БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Международный Институт Дистанционного Образования

Кафедра “Информационные системы и технологии”

Контрольная работа по дисциплине «Методы и алгоритмы компьютерной графики»

Выполнил:
студент 3 курса, гр. 41703120
Реут Владислав Леонидович

Проверил: Анцыпов Н.А.

Минск 2023

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Нейронные сети (или искусственные нейронные сети) - это математическая модель, а также ее программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей - сетей нервных клеток живого организма. Они представляют собой систему соединенных и взаимодействующих между собой простых процессоров (искусственных нейронов), каждый из которых имеет дело только с сигналами, которые он периодически получает, и сигналами, которые он отправляет другим процессорам. Нейронные сети используются в искусственном интеллекте для обработки данных таким же способом, как и человеческий мозг. Они могут использоваться для решения различных задач, таких как прогнозирование, распознавание образов, управление и другие. Обучение нейронных сетей происходит путем первичной обработки больших наборов размеченных или неразмеченных данных.

Задание

Написать программу способную распознать рукописный ввод на примере цифр.

Описание программы

Для решения данной задачи была написана примитивная нейронная сеть с применением языка программирования C++. Обучена сеть при помощи доступных в сети данных (MNIST). Так же для взаимодействия с пользователем, с использованием фреймворка Qt, был разработан интерфейс для ввода символов.

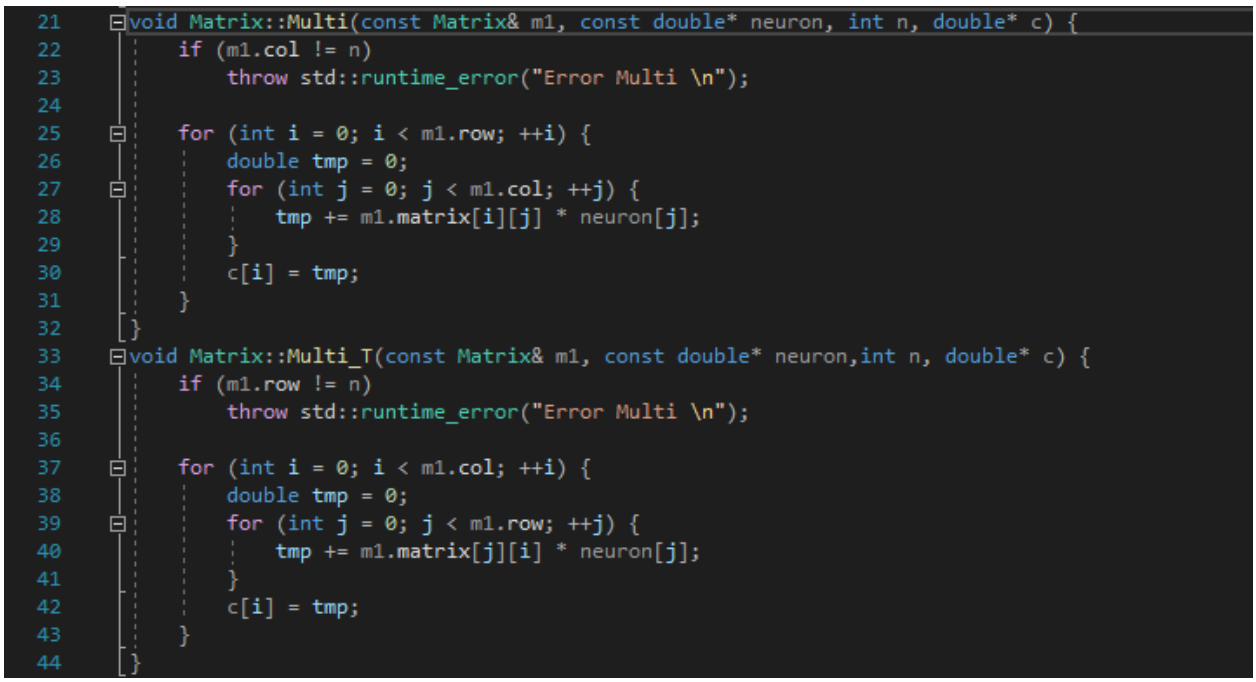
```
1   #pragma once
2   #include <iostream>
3   class Matrix
4   {
5       double** matrix;
6       int row, col;
7   public:
8       void Init(int row, int col);
9       void Rand();
10      static void Multi(const Matrix& m,  const double* b, int n, double* c);
11      static void Multi_T(const Matrix& m, const double* b, int n, double* c);
12      static void SumVector(double* a, const double* b, int n);
13      double& operator ()(int i, int j);
14      friend std::ostream& operator << (std::ostream& os, const Matrix& m);
15      friend std::istream& operator >> (std::istream& is, Matrix& m);
16  };
17
18
```

Рис.1 ***Matrix.h.***

```
1   #include "Matrix.h"
2   void Matrix::Init(int row, int col) {
3       this->row = row; this->col = col;
4       matrix = new double* [row];
5       for (int i = 0; i < row; i++)
6           matrix[i] = new double[col];
7
8       for (int i = 0; i < row; i++) {
9           for (int j = 0; j < col; j++) {
10              matrix[i][j] = 0;
11          }
12      }
13  }
14  void Matrix::Rand() {
15      for (int i = 0; i < row; i++) {
16          for (int j = 0; j < col; j++) {
17              matrix[i][j] = ((rand() % 100)) * 0.03 / (row + 35);
18          }
19      }
20  }
```

Рис.2 ***Matrix.cpp*** *функции* ***Init*** *и* ***Rand****.*

На Рис.2 находятся функции инициализации и заполнения матриц.



