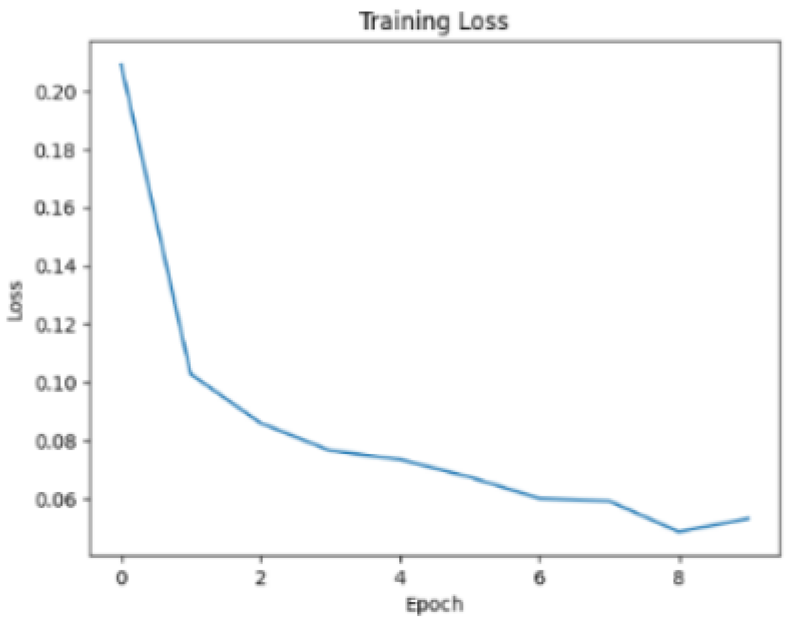


Fig 19. Graph of losses

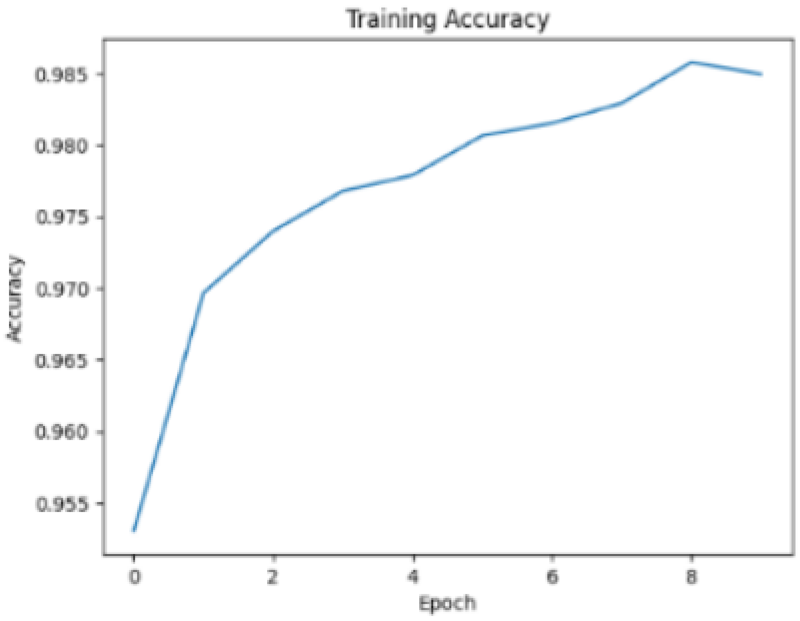


Fig 20. Graph of accuracy

We got an accuracy on the test data of 0.8758, which is much less than the result of 0.9762 obtained after the model was trained on an undistorted dataset, and also less than after the first training, where the result on the test data was 0.9542.

	First neural network training	**Additional training of a neural network on an undistorted dataset**	**Additional training of a neural network on a distorted dataset**
Accuracy on test data	0,9542	0,9762	0,8758

This confirms the hypothesis. Thus, when retraining a neural network on a distorted dataset, the accuracy on test data drops, and when retraining on an undistorted dataset, it increases.

