|  |  |
| --- | --- |
| **Scenariusz 2** | |
| **Temat ćwiczenia:** | Budowa i działanie sieci jednowarstwowej |
| **Wykonał:** | Mateusz Mazur, gr 2 |

1. **Cel ćwiczenia**

Celem ćwiczenia jest poznanie budowy i działanie jednowarstwowych sieci neuronowych oraz uczenie rozpoznawania wielkości liter.

1. **Budowa sieci i wykorzystanych algorytmów uczenia**

Celem budowanej sieci jest rozpoznawanie wielkości liter w tym celu stworzony został zestaw liter (10 dużych i 10 małych), które są reprezentowane w postaci dwuwymiarowej tablicy 4x5 pikseli dla jednej litery. Użyte litery są przedstawione w poniższej tabeli.

|  |  |  |
| --- | --- | --- |
|  | 0110  1001  1111  1001  1001 | 1110  1010  1010  1010  1111 |
|  | 1110  1001  1110  1001  1110 | 1000  1000  1011  1010  1110 |
|  | 1110  1001  1001  1001  1110 | 0010  0010  1110  1010  1111 |
|  | 1111  1000  1111  1000  1111 | 0110  1001  1111  1000  0111 |
|  | 1111  1000  1110  1000  1000 | 0111  0101  0111  1100  0100 |
|  | 1111  1000  1011  1001  1111 | 1110  1011  1110  0110  1110 |
|  | 1001  1001  1111  1001  1001 | 1000  1000  1111  1001  1001 |
|  | 1111  0001  0011  0101  1111 | 0010  1111  0010  0110  1110 |
|  | 1000  1000  1000  1000  1111 | 1000  1000  1000  1001  0110 |
|  | 1111  1001  1111  1000  1000 | 0111  1101  0101  0100  0100 |

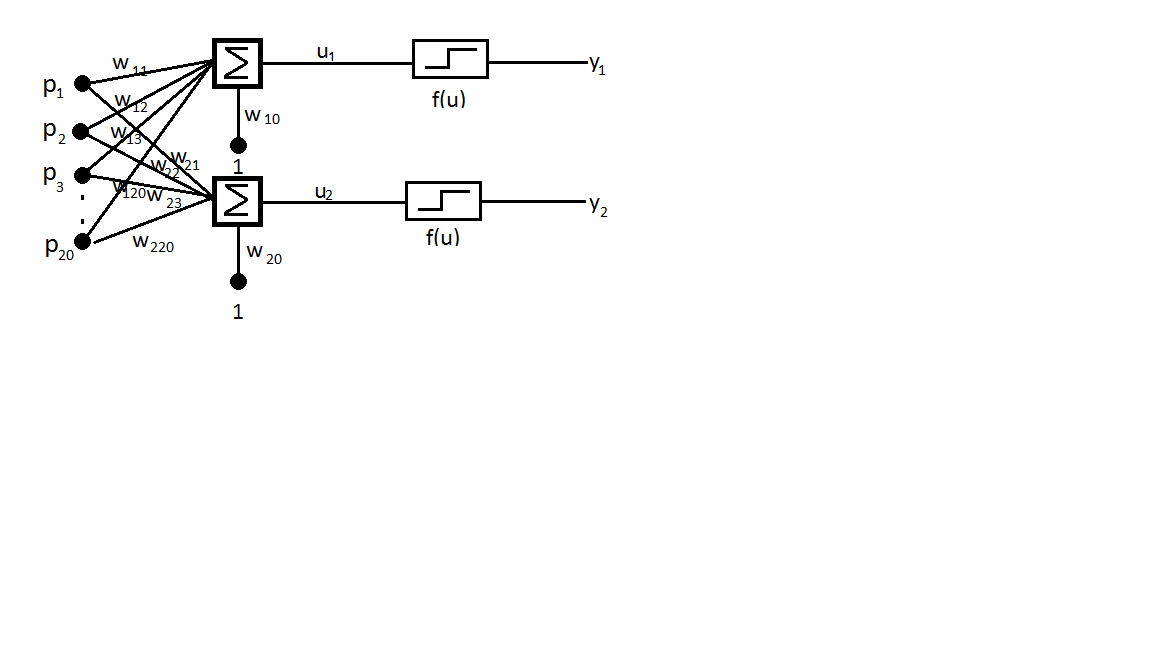
Tab. 1 Przygotowany alfabet

A więc ciąg uczący składa się z 20 wektorów uczących, natomiast mamy dwa wektory wyjściowe.

|  |  |
| --- | --- |
| **Wektor wejścia** | **Wektor wyjścia** |
| A, B, D, E, F, G, H, J, L, P | 1 0 |
| a, b, d, e, f, g, h, j, l, p | 0 1 |

Zostały wybrane sieci jednowarstwowe składające się z dwóch neuronów.

Do pierwszej z sieci została wykorzystany program przygotowany do poprzedniego zagadnienia. Schemat całej sieci został zamieszczony na poniższym rysunku.