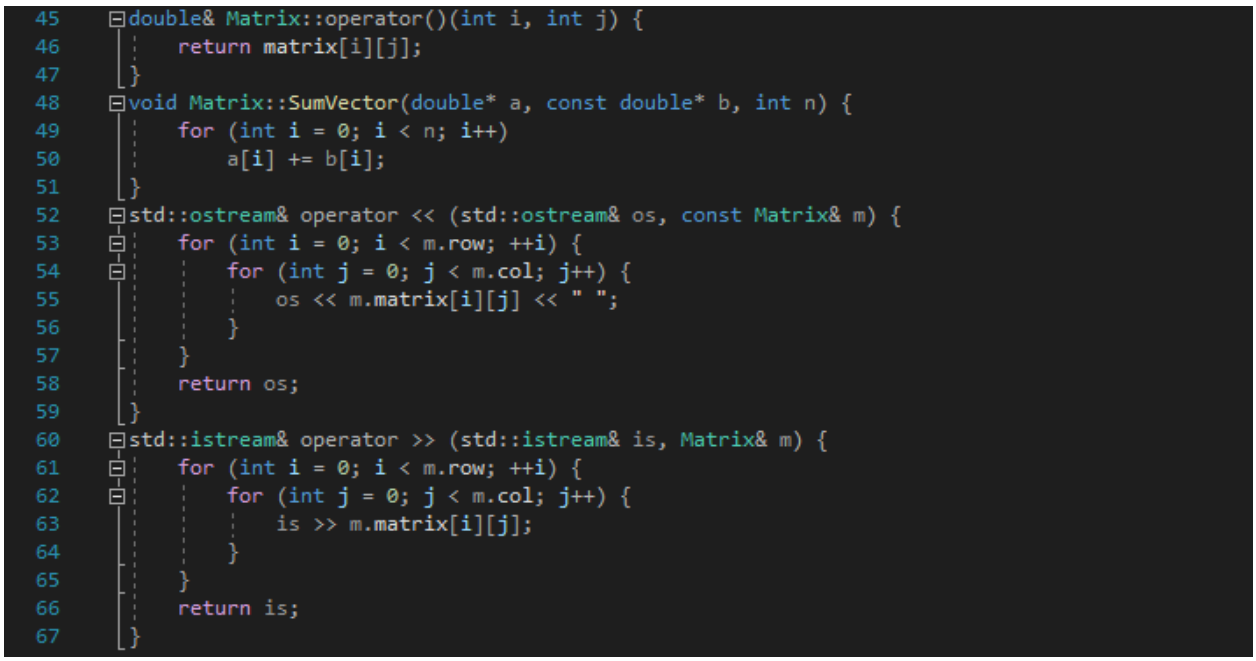
```
void Matrix::Multi(const Matrix& m1, const double* neuron, int n, double* c) {
    if (m1.col != n)
        throw std::runtime_error("Error Multi \n");

    for (int i = 0; i < m1.row; ++i) {
        double tmp = 0;
        for (int j = 0; j < m1.col; ++j) {
            tmp += m1.matrix[i][j] * neuron[j];
        }
        c[i] = tmp;
    }
}
void Matrix::Multi_T(const Matrix& m1, const double* neuron,int n, double* c) {
    if (m1.row != n)
        throw std::runtime_error("Error Multi \n");

    for (int i = 0; i < m1.col; ++i) {
        double tmp = 0;
        for (int j = 0; j < m1.row; ++j) {
            tmp += m1.matrix[j][i] * neuron[j];
        }
        c[i] = tmp;
    }
}
```

Рис.3 ***Matrix.cpp*** *функции* ***Multi*** *и* ***Multi_t****.*

Рис.3 содержит функции перемножения матриц (обычной и транспонированной).



```
double& Matrix::operator()(int i, int j) {
    return matrix[i][j];
}
void Matrix::SumVector(double* a, const double* b, int n) {
    for (int i = 0; i < n; i++)
        a[i] += b[i];
}
std::ostream& operator << (std::ostream& os, const Matrix& m) {
    for (int i = 0; i < m.row; ++i) {
        for (int j = 0; j < m.col; j++) {
            os << m.matrix[i][j] << " ";
        }
    }
    return os;
}
std::istream& operator >> (std::istream& is, Matrix& m) {
    for (int i = 0; i < m.row; ++i) {
        for (int j = 0; j < m.col; j++) {
            is >> m.matrix[i][j];
        }
    }
    return is;
}
```

Рис.4 ***Matrix.cpp*** *функция* ***SumVector*** *и перегруженные операторы*.

На Рис.4 можно увидеть функцию **SumVector**, которая предназначена для сложения векторов.

```h
#pragma once
#include "ActivateFunction.h"
#include "Matrix.h"
#include <fstream>
using namespace std;
struct data_NetWork {
    int L;
    int* size;
};
class NetWork
{
    int L;
    int* size;
    ActivateFunction actFunc;
    Matrix* weights;
    double** bios;
    double** neurons_val, ** neurons_err;
    double* neurons_bios_val;
public:
    void Init(data_NetWork data);
    void PrintConfig();
    void SetInput(double* values);

    double ForwardFeed();
    int SearchMaxIndex(double* value);
    void PrintValues(int L);

    void BackPropogation(double expect);
    void WeightsUpdater(double lr);

    void SaveWeights();
    void ReadWeights();
};


```

Рис.5 ***NetWork.h.***

```cpp
void NetWork::PrintConfig() {
    cout << "************************************************************\n";
    cout << "NetWork has " << L << " layers\nSIZE[]: ";
    for (int i = 0; i < L; i++) {
        cout << size[i] << " ";
    }
    cout << "\n************************************************************\n\n";
}
void NetWork::SetInput(double* values) {
    for (int i = 0; i < size[0]; i++) {
        neurons_val[0][i] = values[i];
    }
}
```

Рис.6 ***NetWork.cpp****. функции* ***PrintConfig*** *и* ***SetInput****.*

На Рис.6 изображены функция вывода в консоль конфигурации создаваемой сети и функция задания входных значений.

```cpp
#include "NetWork.h"
void NetWork::Init(data_NetWork data) {
    actFunc.set();
    srand(time(NULL));
    L = data.L;
    size = new int[L];
    for (int i = 0; i < L; i++)
        size[i] = data.size[i];

    weights = new Matrix[L - 1];
    bios = new double* [L - 1];
    for (int i = 0; i < L - 1; i++) {
        weights[i].Init(size[i + 1], size[i]);
        bios[i] = new double[size[i + 1]];
        weights[i].Rand();
        for (int j = 0; j < size[i + 1]; j++) {
            bios[i][j] = ((rand() % 50)) * 0.06 / (size[i] + 15);
        }
    }
    neurons_val = new double* [L]; neurons_err = new double* [L];
    for (int i = 0; i < L; i++) {
        neurons_val[i] = new double[size[i]]; neurons_err[i] = new double[size[i]];
    }
    neurons_bios_val = new double[L - 1];
    for (int i = 0; i < L - 1; i++)
        neurons_bios_val[i] = 1;
}
```

Рис.7 ***NetWork.cpp****. функция* ***Init.***

Рис.7 демонстрирует функцию **Init**, это функция инициализации сети. В ней происходит выбор функции активации (об этом ниже) и заполнение всех необходимых матриц.

```cpp
double NetWork::ForwardFeed() {
    for (int k = 1; k < L; ++k) {
        Matrix::Multi(weights[k - 1], neurons_val[k - 1], size[k - 1], neurons_val[k]);
        Matrix::SumVector(neurons_val[k], bios[k - 1], size[k]);
        actFunc.use(neurons_val[k], size[k]);
    }
    int pred = SearchMaxIndex(neurons_val[L - 1]);
    return pred;
}
int NetWork::SearchMaxIndex(double* value) {
    double max = value[0];
    int prediction = 0;
    double tmp;
    for (int j = 1; j < size[L - 1]; j++) {
        tmp = value[j];
        if (tmp > max) {
            prediction = j;
            max = tmp;
        }
    }
    return prediction;
}
```