This is due to the fact that the information received by the neural network about handwritten numbers from the MNIST dataset during initial training comes into conflict with the information received during re-training on the dataset with changed labels.

Therefore, a sharp decrease in the accuracy of the neural network during additional training is one of the signs of an attacker's intervention.

The presented method is one of the simplest methods for detecting malicious interference, and in further research it is planned to rely on the gradient descent method, which can be more accurate for this task. This is exactly what scientists from Cornell University did for the article "Interpreting Deep Neural Networks with SVCCA" [30].

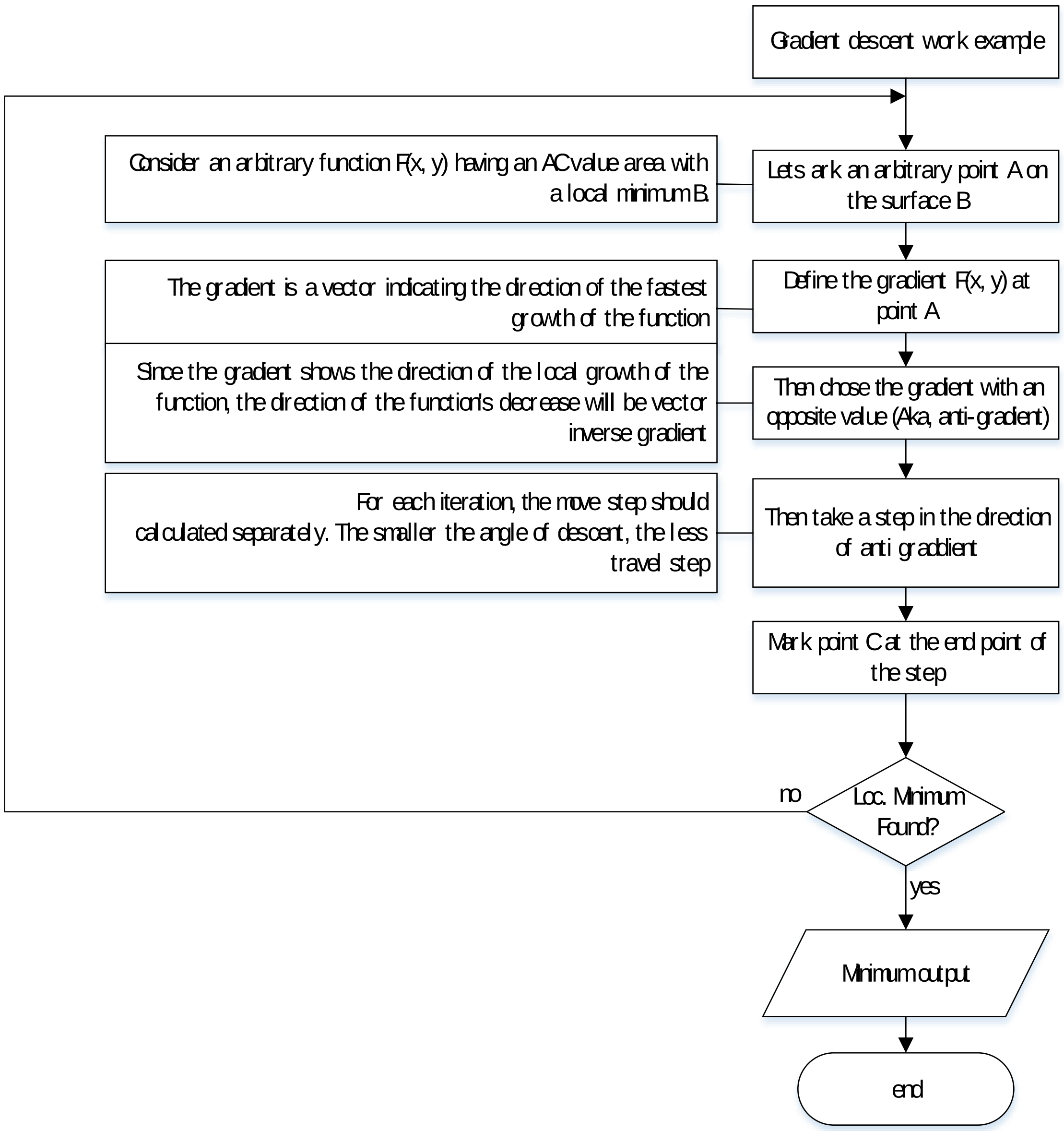

Gradient descent work example
Consider an arbitrary function F(x, y) having an AC value area with a local minimum B.
Lets ark an arbitrary point A on the surface B
The gradient is a vector indicating the direction of the fastest growth of the function
Define the gradient F(x, y) at point A
Since the gradient shows the direction of the local growth of the function, the direction of the function's decrease will be vector inverse gradient
Then chose the gradient with an opposite value (Aka, anti-gradient)
For each iteration, the move step should calculated separately. The smaller the angle of descent, the less travel step
Then take a step in the direction of anti gradient
Mark point C at the end point of the step
no
Loc. Minimum Found?
yes
Minimum output
end