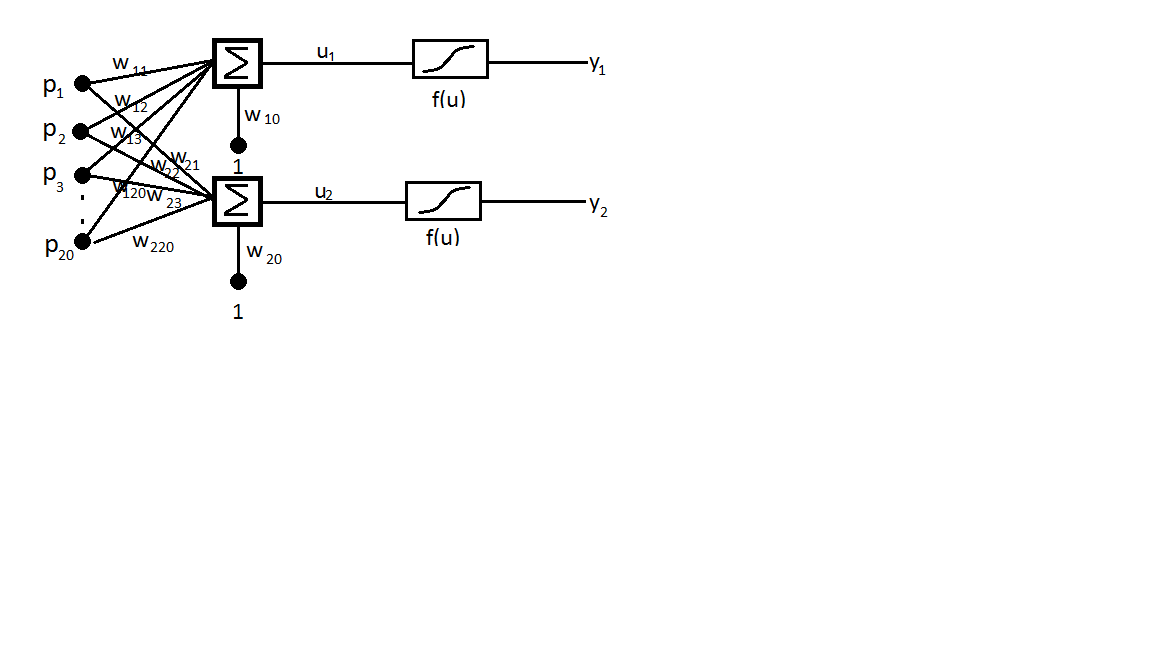


Rys. 1 Pierwsza sieć neuronowa

Sieć wykorzystuje unipolarną, binarną funkcję aktywacji, natomiast proces uczenia przebiega według następującego schematu:  
1. Na wstępie wagi dobrano w sposób losowy.  
2. Obliczana jest suma ilorazów sygnałów wejściowych i wag- u  
3. Obliczana jest wartość wyjściowa y za pomocą funkcji binarnej, unipolarnej funkcji aktywacji.

1. W wyniku porównania aktualnej wartości yi oraz wartości zadanej di dokonuje się aktualizacji wag. Jeżeli wartości otrzymana i zadana się nie różnią to wagi pozostają niezmienione. A więc zgodnie z tą regułą dobór wag odbywa się według zależności:

W drugiej ze stworzonych sieci wykorzystano nieliniową funkcję aktywacji. Jej schemat jest przedstawiony na poniższym rysunku.



Rys. 2 Druga sieć neuronowa

Sieć wykorzystuje unipolarną, logistyczną(sigmoidalną) funkcję aktywacji, natomiast proces uczenia przebiega według następującego schematu w danej epoce uczenia:  
1. Wybór > 0 oraz Emax > 0  
2. Wybór początkowych wartości wag, jako niewielkich liczb losowych  
3. Ustawienie sumy błędu kwadratowego na zero (E = 0)  
4. Obliczana jest suma ilorazów sygnałów wejściowych i wag- u  
5. Obliczana jest wartość wyjściowa y za pomocą unipolarnej, sigmoidalnej funkcji aktywacji według wzoru:

gdzie wybrano z zakresu (0, 1>  
6. Uaktualnienie wag za pomocą reguły delty według zależności:

7. Obliczenie łącznego błędu epoki:

8. Uczenie uważamy za zakończone, jeżeli E < Emax. W przeciwnym razie rozpoczynamy nową epokę uczenia i wracamy do punktu nr 3.

1. **Otrzymane wyniki**

Testowanie zaimplementowanej sieci neuronowej odbywało się przy stałej liczbie epok równej 50 dla obu metod uczenia się sieci neuronowej. Następnie dla każdego z przypadków uruchamiane były dane testujące w liczbie 20 składające się z 10 dużych liter alfabetu i 10 małych liter alfabetu. Czynnikiem różnicującym poszczególne fazy testów była wartość współczynnika uczenia. Efekty nauki sieci neuronowych przetestowano przy użyciu przygotowanych wcześniej danych testujących.