Рис.8 ***NetWork.cpp****. функции* ***ForwardFeed*** *и* ***SearchMaxIndex****.*

Рис.8 демонстрирует функцию **ForwardFeed** – функция прямого распространения сети (двигаемся от входных нейронов через скрытые к выходным). Там же находится и функция **SearchMaxIndex** которая возвращает индекс элемента с максимальным значением.

```
70  void NetWork::BackPropogation(double expect) {
71      for (int i = 0; i < size[L - 1]; i++) {
72          if (i != int(expect))
73              neurons_err[L - 1][i] = -neurons_val[L - 1][i] * actFunc.useDer(neurons_val[L - 1][i]);
74          else
75              neurons_err[L - 1][i] = (1.0 - neurons_val[L - 1][i]) * actFunc.useDer(neurons_val[L - 1][i]);
76      }
77      for (int k = L - 2; k > 0; k--) {
78          Matrix::Multi_T(weights[k], neurons_err[k + 1], size[k + 1], neurons_err[k]);
79          for (int j = 0; j < size[k]; j++)
80              neurons_err[k][j] *= actFunc.useDer(neurons_val[k][j]);
81      }
82  }
83  void NetWork::WeightsUpdater(double lr) {
84      for (int i = 0; i < L - 1; ++i) {
85          for (int j = 0; j < size[i + 1]; ++j) {
86              for (int k = 0; k < size[i]; ++k) {
87                  weights[i](j, k) += neurons_val[i][k] * neurons_err[i + 1][j] * lr;
88              }
89          }
90      }
91      for (int i = 0; i < L - 1; i++) {
92          for (int k = 0; k < size[i + 1]; k++) {
93              bios[i][k] += neurons_err[i + 1][k] * lr;
94          }
95      }
96  }
```

Рис.9 ***NetWork.cpp****. функции* ***BackPropogation*** *и* ***WeightsUpdater****.*

```
97  void NetWork::SaveWeights() {
98      ofstream fout;
99      fout.open("Weights.txt");
100     if (!fout.is_open()) {
101         cout << "Error reading the file";
102         system("pause");
103     }
104     for (int i = 0; i < L - 1; ++i)
105         fout << weights[i] << " ";
106
107     for (int i = 0; i < L - 1; ++i) {
108         for (int j = 0; j < size[i + 1]; ++j) {
109             fout << bios[i][j] << " ";
110         }
111     }
112     cout << "Weights saved \n";
113     fout.close();
114 }
115 void NetWork::ReadWeights() {
116     ifstream fin;
117     fin.open("Weights.txt");
118     if (!fin.is_open()) {
119         cout << "Error reading the file";
120         system("pause");
121     }
122     for (int i = 0; i < L - 1; ++i) {
123         fin >> weights[i];
124     }
125     for (int i = 0; i < L - 1; ++i) {
126         for (int j = 0; j < size[i + 1]; ++j) {
127             fin >> bios[i][j];
128         }
129     }
130     cout << "Weights readed \n";
131     fin.close();
132 }
```

Рис.10 ***NetWork.cpp****. функции* ***SaveWeights*** *и* ***ReadWeights****.*

На Рис.9 представлены функции **BackPropogation** и **WeightsUpdater**. Первая получает на вход правильное значение и высчитывает ошибку необходимую для обучения сети. Вторая отвечает за обновление весов.

На Рис.10 изображены функции сохранения весов в файл, а так же чтение их же из файла.

```
1   #pragma once
2   #include <iostream>
3   enum activateFunc { sigmoid = 1, ReLU, thx };
4   class ActivateFunction
5   {
6       activateFunc actFunc;
7   public:
8       void set();
9       void use(double* value, int n);
10      void useDer(double* value, int n);
11      double useDer(double value);
12  };
```

Рис.11 ***ActivateFunction.h.***

```
1   #include "ActivateFunction.h"
2   void ActivateFunction::set() {
3       std::cout << "Set actFunc pls\n1 - sigmoid \n2 - ReLU \n3 - th(x) \n";
4       int tmp;
5       std::cin >> tmp;
6       switch (tmp)
7       {
8       case sigmoid:
9           actFunc = sigmoid;
10          break;
11      case ReLU:
12          actFunc = ReLU;
13          break;
14      case thx:
15          actFunc = thx;
16          break;
17      default:
18          throw std::runtime_error("Error read actFunc");
19          break;
20      }
21  }
```

Рис.12 ***ActivateFunction.cpp*** *функция* ***Set.***

Рис.12 содержит функцию выбора функции активации. В этой работе их всего три.

```cpp
void ActivateFunction::use(double* value, int n) {
    switch (actFunc)
    {
    case activateFunc::sigmoid:
        for (int i = 0; i < n; i++)
            value[i] = 1 / (1 + exp(-value[i]));
        break;
    case activateFunc::ReLU:
        for (int i = 0; i < n; i++) {
            if (value[i] < 0)
                value[i] *= 0.01;
            else if (value[i] > 1)
                value[i] = 1. + 0.01 * (value[i] - 1.);
            //else value = value;
        }
        break;

    case activateFunc::thx:
        for (int i = 0; i < n; i++) {
            if (value[i] < 0)
                value[i] = 0.01 * (exp(value[i]) - exp(-value[i])) / (exp(value[i]) + exp(-value[i]));
            else
                value[i] = (exp(value[i]) - exp(-value[i])) / (exp(value[i]) + exp(-value[i]));
        }
        break;
    default:
        throw std::runtime_error("Error actFunc \n");
        break;
    }
}
```

Рис.13 ***ActivateFunction.cpp*** *функция* ***Use***.

Функция (Рис.13) содержит в себе реализации самих функций активации.

```cpp
void ActivateFunction::useDer(double* value, int n) {
    switch (actFunc)
    {
    case activateFunc::sigmoid:
        for (int i = 0; i < n; i++)
            value[i] = value[i] * (1 - value[i]);
        break;
    case activateFunc::ReLU:
        for (int i = 0; i < n; i++) {
            if (value[i] < 0 || value[i] > 1)
                value[i] = 0.01;
            else
                value[i] = 1;
        }
        break;
    case activateFunc::thx:
        for (int i = 0; i < n; i++) {
            if (value[i] < 0)
                value[i] = 0.01 * (1 - value[i] * value[i]);
            else
                value[i] = 1 - value[i] * value[i];
        }
        break;
    default:
        throw std::runtime_error("Error actFuncDer \n");
        break;
    }
}
```