Fig 21. Scheme of the gradient descent method theoretical algorithm

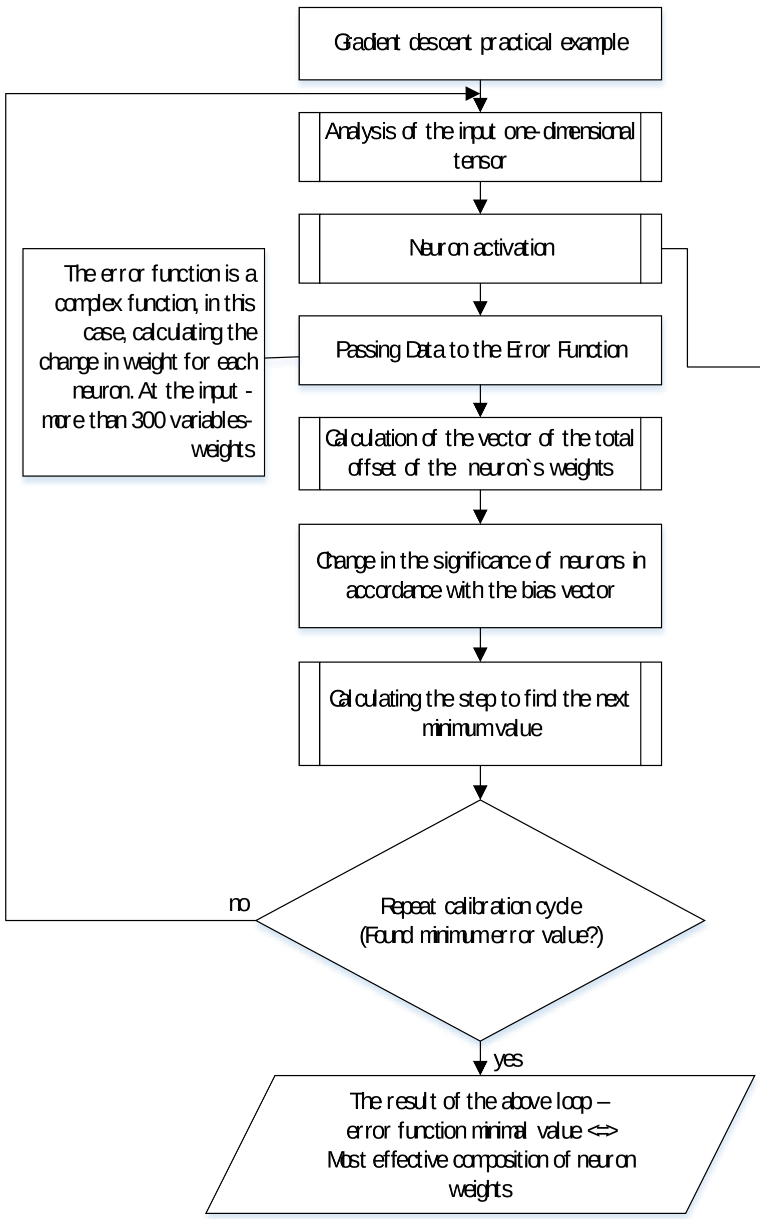

Gradient descent practical example
Analysis of the input one-dimensional tensor
Neuron activation
Passing Data to the Error Function
Calculation of the vector of the total offset of the neuron`s weights
Change in the significance of neurons in accordance with the bias vector
Calculating the step to find the next minimum value
Repeat calibration cycle (Found minimum error value?)
no
yes
The result of the above loop – error function minimal value ⇔ Most effective composition of neuron weights
The error function is a complex function, in this case, calculating the change in weight for each neuron. At the input - more than 300 variables-weights
Recognition of the desired sequence
Direct neuron activation
Direct transfer of data to the next layer with a certain "strength

Fig 22. Scheme of the gradient descent method practical algorithm

In conclusion, adversarial attacks are of particular interest in the field of artificial intelligence, and a large amount of research has been done in this direction. While defense strategies have been proposed and the arms race between attackers and defenders continues, new methods of attack are constantly being developed. Thus, it is important for researchers to study adversarial attacks and develop new protection strategies to ensure the security of machine learning systems, the formation of a secure model of trusted artificial intelligence.

References

1. Attacks on biometric systems // InformationSecurity URL: https://www.itsec.ru/articles/ataka-na-biometricheskie-sistemy [Online; accessed 30-March-2023].