W poniższej tabeli zebrano dane z pracy programu, zawierające między innymi ilość poprawnych i błędnych odpowiedzi neuronu w fazie testowania, wartość oraz zmianę wartości błędu średniokwadratowego podczas procesu uczenia jak i podczas testów dla każdej wartości współczynnika uczenia jak i metody uczenia się sieci z osobna.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Wartość współczynnika uczenia | Liczba prawidłowych odpowiedzi podczas testów | | Liczba błędnych odpowiedzi podczas testów | | Procent poprawnych odpowiedzi | | Wartość błędu średniokwadratowego podczas testów | | Wartość początkowa błędu średniokwadratowego | | Wartość końcowa błędu średniokwadratowego | | Różnica błędu średniokwadratowego pomiędzy początkiem, a końcem uczenia | |
| Metoda 1 | Metoda 2 | Metoda 1 | Metoda 2 | Metoda 1 | Metoda 2 | Metoda 1 | Metoda 2 | Metoda 1 | Metoda 2 | Metoda 1 | Metoda 2 | Metoda 1 | Metoda 2 |
| 0,0009 | 21 | 21 | 19 | 19 | 52,5% | 52,5% | 9,5 | 10,5 | 12,5 | 7,33984 | 9,5 | 6,61027 | 3 | 0,72957 |
| 0,009 | 38 | 24 | 2 | 16 | 95,0% | 60,0% | 1 | 8 | 9 | 9,26902 | 1 | 5,38877 | 8 | 3,88025 |
| 0,09 | 40 | 37 | 0 | 3 | 100,0% | 92,5% | 0 | 1,5 | 12,5 | 6,22351 | 0 | 1,48951 | 12,5 | 4,734 |
| 0,1 | 40 | 39 | 0 | 1 | 100,0% | 97,5% | 0 | 0,5 | 14,5 | 7,41462 | 0 | 1,24948 | 14,5 | 6,16514 |
| 0,5 | 40 | 40 | 0 | 0 | 100,0% | 100,0% | 0 | 0 | 14,5 | 7,76511 | 0 | 0,34717 | 14,5 | 7,41793 |
| 1 | 40 | 40 | 0 | 0 | 100,0% | 100,0% | 0 | 0 | 14,5 | 8,44406 | 0 | 0,14868 | 14,5 | 8,29537 |

Tab. 2 Zestawienie wyników uzyskanych przed obie sieci neuronowe

1. **Analiza wyników**

Wykres nr. 1 Procentowa zależność poprawnych odpowiedzi od współczynnika uczenia

Na podstawie wykresu można stwierdzić, że wraz ze wzrostem wartości współczynnika uczenia procent dobrych odpowiedzi się zwiększa, aż do osiągnięcia szczytowej wartości równej 100% poprawności odpowiedzi. O ile dla niskiej wartości naszego współczynnika widzimy, iż obydwie metody uczenia osiągają podobne wyniki, to przy zwiększaniu naszego współczynnika zauważamy o wiele szybszy wzrost poprawnych odpowiedzi w przypadku sieci pierwszej, która to już przy wartości 0,009 jest w stanie osiągnąć bardzo wysoką skuteczność, której druga metoda jest w stanie dorównać jedynie w przypadku zwiększenia współczynnika uczenia dziesięciokrotnie. W momencie osiągnięcia przez współczynnik wartości 0,5 zauważamy, iż dla każdej z metod osiągnęliśmy 100% poprawność wskazywania właściwej odpowiedzi. Dalsze zwiększanie współczynnika uczenia nie przynosi już żadnych wymiernych rezultatów.

Wykres nr. 2 Zależność końcowej wartości błędu średniokwadratowego od wartości współczynnika uczenia podczas uczenia sieci

Na powyższym wykresie możemy obserwować zależność występującą podczas procesu uczenia dotyczącą wartości błędu średniokwadratowego w zależności od wartości współczynnika uczenia. Wartość błędu jest tym mniejsza im wartość współczynnika uczenia jest większa, co ma bezpośredni związek z procentowym stosunkiem prawidłowych odpowiedzi do wszystkich odpowiedzi. Niska wartość współczynnika wskazuje na mniejszą rozbieżność w odpowiedziach, margines błędu jest znacznie mniejszy, a w szczytowych przypadkach jest on równy 0, co oznacza 100% pewność, że odpowiedź neuronu będzie odpowiedzą prawidłową. Na podstawie powyższego wykresu możemy zauważyć, iż metoda 1 cechuje się znacznie niższymi wartościami błędu średniokwadratowego niż metoda 2. Już w przypadku wartości współczynnika uczenia równej 0,09 widzimy, iż dla metody pierwszej wartość ta wynosi 0, co oznacza 100% poprawność odpowiedzi sieci neuronowej. W przypadku drugiej metody uczenia sieci, możemy zauważyć dla nawet największej wartości współczynnika uczenia równej w tym przypadku 1, iż współczynnik ten nie jest równy 0, co pozwala stwierdzić, iż istnieje szansa na wystąpienie minimalnego błędu, jednak w naszym przypadku pomimo wielokrotnych testów sieć druga wykazywała zawsze 100% skuteczność dla tej wartości współczynnika uczenia.