Рис.14 ***ActivateFunction.cpp*** *функция* ***useDer***.

```
81  double ActivateFunction::useDer(double value) {
82      switch (actFunc)
83      {
84      case activateFunc::sigmoid:
85          value = 1 / (1 + exp(-value));
86          break;
87      case activateFunc::ReLU:
88          if (value < 0 || value > 1)
89              value = 0.01;
90          break;
91      case activateFunc::thx:
92          if (value < 0)
93              value = 0.01 * (exp(value) - exp(-value)) / (exp(value) + exp(-value));
94          else
95              value = (exp(value) - exp(-value)) / (exp(value) + exp(-value));
96          break;
97
98      default:
99          throw std::runtime_error("Error actFunc \n");
100         break;
101     }
102     return value;
103 }
```

Рис.15 ***ActivateFunction.cpp*** *перегрузка функции* ***useDer***

Рис.14 содержит функцию **useDer** – производные функций активаций. А на Рис.15 изображена ее перегрузка.

```
1   #include "NetWork.h"
2   #include <chrono>
3
4   struct data_info {
5       double* pixels;
6       int digit;
7   };
8   data_NetWork ReadDataNetWork(string path) {
9       data_NetWork data{};
10      ifstream fin;
11      fin.open(path);
12      if (!fin.is_open()) {
13          cout << "Error reading the file " << path << endl;
14          system("pause");
15      }
16      else
17          cout << path << " loading...\n";
18      string tmp;
19      int L;
20      while (!fin.eof()) {
21          fin >> tmp;
22          if (tmp == "NetWork") {
23              fin >> L;
24              data.L = L;
25              data.size = new int[L];
26              for (int i = 0; i < L; i++) {
27                  fin >> data.size[i];
28              }
29          }
30      }
31      fin.close();
32      return data;
33  }
```

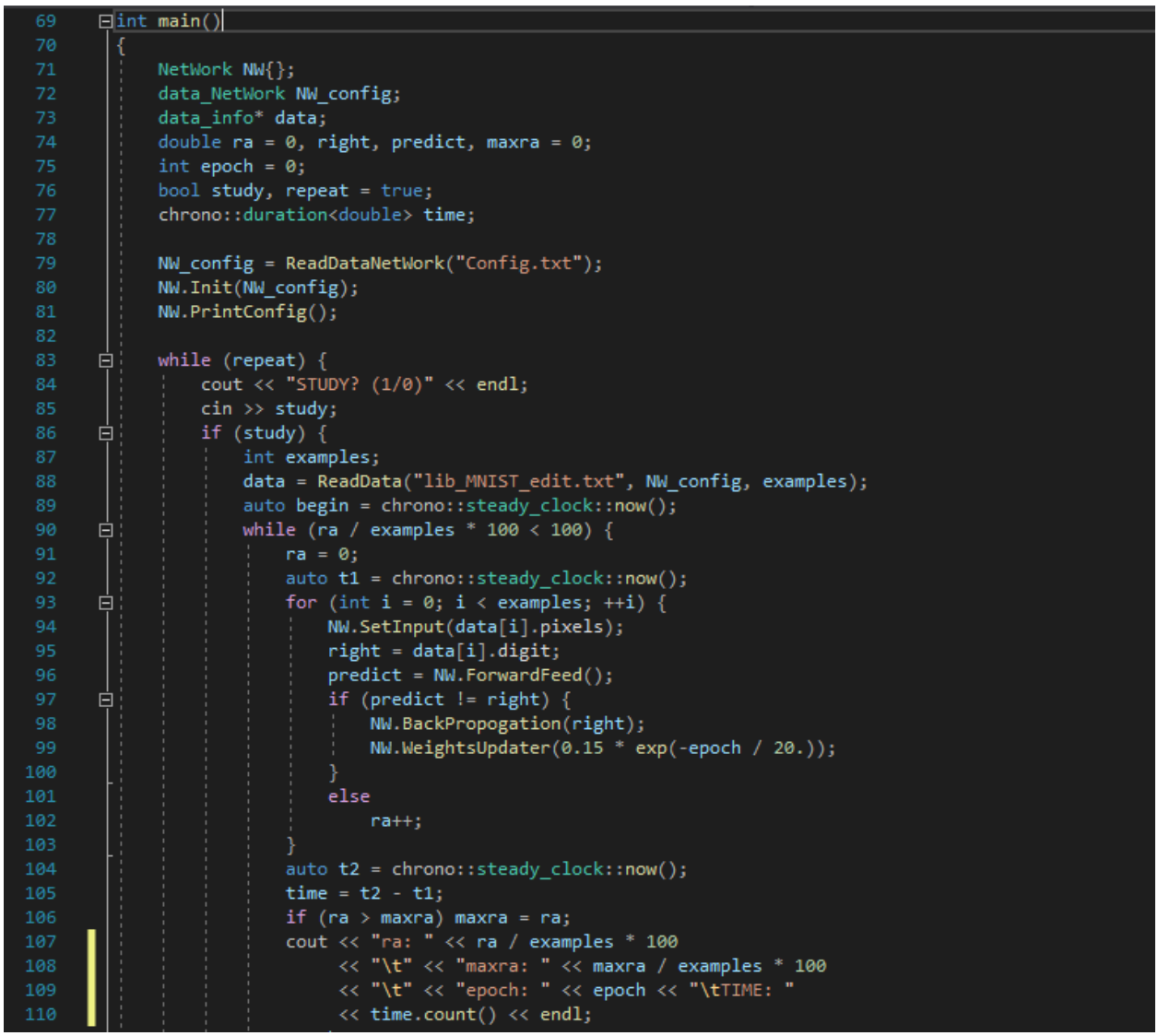
Рис.16 ***Source.cpp*** *структура* ***data_info*** *и функция* ***ReadDataNetWork***.

data_info (Рис.16) содержит поля для цифр и пикселей. **ReadDataNetWork** считывает с файла данные о количестве слоев и нейронов в ней. В данном случае это 3 слоя 784 – 264 – 10 нейронов.

```
34  data_info* ReadData(string path, const data_NetWork& data_NW, int &examples) {
35      data_info* data;
36      ifstream fin;
37      fin.open(path);
38      if (!fin.is_open()) {
39          cout << "Error reading the file " << path << endl;
40          system("pause");
41      }
42      else
43          cout << path << " loading... \n";
44      string tmp;
45      fin >> tmp;
46      if (tmp == "Examples") {
47          fin >> examples;
48          cout << "Examples: " << examples << endl;
49          data = new data_info[examples];
50          for (int i = 0; i < examples; ++i)
51              data[i].pixels = new double[data_NW.size[0]];
52
53          for (int i = 0; i < examples; ++i) {
54              fin >> data[i].digit;
55              for (int j = 0; j < data_NW.size[0]; ++j) {
56                  fin >> data[i].pixels[j];
57              }
58          }
59          fin.close();
60          cout << "lib_MNIST loaded... \n";
61          return data;
62      }
63      else {
64          cout << "Error loading: " << path << endl;
65          fin.close();
66          return nullptr;
67      }
68  }
```