2. Mamirkhodzhaev M. M., Umaraliev J. T., Sotvoldieva M. B, Tuychiboev A. E. Vozhmozhnosti nejronnyh setej [CAPABILITIES OF NEURAL NETWORKS] .Talqin va tadqiqotlar ilmiy-uslubiy jurnali, 2022, №6. URL: https://cyberleninka.ru/article/n/vozmozhnosti-neyronnyh-setey (date of access: 03/30/2023). (in Russian).

3. Ivanyuk V. A NEJRONNYE SETI I IH ANALIZ [NEURAL NETWORKS AND THEIR ANALYSIS]. Hronojekonomika [Chronoeconomics], 2021, No. 4 (32). URL: https://cyberleninka.ru/article/n/neyronnye-seti-i-ih-analiz (accessed 03/30/2023). (in Russian).

4. Kachagina K. S., Safarova A. D. NEJRONNYE SETI - PERSPEKTIVY RAZVITIJa [NERON NETWORKS - DEVELOPMENT PROSPECTS]. E-Scio [E-Scio], 2021, No. 2 (53). URL: https://cyberleninka.ru/article/n/neyronnye-seti-perspektivy-razvitiya (accessed 03.30.2023). (in Russian).

5. Akhtar Z., Luca Foresti G.7. Face Spoof Attack Recognition Using Discriminative Image Patches, Department of Mathematics and Computer Science, University of Udine, Via delle Scienze 206, 33100 Udine, Italy Journal of Electrical and Computer Engineering, Vol. 2016, Article ID 4721849. 14 pp

6. Namiot D.E., Ilyushin E.A., Chizhov I.V. ATAKI NA SISTEMY MAShINNOGO OBUChENIJa - OBShhIE PROBLEMY I METODY [ATTACKS ON MACHINE LEARNING SYSTEMS - GENERAL PROBLEMS AND METHODS]. Mezhdunarodnyj zhurnal otkrytyh informacionnyh tehnologij [International Journal of Open Information Technologies], 2022, №3. URL: https://cyberleninka.ru/article/n/ataki-na-sistemy-mashinnogo-obucheniya-obschie-problemy-i-methody (accessed 03/30/2023). (in Russian).