Wykres nr. 3 Różnica pomiędzy początkową wartością błędu średniokwadratowego a wartością końcową w zależności od wartości współczynnika uczenia dla procesu uczenia sieci

Na powyższym wykresie możemy zaobserwować zależność pomiędzy wartością współczynnika uczenia oraz różnicą pomiędzy początkową wartością błędu średniokwadratowego, a końcową wartością tego błędu dla procesu uczenia. Widzimy, iż większa różnica występuję w przypadku wyższych współczynników uczenia, co jest związane z większą liczbą poprawnych odpowiedzi uzyskiwaną przez sieci. Większa różnica oznacza, że współczynnik o wiele szybciej dąży do wartości równej 0 dającej większa poprawność odpowiedzi. Tutaj podobnie jak wcześniej widzimy, iż sieć pierwsza osiąga lepsze rezultaty już przy niższych wartość współczynnika uczenia, jednak jednocześnie możemy też stwierdzić, iż sieć druga przy odpowiednio wielkiej wartości współczynnika uczenia jest w stanie dorównać pierwszej w poprawności odpowiedzi, cechując się jednocześnie o wiele mniejszym wahaniem błędu średniokwadratowego.

1. **Wnioski**

* Efektywność procesu uczenia dla każdego z algorytmów uczenia sieci jest silnie powiązana z wartością współczynnika uczenia sieci. Im wartość współczynnika jest większa tym sieć w mniejszej ilości epok jest w stanie osiągnąć 100% poprawność odpowiedzi. Jest to powiązane z faktem, iż dzięki wyższej wartości współczynnika uczenia wagi zmieniają się szybciej w związku z czym szybciej osiągają poprawną wartość.
* W przypadku drugiego algorytmu, bardziej złożonego zauważono, że przy etapie wyboru współczynnika β, dla którego to im wyższa jest jego wartość tym jego zachowanie bardziej przypomina działanie funkcji binarnej, że im jego wartość była niższa tym sieć działała gorzej, a przy jego wartościach równych 0.5 postęp uczenia był prawie niezauważalny. Szybkość uczenia takiej sieci jest niewielka, co może jednak być przydatne w bardziej zaawansowanych problemach, co pozwoli nie dopuścić do przeuczenia sieci.
* Porównując obie metody uczenia się naszych sieci możemy zauważyć, iż sieć pierwsza wyposażona w binarną funkcję aktywacji jest w stanie osiągnąć takie same lub lepsze wyniki przy danej wartości współczynnika uczenia w porównaniu z wynikami uzyskanymi przez drugi algorytm wykorzystujący funkcję sigmoidalną. Projektując więc swoją własną sieć musimy mieć na uwadze właściwy dobór algorytmu uczenia, tak aby sieć sprostała naszym wymaganiom. Nie zawsze bardziej skomplikowany algorytm będzie sprawował się lepiej jak to widać w tym przypadku.

1. **Kod programu**

**Neuron.h:**

#pragma once

#include <iostream>

#include <vector>

using namespace std;

class Neuron

{

public:

void createInputs(int amountOfDendrites, int amountOfSynapses);

void createDendrite() { \_dendrites.push\_back(0); }

void createSynapse(int index) { \_synapses[index].push\_back(0); }

int getDendritesAmount() { return \_dendrites.size(); }

int getSynapsesAmount() { return \_synapses.size(); }

double getInput(int index) { return \_dendrites[index]; }

void setInput(int index, double value) { \_dendrites[index] = value; }

double getSynapse(int index1, int index2) { return \_synapses[index1][index2]; }

void setSynapse(int synapseIndex, int index, double value) { \_synapses[synapseIndex][index] = value; }

double getSumOfAllInputs(int index) { return \_sumOfAllInputs[index]; }

double\* getOutputValue() { return \_outputValue; }

double getOutputValue(int index){ return \_outputValue[index]; }

double processSingleInput(int index, int synapseIndex) { return \_dendrites[index] \* \_synapses[synapseIndex][index]; }

void sumOfInputs();

void processOutput(bool secondAlgorithm, bool testing);

void countNewWeights(int rightAnswer, int neuronAnswer, int \*inputArray, int inputNumber, bool secondAlgorithm);

Neuron(int amountOfDendrites, int amountOfSynapses, double learningCoefficient)

{

createInputs(amountOfDendrites, amountOfSynapses);

\_learningCoefficient = learningCoefficient;

\_sumOfAllInputs = new double[amountOfSynapses];

\_outputValue = new double[amountOfSynapses];

}

private:

double countFirstWeight();

vector<double> \_dendrites; //inputs

vector< vector<double> > \_synapses; //weights

double \*\_sumOfAllInputs;

double \*\_outputValue;

double \_learningCoefficient;

int \_size;

};

**Neuron.cpp:**

#include "Neuron.h"

#include <time.h>

#include <math.h>

void Neuron::createInputs(int amountOfDendrites, int amountOfSynapses)

{

double weight = countFirstWeight();

vector<double> row;

for (int i = 0; i < amountOfSynapses; i++)

{

row.clear();

for (int j = 0; j < amountOfDendrites; j++)

{

if (i == 0)

{

\_dendrites.push\_back(0);

}

row.push\_back(countFirstWeight());

}

\_synapses.push\_back(row);

}

}

void Neuron::sumOfInputs()

{

double \*sum = new double[getSynapsesAmount()];

for (int i = 0; i < getSynapsesAmount(); i++)

{

sum[i] = 0;

for (int j = 0; j < getDendritesAmount(); j++)

{

sum[i] += processSingleInput(j, i);

}

\_sumOfAllInputs[i] = sum[i];

}

}

void Neuron::processOutput(bool secondAlgorithm, bool testing)