Рис.17 ***Source.cpp*** *функция* ***ReadData.***

ReadData (Рис.17) – считываем данные с датасета.



```
int main()
{
    NetWork NW{};
    data_NetWork NW_config;
    data_info* data;
    double ra = 0, right, predict, maxra = 0;
    int epoch = 0;
    bool study, repeat = true;
    chrono::duration<double> time;

    NW_config = ReadDataNetWork("Config.txt");
    NW.Init(NW_config);
    NW.PrintConfig();

    while (repeat) {
        cout << "STUDY? (1/0)" << endl;
        cin >> study;
        if (study) {
            int examples;
            data = ReadData("lib_MNIST_edit.txt", NW_config, examples);
            auto begin = chrono::steady_clock::now();
            while (ra / examples * 100 < 100) {
                ra = 0;
                auto t1 = chrono::steady_clock::now();
                for (int i = 0; i < examples; ++i) {
                    NW.SetInput(data[i].pixels);
                    right = data[i].digit;
                    predict = NW.ForwardFeed();
                    if (predict != right) {
                        NW.BackPropogation(right);
                        NW.WeightsUpdater(0.15 * exp(-epoch / 20.));
                    }
                    else
                        ra++;
                }
                auto t2 = chrono::steady_clock::now();
                time = t2 - t1;
                if (ra > maxra) maxra = ra;
                cout << "ra: " << ra / examples * 100
                     << "\t" << "maxra: " << maxra / examples * 100
                     << "\t" << "epoch: " << epoch << "\tTIME: "
                     << time.count() << endl;
```

Рис.18 ***Source.cpp*** *функция* ***main*** *(начало).*

```
111                 epoch++;
112                 if (epoch == 20)
113                     break;
114             }
115             auto end = chrono::steady_clock::now();
116             time = end - begin;
117             cout << "TIME: " << time.count() / 60. << " min" << endl;
118             NW.SaveWeights();
119         }
120         else {
121             NW.ReadWeights();
122         }
123         cout << "Test? (1/0)\n";
124         bool test_flag;
125         cin >> test_flag;
126         if (test_flag) {
127             int ex_tests;
128             data_info* data_test;
129             data_test = ReadData("lib_10k.txt", NW_config, ex_tests);
130             ra = 0;
131             for (int i = 0; i < ex_tests; ++i) {
132                 NW.SetInput(data_test[i].pixels);
133                 predict = NW.ForwardFeed();
134                 right = data_test[i].digit;
135                 if (right == predict)
136                     ra++;
137             }
138             cout << "RA: " << ra / ex_tests * 100 << endl;
139         }
140         cout << "Repeat? (1/0)\n";
141         cin >> repeat;
142     }
143     system("pause");
144     return 0;
145 }
```

Рис.19 ***Source.cpp*** *функция* ***main*** *(окончание).*

На Рис.18-19 находится функция **main**, которая инициализирует все необходимые объекты а так же запускает основной цикл обучения.

Тестирование программы

```
D:\C++\Practice\Pract_Qt_without_Git\Neural_Network\NetWorkForVideoOne\Debug\NetWorkForVideoOne.exe
Config.txt loading...
Set actFunc pls
1 - sigmoid
2 - ReLU
3 - th(x)
2
************************************************************
NetWork has 3 layers
SIZE[]: 784 256 10
************************************************************

STUDY? (1/0)
1
lib_MNIST_edit.txt loading...
Examples: 60048
```

Рис.20 Процесс обучения сети (начало)

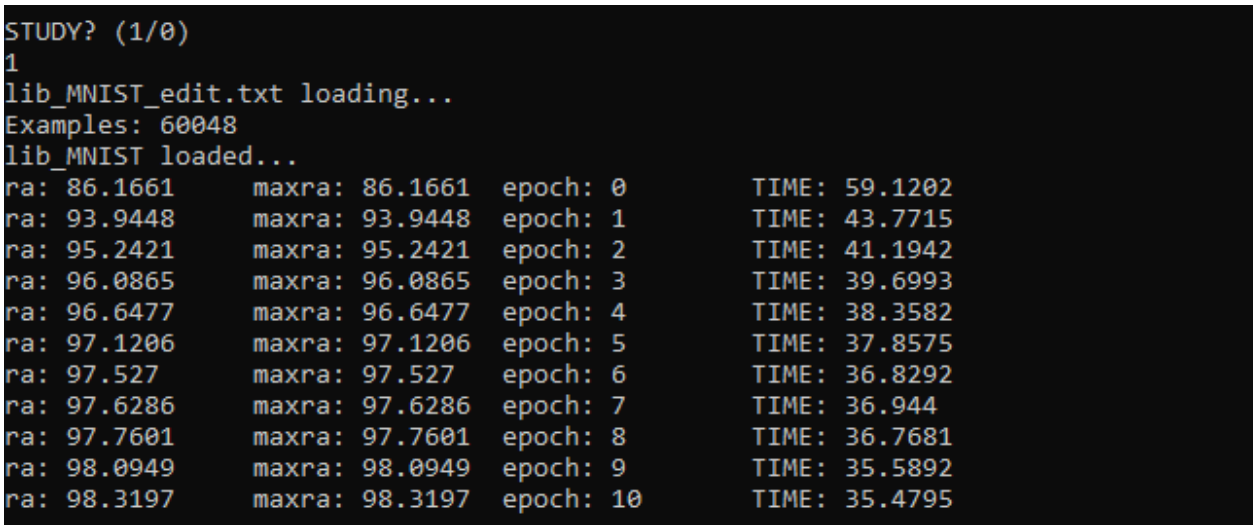


Рис.21 Процесс обучения сети (продолжение)