7. Gafarov F.M., Galimyanov A.F. ARTIFICIAL NEURAL NETWORKS AND THEIR APPLICATIONS. - 1st ed. - Kazan: Kazan University Press, 2018. - 121 p.

8. Charu Aggarwal Neural Networks and Deep Learning:. - 1st ed. - St. Petersburg: Dialectics LLC, 2020. - 752 p

9. Artemenko A.V., Golovko V. A. Analysis of neural network methods for recognizing computer viruses / Materials of breakout sessions. Youth Innovation Forum "INTRI" – 2010. — Minsk: GU "BelISA", 2017. – 239 p.

10. How to cheat a neural network or what is an Adversarial attack // ALEXEY CHERNOBROV ANALYST [Online] URL: https://chernobrovov.ru/articles/kak-obmanut-nejroset-ili-chto-takoe-adversarial-attack.html (accessed: 02.04.2023)

11. Akhtar, N., & Mian, A. (2018). Threat of adversarial attacks on deep learning in computer vision: A survey. IEEE Access, 6, 14410-14430. doi: 10.1109/ACCESS.2018.2806824

12. Carlini, N., & Wagner, D. (2017). Adversarial examples are not easily detected: Bypassing ten detection methods. In Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security (pp. 3-14). doi: 10.1145/3128572.3140444

13. Goodfellow, I., Shlens, J., & Szegedy, C. (2015). Explaining and harnessing adversarial examples. In Proceedings of the International Conference on Learning Representations.

14. Kurakin, A., Goodfellow, I., & Bengio, S. (2016). Adversarial examples in the physical world. arXiv preprint arXiv:1607.

15. Lu, J., Sibiryakov, A., & Fabian, T. (2017). Adversarial examples for semantic image segmentation. In Proceedings of the IEEE International Conference on Computer Vision (pp. 1378-1387).

16. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., & Vladu, A. (2018). Towards deep learning models resistant to adversarial attacks. arXiv preprint arXiv:1706.06083.

17. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., & Swami, A. (2016). The limitations of deep learning in adversarial settings. In Security and Privacy (EuroS&P), 2016 IEEE European Symposium on (pp. 372-387). IEEE.

18. Xiao, C., Li, B., Zhu, J., He, W., Liu, M., & Song, D. (2018). Generating adversarial examples with adversarial networks. ACM SIGSAC Conference on Computer and Communications Security.

19. Xu, W., Evans, D., & Qi, Y. (2018). Feature squeezing: Detecting adversarial examples in deep neural networks. Network and Distributed System

20. Li, X., Chen, T., & Yang, J. (2019). Adversarial fingerprint attacks and defenses. IEEE Transactions on Information Forensics and Security, 14(1), 66-80.
21. Nguyen, T. M., Kim, K. H., Lee, S., & Kim, J. (2019). Generative adversarial network-based face presentation attack detection using partial convolution and multi-domain learning. IEEE Transactions on Information Forensics and Security, 14(10), 2764-2779.
22. Tan, H., Li, H., Liu, Z., & Jiang, X. (2019). Deep learning based liveness detection: A survey. ACM Computing Surveys (CSUR), 52(3), 1-27.
23. Wang, Y., Kang, L., Li, Y., & Li, X. (2019). Fingerprint presentation attack detection using convolutional neural network with transfer learning. IEEE Access, 7, 131443-131451.
24. Face Antispoofing in_Biometric Systems. [Online] Available: https://www.researchgate.net/publication/311895447_Face_Antispoofing_in_Biometric_Systems
25. A website for interacting with technology ChatGPT. [Online] Available: https://chat.openai.com/
26. 26. The official website of the NumPy Library. [Online] Available: https://numpy.org
27. Example of neural network implementation. [Online] Available: https://webtort.ru
28. What are the layers of neural networks and how do they work. [Online] Available: https://habr.com/ru/articles/542386/
29. Interpreting Deep Neural Networks with SVCCA. [Online] Available: https://arxiv.org/abs/1706.05806