{

if (secondAlgorithm == false || testing == true)

{

for (int i = 0; i < getSynapsesAmount(); i++)

{

if (\_sumOfAllInputs[i] > 0)

{

\_outputValue[i] = 1;

}

else

{

\_outputValue[i] = 0;

}

}

}

else // second algorithm

{

double functionCoefficient = 1;

for (int i = 0; i < getSynapsesAmount(); i++)

{

\_outputValue[i] = (1.0 / (1.0 + (exp(-functionCoefficient \*   
 \_sumOfAllInputs[i]))));

}

}

}

void Neuron::countNewWeights(int rightAnswer, int neuronAnswer, int\* inputArray, int   
 inputNumber, bool secondAlgorithm)

{

if (secondAlgorithm == false)

{

\_synapses[inputNumber][0] += \_learningCoefficient \* (rightAnswer -   
 neuronAnswer);

for (int j = 1; j < getDendritesAmount(); j++)

{

\_synapses[inputNumber][j] += \_learningCoefficient \* (rightAnswer -   
 neuronAnswer) \* inputArray[j];

}

}

else // second algorithm

{

\_synapses[inputNumber][0] += \_learningCoefficient \* (rightAnswer -   
 \_outputValue[inputNumber]);

for (int j = 1; j < getDendritesAmount(); j++)

{

\_synapses[inputNumber][j] += \_learningCoefficient \* (rightAnswer -   
 \_outputValue[inputNumber] ) \* (\_outputValue[inputNumber] \* (1 -   
 \_outputValue[inputNumber])) \*inputArray[j];

}

}

}

//private methods

double Neuron::countFirstWeight()

{

double max = 1.0;

double min = -1.0;

double weight = ((double(rand()) / double(RAND\_MAX)) \* (max - min)) + min;

return weight;

}

**main.cpp:**

#include <iostream>

#include <vector>

#include "Neuron.h"

#include <time.h>

#include <fstream>

using namespace std;

void setInputValue(Neuron& neuron, bool testing, int numberOfEntrances, int   
 numberOfWeights, bool secondAlgorithm);

void learn(Neuron& neuron, bool testing, int numberOfEntrances, int numberOfWeights,   
 bool secondAlgorithm);

fstream OUTPUT\_LEARNING\_FILE;

fstream OUTPUT\_TESTING\_FILE;

fstream OUTPUT\_LEARNING\_SECOND\_ALGORITHM\_FILE;

fstream OUTPUT\_TESTING\_SECOND\_ALGORITHM\_FILE;

fstream LEARNING\_DATA;

fstream TESTING\_DATA;

double E\_F;

double E\_F\_TESTING;

double E;

double E\_TESTING;

int BAD\_ANSWERS\_AMOUNT;

int TOTAL\_BAD\_ANSWERS\_AMOUNT;

int RIGHT\_ANSWERS\_AMOUNT;

int TOTAL\_RIGHT\_ANSWERS\_AMOUNT;

int CORRECT\_ANSWER[2];

int main()

{

srand(time(NULL));

int numberOfEntrances = 21;

int numberOfWeights = 2;

Neuron neuron(numberOfEntrances, numberOfWeights, 1);

LEARNING\_DATA.open("learning\_data.txt", ios::in);

TESTING\_DATA.open("testing\_data.txt", ios::in);

OUTPUT\_LEARNING\_FILE.open("output\_learning\_data.txt", ios::out);

OUTPUT\_LEARNING\_SECOND\_ALGORITHM\_FILE.open("output\_learning\_second\_algorithm\_dat  
 a.txt", ios::out);

do

{

cout << "1. Learn" << endl;

cout << "2. Test" << endl;

cout << "3. Learn (second algorithm)" << endl;

cout << "4. Test (second algorithm)" << endl;

cout << "5. Exit" << endl;

int choice;

cin >> choice;

switch (choice)

{

case 1:

BAD\_ANSWERS\_AMOUNT = 0;

RIGHT\_ANSWERS\_AMOUNT = 0;

TOTAL\_BAD\_ANSWERS\_AMOUNT = 0;

TOTAL\_RIGHT\_ANSWERS\_AMOUNT = 0;

int numberOfEpochs;

cout << "Enter number of epochs: " << endl;

cin >> numberOfEpochs;

OUTPUT\_LEARNING\_FILE << "NEW EPOCH NUMBER: 1" << endl;

for (int i = 1, epochNumber = 1; i <= numberOfEpochs \* 20; i++)

{

learn(neuron, false, numberOfEntrances, numberOfWeights,   
 false);

if (i % 20 == 0)

{

epochNumber++;

LEARNING\_DATA.clear();

LEARNING\_DATA.seekg(0, ios::beg);

OUTPUT\_LEARNING\_FILE << "Bad answers amount: " <<   
 BAD\_ANSWERS\_AMOUNT << endl;

OUTPUT\_LEARNING\_FILE << "Right answers amount: " <<  
 RIGHT\_ANSWERS\_AMOUNT << endl;

OUTPUT\_LEARNING\_FILE << "EF: " << E\_F << endl;

E\_F = 0;

if (epochNumber <= numberOfEpochs)

{

OUTPUT\_LEARNING\_FILE << "NEW EPOCH NUMBER: " <<   
 epochNumber << endl;