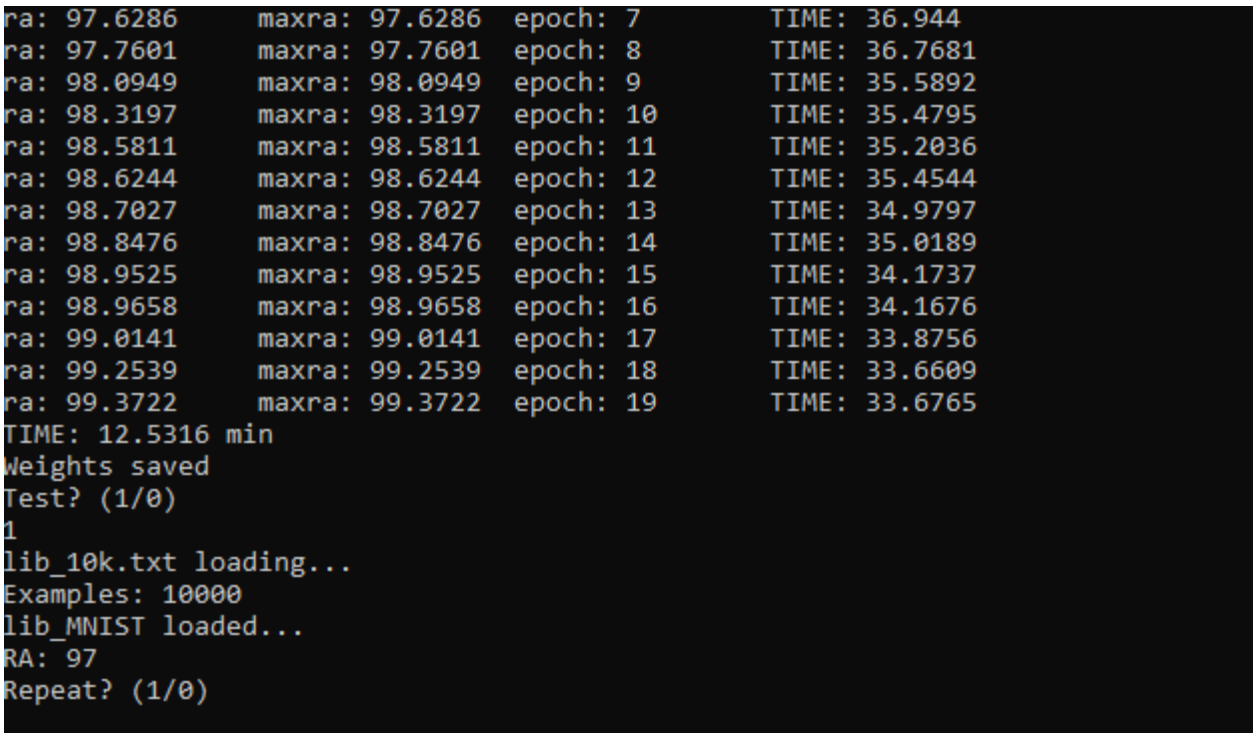
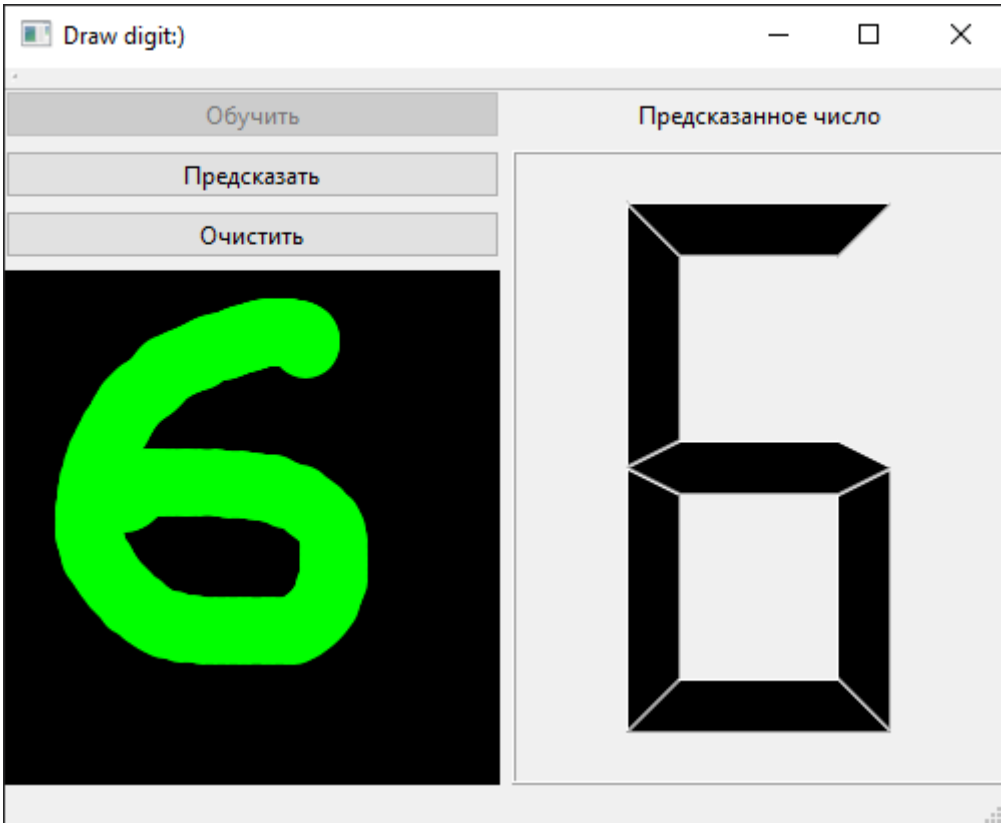


Рис.21 Процесс обучения сети (окончание)

ra – правильный ответ, maxra – максимально правильный ответ. Время обучения заняло 12 минут. Сеть обучилась распознавать значения в 97% случаев.

Теперь протестируем рукописный ввод:



Вывод

В результате разработки приложения были изучены принципы работы нейронных сетей, а так же оценена сложность и перспективность данной области.

Литература

1. Создаем нейронную сеть. Тарик Рашид.

Листинг программы

Matrix.h:

```
#pragma once
#include <iostream>
class Matrix
{
	double** matrix;
	int row, col;
public:
	void Init(int row, int col);
	void Rand();
	static void Multi(const Matrix& m, const double* b, int n, double* c);
	static void Multi_T(const Matrix& m, const double* b, int n, double* c);
	static void SumVector(double* a, const double* b, int n);
	double& operator ()(int i, int j);
	friend std::ostream& operator << (std::ostream& os, const Matrix& m);
	friend std::istream& operator >> (std::istream& is, Matrix& m);
};
```

Matrix.cpp:

```
#include "Matrix.h"
void Matrix::Init(int row, int col) {
	this->row = row; this->col = col;
	matrix = new double* [row];
	for (int i = 0; i < row; i++)
		matrix[i] = new double[col];

	for (int i = 0; i < row; i++) {
		for (int j = 0; j < col; j++) {
			matrix[i][j] = 0;
		}
	}
}
void Matrix::Rand() {
	for (int i = 0; i < row; i++) {
		for (int j = 0; j < col; j++) {
			matrix[i][j] = ((rand() % 100)) * 0.03 / (row + 35);
		}
	}
}
void Matrix::Multi(const Matrix& m1, const double* neuron, int n, double* c) {
	if (m1.col != n)
		throw std::runtime_error("Error Multi \n");

	for (int i = 0; i < m1.row; ++i) {
		double tmp = 0;
		for (int j = 0; j < m1.col; ++j) {
			tmp += m1.matrix[i][j] * neuron[j];
		}
		c[i] = tmp;
	}
}
void Matrix::Multi_T(const Matrix& m1, const double* neuron,int n, double* c) {
	if (m1.row != n)
		throw std::runtime_error("Error Multi \n");

	for (int i = 0; i < m1.col; ++i) {
		double tmp = 0;
		for (int j = 0; j < m1.row; ++j) {
```

```
                        tmp += m1.matrix[j][i] * neuron[j];
                }
                c[i] = tmp;
        }
}
double& Matrix::operator()(int i, int j) {
        return matrix[i][j];
}
void Matrix::SumVector(double* a, const double* b, int n) {
        for (int i = 0; i < n; i++)
                a[i] += b[i];
}
std::ostream& operator << (std::ostream& os, const Matrix& m) {
        for (int i = 0; i < m.row; ++i) {
                for (int j = 0; j < m.col; j++) {
                        os << m.matrix[i][j] << " ";
                }
        }
        return os;
}
std::istream& operator >> (std::istream& is, Matrix& m) {
        for (int i = 0; i < m.row; ++i) {
                for (int j = 0; j < m.col; j++) {
                        is >> m.matrix[i][j];
                }
        }
        return is;
}
```