}

TOTAL\_BAD\_ANSWERS\_AMOUNT += BAD\_ANSWERS\_AMOUNT;

TOTAL\_RIGHT\_ANSWERS\_AMOUNT += RIGHT\_ANSWERS\_AMOUNT;

RIGHT\_ANSWERS\_AMOUNT = 0;

BAD\_ANSWERS\_AMOUNT = 0;

}

}

OUTPUT\_LEARNING\_FILE << "\nTotal bad answers amount: " <<  
 TOTAL\_BAD\_ANSWERS\_AMOUNT << endl;

OUTPUT\_LEARNING\_FILE << "Total right answers amount: " <<   
 TOTAL\_RIGHT\_ANSWERS\_AMOUNT << endl;

break;

case 2:

OUTPUT\_TESTING\_FILE.open("output\_testing\_data.txt", ios::out);

BAD\_ANSWERS\_AMOUNT = 0;

RIGHT\_ANSWERS\_AMOUNT = 0;

E\_F\_TESTING = 0;

for (int i = 0; i < 20; i++)

{

learn(neuron, true, numberOfEntrances, numberOfWeights,  
 false);

}

OUTPUT\_TESTING\_FILE << "Bad answers amount: " << BAD\_ANSWERS\_AMOUNT  
 << endl;

OUTPUT\_TESTING\_FILE << "Right answers amount: " <<   
 RIGHT\_ANSWERS\_AMOUNT << endl;

OUTPUT\_TESTING\_FILE << "E: " << E\_F\_TESTING << endl;

break;

case 3:

BAD\_ANSWERS\_AMOUNT = 0;

RIGHT\_ANSWERS\_AMOUNT = 0;

TOTAL\_BAD\_ANSWERS\_AMOUNT = 0;

TOTAL\_RIGHT\_ANSWERS\_AMOUNT = 0;

cout << "Enter number of epochs: " << endl;

cin >> numberOfEpochs;

OUTPUT\_LEARNING\_SECOND\_ALGORITHM\_FILE << "NEW EPOCH NUMBER: 1" <<   
 endl;

for (int i = 1, epochNumber = 1; i <= numberOfEpochs \* 20; i++)

{

learn(neuron, false, numberOfEntrances, numberOfWeights,   
 true);

if (i % 20 == 0)

{

OUTPUT\_LEARNING\_SECOND\_ALGORITHM\_FILE << "E: " << E <<  
 endl;

E = 0;

epochNumber++;

LEARNING\_DATA.clear();

LEARNING\_DATA.seekg(0, ios::beg);

if (epochNumber <= numberOfEpochs)

{

OUTPUT\_LEARNING\_SECOND\_ALGORITHM\_FILE << "NEW   
 EPOCH NUMBER: " << epochNumber << endl;

}

TOTAL\_BAD\_ANSWERS\_AMOUNT += BAD\_ANSWERS\_AMOUNT;

TOTAL\_RIGHT\_ANSWERS\_AMOUNT += RIGHT\_ANSWERS\_AMOUNT;

RIGHT\_ANSWERS\_AMOUNT = 0;

BAD\_ANSWERS\_AMOUNT = 0;

}

}

break;

case 4:

OUTPUT\_TESTING\_SECOND\_ALGORITHM\_FILE.open("output\_testing\_second\_algorithm\_data.  
 txt", ios::out);

BAD\_ANSWERS\_AMOUNT = 0;

RIGHT\_ANSWERS\_AMOUNT = 0;

E\_TESTING = 0;

for (int i = 0; i < 20; i++)

{

learn(neuron, true, numberOfEntrances, numberOfWeights,  
 true);

}

OUTPUT\_TESTING\_SECOND\_ALGORITHM\_FILE << "Bad answers amount: " <<   
 BAD\_ANSWERS\_AMOUNT << endl;

OUTPUT\_TESTING\_SECOND\_ALGORITHM\_FILE << "Right answers amount: " <<  
 RIGHT\_ANSWERS\_AMOUNT << endl;

OUTPUT\_TESTING\_SECOND\_ALGORITHM\_FILE << "E: " << E\_TESTING << endl;

break;

case 5:

OUTPUT\_LEARNING\_FILE.close();

OUTPUT\_TESTING\_FILE.close();

OUTPUT\_LEARNING\_SECOND\_ALGORITHM\_FILE.close();

OUTPUT\_TESTING\_SECOND\_ALGORITHM\_FILE.close();

LEARNING\_DATA.close();

TESTING\_DATA.close();

return 0;

}

} while (true);

return 0;

}

void setInputValue(Neuron& neuron, bool testing, int numberOfEntrances, int   
 numberOfWeights, bool secondAlgorithm)

{

if (testing == false)

{

int\* inputs = new int[numberOfEntrances];

int\* answers = new int[numberOfWeights];

if (secondAlgorithm == false)

{

for (int i = 0; i < numberOfEntrances; i++)

{

LEARNING\_DATA >> inputs[i];

}

for (int i = 0; i < numberOfWeights; i++)

{

LEARNING\_DATA >> answers[i];

}

for (int i = 0; i < numberOfEntrances; i++)

{

neuron.setInput(i, inputs[i]);

}

for (int i = 0; i < numberOfWeights; i++)

{

CORRECT\_ANSWER[i] = answers[i];

}

OUTPUT\_LEARNING\_FILE << "IN: ";

for (int i = 0; i < numberOfEntrances; i++)

{

OUTPUT\_LEARNING\_FILE << neuron.getInput(i) << " ";

}

for (int i = 0; i < numberOfWeights; i++)

{

OUTPUT\_LEARNING\_FILE << " RA: " << CORRECT\_ANSWER[i];

}

}

else

{

for (int i = 0; i < numberOfEntrances; i++)

{

LEARNING\_DATA >> inputs[i];

}

for (int i = 0; i < numberOfWeights; i++)

{

LEARNING\_DATA >> answers[i];

}

for (int i = 0; i < numberOfEntrances; i++)

{

neuron.setInput(i, inputs[i]);

}

for (int i = 0; i < numberOfWeights; i++)

{

CORRECT\_ANSWER[i] = answers[i];

}

OUTPUT\_LEARNING\_SECOND\_ALGORITHM\_FILE << "IN: ";

for (int i = 0; i < numberOfEntrances; i++)

{

OUTPUT\_LEARNING\_SECOND\_ALGORITHM\_FILE << neuron.getInput(i)   
 << " ";

}

for (int i = 0; i < numberOfWeights; i++)

{

OUTPUT\_LEARNING\_SECOND\_ALGORITHM\_FILE << " RA: " <<   
 CORRECT\_ANSWER[i];

}

}

delete inputs;

delete answers;

}

else

{

int\* inputs = new int[numberOfEntrances];

int\* answers = new int[numberOfWeights];

if (secondAlgorithm == false)

{

for (int i = 0; i < numberOfEntrances; i++)//load input

{

TESTING\_DATA >> inputs[i];

}

for (int i = 0; i < numberOfWeights; i++)//load correct answers

{

TESTING\_DATA >> answers[i];

}

for (int i = 0; i < numberOfEntrances; i++)//set neuron inputs

{

neuron.setInput(i, inputs[i]);

}

for (int i = 0; i < numberOfWeights; i++)//set correct answer

{

CORRECT\_ANSWER[i] = answers[i];

}

OUTPUT\_TESTING\_FILE << "IN: ";

for (int i = 0; i < numberOfEntrances; i++)

{

OUTPUT\_TESTING\_FILE << neuron.getInput(i) << " ";

}

for (int i = 0; i < numberOfWeights; i++)

{

OUTPUT\_TESTING\_FILE << " RA: " << CORRECT\_ANSWER[i];

}

}

else

{

for (int i = 0; i < numberOfEntrances; i++)//load input

{

TESTING\_DATA >> inputs[i];

}

for (int i = 0; i < numberOfWeights; i++)//load correct answers

{

TESTING\_DATA >> answers[i];

}

for (int i = 0; i < numberOfEntrances; i++)//set neuron inputs

{

neuron.setInput(i, inputs[i]);

}

for (int i = 0; i < numberOfWeights; i++)//set correct answer

{

CORRECT\_ANSWER[i] = answers[i];

}

OUTPUT\_TESTING\_SECOND\_ALGORITHM\_FILE << "IN: ";

for (int i = 0; i < numberOfEntrances; i++)

{

OUTPUT\_TESTING\_SECOND\_ALGORITHM\_FILE << neuron.getInput(i)  
 << " ";

}

for (int i = 0; i < numberOfWeights; i++)

{

OUTPUT\_TESTING\_SECOND\_ALGORITHM\_FILE << " RA: " << CORRECT\_ANSWER[i];

}

}

delete inputs;

delete answers;

}

}

void learn(Neuron& neuron, bool testing, int numberOfEntrances, int numberOfWeights  
 bool secondAlgorithm)

{

setInputValue(neuron, testing, numberOfEntrances, numberOfWeights,   
 secondAlgorithm);

neuron.sumOfInputs();

neuron.processOutput(secondAlgorithm, testing);

double\* neuronAnswer = neuron.getOutputValue();

int\* inputArray = new int[numberOfEntrances];

for (int i = 0; i < numberOfEntrances; i++)

{

inputArray[i] = neuron.getInput(i);

}

for (int i = 0; i < numberOfWeights; i++)

{

if (neuronAnswer[i]) //neuron answer = 1

{

if (CORRECT\_ANSWER[i] == 0)

{

BAD\_ANSWERS\_AMOUNT++;

if (testing == false)

{

neuron.countNewWeights(CORRECT\_ANSWER[i],   
 neuronAnswer[i], inputArray, i, secondAlgorithm);

if (secondAlgorithm == false)

{

E\_F += 0.5 \* pow(((double)CORRECT\_ANSWER[i] -   
 neuron.getOutputValue(i)), 2);

OUTPUT\_LEARNING\_FILE << " A: 1 - NEW WEIGHTS";

}

else

{

E += 0.5 \* pow(((double)CORRECT\_ANSWER[i] -   
 neuron.getOutputValue(i)), 2);

OUTPUT\_LEARNING\_SECOND\_ALGORITHM\_FILE << " NEW   
 WEIGHTS";

}

}

else

{

if (secondAlgorithm == false)