ActivateFunction.h:

```
#pragma once
#include <iostream>
enum activateFunc { sigmoid = 1, ReLU, thx };
class ActivateFunction
{
        activateFunc actFunc;
public:
        void set();
        void use(double* value, int n);
        void useDer(double* value, int n);
        double useDer(double value);
};
```

ActivateFunction.cpp:

```
#include "ActivateFunction.h"
void ActivateFunction::set() {
        std::cout << "Set actFunc pls\n1 - sigmoid \n2 - ReLU \n3 - th(x) \n";
        int tmp;
        std::cin >> tmp;
        switch (tmp)
        {
        case sigmoid:
                actFunc = sigmoid;
                break;
        case ReLU:
                actFunc = ReLU;
                break;
        case thx:
                actFunc = thx;
```

```
			break;
		default:
			throw std::runtime_error("Error read actFunc");
			break;
		}
}
void ActivateFunction::use(double* value, int n) {
		switch (actFunc)
		{
		case activateFunc::sigmoid:
			for (int i = 0; i < n; i++)
				value[i] = 1 / (1 + exp(-value[i]));
			break;
		case activateFunc::ReLU:
			for (int i = 0; i < n; i++) {
				if (value[i] < 0)
					value[i] *= 0.01;
				else if (value[i] > 1)
					value[i] = 1. + 0.01 * (value[i] - 1.);
				//else value = value;
			}
			break;

		case activateFunc::thx:
			for (int i = 0; i < n; i++) {
				if (value[i] < 0)
					value[i] = 0.01 * (exp(value[i]) - exp(-value[i])) /
(exp(value[i]) + exp(-value[i]));
				else
					value[i] = (exp(value[i]) - exp(-value[i])) / (exp(value[i])
+ exp(-value[i]));
			}
			break;
		default:
			throw std::runtime_error("Error actFunc \n");
			break;
		}
}
void ActivateFunction::useDer(double* value, int n) {
		switch (actFunc)
		{
		case activateFunc::sigmoid:
			for (int i = 0; i < n; i++)
				value[i] = value[i] * (1 - value[i]);
			break;
		case activateFunc::ReLU:
			for (int i = 0; i < n; i++) {
				if (value[i] < 0 || value[i] > 1)
					value[i] = 0.01;
				else
					value[i] = 1;
			}
			break;
		case activateFunc::thx:
			for (int i = 0; i < n; i++) {
				if (value[i] < 0)
					value[i] = 0.01 * (1 - value[i] * value[i]);
				else
					value[i] = 1 - value[i] * value[i];
			}
			break;
		default:
			throw std::runtime_error("Error actFuncDer \n");
			break;
		}
```

```
}

double ActivateFunction::useDer(double value) {
	switch (actFunc)
	{
	case activateFunc::sigmoid:
		value = 1 / (1 + exp(-value));
		break;
	case activateFunc::ReLU:
		if (value < 0 || value > 1)
			value = 0.01;
		break;
	case activateFunc::thx:
		if (value < 0)
			value = 0.01 * (exp(value) - exp(-value)) / (exp(value) + exp(-
value));
		else
			value = (exp(value) - exp(-value)) / (exp(value) + exp(-value));
		break;

	default:
		throw std::runtime_error("Error actFunc \n");
		break;
	}
	return value;
}
```

NetWork.h:

```
#pragma once
#include "ActivateFunction.h"
#include "Matrix.h"
#include <fstream>
using namespace std;
struct data_NetWork {
	int L;
	int* size;
};
class NetWork
{
	int L;
	int* size;
	ActivateFunction actFunc;
	Matrix* weights;
	double** bios;
	double** neurons_val, ** neurons_err;
	double* neurons_bios_val;
public:
	void Init(data_NetWork data);
	void PrintConfig();
	void SetInput(double* values);

	double ForwardFeed();
	int SearchMaxIndex(double* value);
	void PrintValues(int L);

	void BackPropogation(double expect);
	void WeightsUpdater(double lr);

	void SaveWeights();
	void ReadWeights();
};
```