{

E\_F\_TESTING += 0.5 \*   
 pow(((double)CORRECT\_ANSWER[i] - neuron.getOutputValue(i)), 2);

OUTPUT\_TESTING\_FILE << " A: 1 -";

}

else

{

E\_TESTING += 0.5 \*   
 pow(((double)CORRECT\_ANSWER[i] - neuron.getOutputValue(i)),   
 2);

OUTPUT\_TESTING\_SECOND\_ALGORITHM\_FILE << " A: 1   
 -";

}

}

}

else

{

RIGHT\_ANSWERS\_AMOUNT++;

if (testing == false)

{

if (secondAlgorithm == false)

{

E\_F += 0.5 \* pow(((double)CORRECT\_ANSWER[i] -   
 neuron.getOutputValue(i)), 2);

OUTPUT\_LEARNING\_FILE << " A: 1 + KEEPING   
 WEIGHTS";

}

else

{

neuron.countNewWeights(CORRECT\_ANSWER[i],  
 neuronAnswer[i], inputArray, i, secondAlgorithm);

OUTPUT\_LEARNING\_SECOND\_ALGORITHM\_FILE << " NEW   
 WEIGHTS";

E += 0.5\* pow(((double)CORRECT\_ANSWER[i] -   
 neuron.getOutputValue(i)), 2);

}

}

else

{

if (secondAlgorithm == false)

{

E\_F\_TESTING += 0.5 \*   
 pow(((double)CORRECT\_ANSWER[i] - neuron.getOutputValue(i)), 2);

OUTPUT\_TESTING\_FILE << " A: 1 +";

}

else

{

E\_TESTING += 0.5 \*   
 pow(((double)CORRECT\_ANSWER[i] - neuron.getOutputValue(i)), 2);

OUTPUT\_TESTING\_SECOND\_ALGORITHM\_FILE << " A: 1   
 +";

}

}

}

}

else //neuron answer = 0

{

if (CORRECT\_ANSWER[i] == 1)

{

BAD\_ANSWERS\_AMOUNT++;

if (testing == false)

{

neuron.countNewWeights(CORRECT\_ANSWER[i],   
 neuronAnswer[i], inputArray, i, secondAlgorithm);

if (secondAlgorithm == false)

{

E\_F += 0.5 \* pow(((double)CORRECT\_ANSWER[i] -   
 neuron.getOutputValue(i)), 2);

OUTPUT\_LEARNING\_FILE << " A: 0 - NEW WEIGHTS";

}

else

{

E += 0.5 \* pow(((double)CORRECT\_ANSWER[i] -   
 neuron.getOutputValue(i)), 2);

OUTPUT\_LEARNING\_SECOND\_ALGORITHM\_FILE << " A:  
 0 - NEW WEIGHTS";

}

}

else // testing == true

{

if (secondAlgorithm == false)

{

E\_F\_TESTING += 0.5 \*   
 pow(((double)CORRECT\_ANSWER[i] - neuron.getOutputValue(i)), 2);

OUTPUT\_TESTING\_FILE << " A: 0 -";

}

else

{

E\_TESTING += 0.5 \*  
 pow(((double)CORRECT\_ANSWER[i] - neuron.getOutputValue(i)), 2);

OUTPUT\_TESTING\_SECOND\_ALGORITHM\_FILE << " A: 0   
 -";

}

}

}

else

{

RIGHT\_ANSWERS\_AMOUNT++;

if (testing == false)

{

if (secondAlgorithm == false)

{

E\_F += 0.5 \* pow(((double)CORRECT\_ANSWER[i] -   
 neuron.getOutputValue(i)), 2);

OUTPUT\_LEARNING\_FILE << " A: 0 + KEEPING   
 WEIGHTS";

}

else

{

neuron.countNewWeights(CORRECT\_ANSWER[i],   
 neuronAnswer[i], inputArray, i, secondAlgorithm);

OUTPUT\_LEARNING\_SECOND\_ALGORITHM\_FILE << " A:  
 0 + NEW WEIGHTS";

E += 0.5 \* pow(((double)CORRECT\_ANSWER[i] -   
 neuron.getOutputValue(i)), 2);

}

}

else

{

if (secondAlgorithm == false)

{

E\_F\_TESTING += 0.5 \*   
 pow(((double)CORRECT\_ANSWER[i] - neuron.getOutputValue(i)), 2);

OUTPUT\_TESTING\_FILE << " A: 0 +";

}

else

{

E\_TESTING += 0.5 \*  
 pow(((double)CORRECT\_ANSWER[i] - neuron.getOutputValue(i)), 2);

OUTPUT\_TESTING\_SECOND\_ALGORITHM\_FILE << " A: 0 +";

}

}

}

}

if (i == (numberOfWeights - 1))// file formatting

{

if (testing == false)

{

if (secondAlgorithm == false)

{

OUTPUT\_LEARNING\_FILE << endl;

}

else

{

OUTPUT\_LEARNING\_SECOND\_ALGORITHM\_FILE << endl;

}

}

else

{

if (secondAlgorithm == false)

{

OUTPUT\_TESTING\_FILE << endl;

}

else

{

OUTPUT\_TESTING\_SECOND\_ALGORITHM\_FILE << endl;

}

}

}

}

}