NetWork.cpp:

```
#include "NetWork.h"
void NetWork::Init(data_NetWork data) {
	actFunc.set();
	srand(time(NULL));
	L = data.L;
	size = new int[L];
	for (int i = 0; i < L; i++)
		size[i] = data.size[i];

	weights = new Matrix[L - 1];
	bios = new double* [L - 1];
	for (int i = 0; i < L - 1; i++) {
		weights[i].Init(size[i + 1], size[i]);
		bios[i] = new double[size[i + 1]];
		weights[i].Rand();
		for (int j = 0; j < size[i + 1]; j++) {
			bios[i][j] = ((rand() % 50)) * 0.06 / (size[i] + 15);
		}
	}
	neurons_val = new double* [L]; neurons_err = new double* [L];
	for (int i = 0; i < L; i++) {
		neurons_val[i] = new double[size[i]]; neurons_err[i] = new double[size[i]];
	}
	neurons_bios_val = new double[L - 1];
	for (int i = 0; i < L - 1; i++)
		neurons_bios_val[i] = 1;
}
void NetWork::PrintConfig() {
	cout << "************************************************************\n";
	cout << "NetWork has " << L << " layers\nSIZE[]: ";
	for (int i = 0; i < L; i++) {
		cout << size[i] << " ";
	}
	cout << "\n************************************************************\n\n";
}
void NetWork::SetInput(double* values) {
	for (int i = 0; i < size[0]; i++) {
		neurons_val[0][i] = values[i];
	}
}

double NetWork::ForwardFeed() {
	for (int k = 1; k < L; ++k) {
		Matrix::Multi(weights[k - 1], neurons_val[k - 1], size[k - 1],
neurons_val[k]);
		Matrix::SumVector(neurons_val[k], bios[k - 1], size[k]);
		actFunc.use(neurons_val[k], size[k]);
	}
	int pred = SearchMaxIndex(neurons_val[L - 1]);
	return pred;
}
int NetWork::SearchMaxIndex(double* value) {
	double max = value[0];
	int prediction = 0;
	double tmp;
	for (int j = 1; j < size[L - 1]; j++) {
		tmp = value[j];
		if (tmp > max) {
			prediction = j;
			max = tmp;
		}
	}
	return prediction;
```

```
}
void NetWork::PrintValues(int L) {
	for (int j = 0; j < size[L]; j++) {
		cout << j << " " << neurons_val[L][j] << endl;
	}
}

void NetWork::BackPropogation(double expect) {
	for (int i = 0; i < size[L - 1]; i++) {
		if (i != int(expect))
			neurons_err[L - 1][i] = -neurons_val[L - 1][i] *
actFunc.useDer(neurons_val[L - 1][i]);
		else
			neurons_err[L - 1][i] = (1.0 - neurons_val[L - 1][i]) *
actFunc.useDer(neurons_val[L - 1][i]);
	}
	for (int k = L - 2; k > 0; k--) {
		Matrix::Multi_T(weights[k], neurons_err[k + 1], size[k + 1],
neurons_err[k]);
		for (int j = 0; j < size[k]; j++)
			neurons_err[k][j] *= actFunc.useDer(neurons_val[k][j]);
	}
}
void NetWork::WeightsUpdater(double lr) {
	for (int i = 0; i < L - 1; ++i) {
		for (int j = 0; j < size[i + 1]; ++j) {
			for (int k = 0; k < size[i]; ++k) {
				weights[i](j, k) += neurons_val[i][k] * neurons_err[i + 1][j]
* lr;
			}
		}
	}
	for (int i = 0; i < L - 1; i++) {
		for (int k = 0; k < size[i + 1]; k++) {
			bios[i][k] += neurons_err[i + 1][k] * lr;
		}
	}
}
void NetWork::SaveWeights() {
	ofstream fout;
	fout.open("Weights.txt");
	if (!fout.is_open()) {
		cout << "Error reading the file";
		system("pause");
	}
	for (int i = 0; i < L - 1; ++i)
		fout << weights[i] << " ";

	for (int i = 0; i < L - 1; ++i) {
		for (int j = 0; j < size[i + 1]; ++j) {
			fout << bios[i][j] << " ";
		}
	}
	cout << "Weights saved \n";
	fout.close();
}
void NetWork::ReadWeights() {
	ifstream fin;
	fin.open("Weights.txt");
	if (!fin.is_open()) {
		cout << "Error reading the file";
		system("pause");
	}
	for (int i = 0; i < L - 1; ++i) {
		fin >> weights[i];
```

```
        }
        for (int i = 0; i < L - 1; ++i) {
               for (int j = 0; j < size[i + 1]; ++j) {
                      fin >> bios[i][j];
               }
        }
        cout << "Weights readed \n";
        fin.close();
}
```

Source.cpp:

```
#include "NetWork.h"
#include <chrono>

struct data_info {
    double* pixels;
    int digit;
};
data_NetWork ReadDataNetWork(string path) {
    data_NetWork data{};
    ifstream fin;
    fin.open(path);
    if (!fin.is_open()) {
        cout << "Error reading the file " << path << endl;
        system("pause");
    }
    else
        cout << path << " loading...\n";
    string tmp;
    int L;
    while (!fin.eof()) {
        fin >> tmp;
        if (tmp == "NetWork") {
            fin >> L;
            data.L = L;
            data.size = new int[L];
            for (int i = 0; i < L; i++) {
                fin >> data.size[i];
            }
        }
    }
    fin.close();
    return data;
}
data_info* ReadData(string path, const data_NetWork& data_NW, int &examples) {
    data_info* data;
    ifstream fin;
    fin.open(path);
    if (!fin.is_open()) {
        cout << "Error reading the file " << path << endl;
        system("pause");
    }
    else
        cout << path << " loading... \n";
    string tmp;
    fin >> tmp;
    if (tmp == "Examples") {
        fin >> examples;
        cout << "Examples: " << examples << endl;
        data = new data_info[examples];
        for (int i = 0; i < examples; ++i)
            data[i].pixels = new double[data_NW.size[0]];
```

```
        for (int i = 0; i < examples; ++i) {
            fin >> data[i].digit;
            for (int j = 0; j < data_NW.size[0]; ++j) {
                fin >> data[i].pixels[j];
            }
        }
        fin.close();
        cout << "lib_MNIST loaded... \n";
        return data;
    }
    else {
        cout << "Error loading: " << path << endl;
        fin.close();
        return nullptr;
    }
}
int main()
{
    NetWork NW{};
    data_NetWork NW_config;
    data_info* data;
    double ra = 0, right, predict, maxra = 0;
    int epoch = 0;
    bool study, repeat = true;
    chrono::duration<double> time;

    NW_config = ReadDataNetWork("Config.txt");
    NW.Init(NW_config);
    NW.PrintConfig();

    while (repeat) {
        cout << "STUDY? (1/0)" << endl;
        cin >> study;
        if (study) {
            int examples;
            data = ReadData("lib_MNIST_edit.txt", NW_config, examples);
            auto begin = chrono::steady_clock::now();
            while (ra / examples * 100 < 100) {
                ra = 0;
                auto t1 = chrono::steady_clock::now();
                for (int i = 0; i < examples; ++i) {
                    NW.SetInput(data[i].pixels);
                    right = data[i].digit;
                    predict = NW.ForwardFeed();
                    if (predict != right) {
                        NW.BackPropogation(right);
                        NW.WeightsUpdater(0.15 * exp(-epoch / 20.));
                    }
                    else
                        ra++;
                }
                auto t2 = chrono::steady_clock::now();
                time = t2 - t1;
                if (ra > maxra) maxra = ra;
                cout << "ra: " << ra / examples * 100
                     << "\t" << "maxra: " << maxra / examples * 100
                     << "\t" << "epoch: " << epoch << "\tTIME: "
                     << time.count() << endl;
                epoch++;
                if (epoch == 20)
                    break;
            }
            auto end = chrono::steady_clock::now();
            time = end - begin;
```

```
            cout << "TIME: " << time.count() / 60. << " min" << endl;
            NW.SaveWeights();
        }
        else {
            NW.ReadWeights();
        }
        cout << "Test? (1/0)\n";
        bool test_flag;
        cin >> test_flag;
        if (test_flag) {
            int ex_tests;
            data_info* data_test;
            data_test = ReadData("lib_10k.txt", NW_config, ex_tests);
            ra = 0;
            for (int i = 0; i < ex_tests; ++i) {
                NW.SetInput(data_test[i].pixels);
                predict = NW.ForwardFeed();
                right = data_test[i].digit;
                if (right == predict)
                    ra++;
            }
            cout << "RA: " << ra / ex_tests * 100 << endl;
        }
        cout << "Repeat? (1/0)\n";
        cin >> repeat;
    }
    system("pause");
    return 0;
}
```