**Scenariusz 2**

**Temat ćwiczenia:** Budowa i działanie sieci jednowarstwowej

**Wykonała:** Burnat Magdalena, IS, gr. 3, 270652

1. **Cel ćwiczenia**

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

1. **Opis budowy sieci i algorytmów uczenia.**

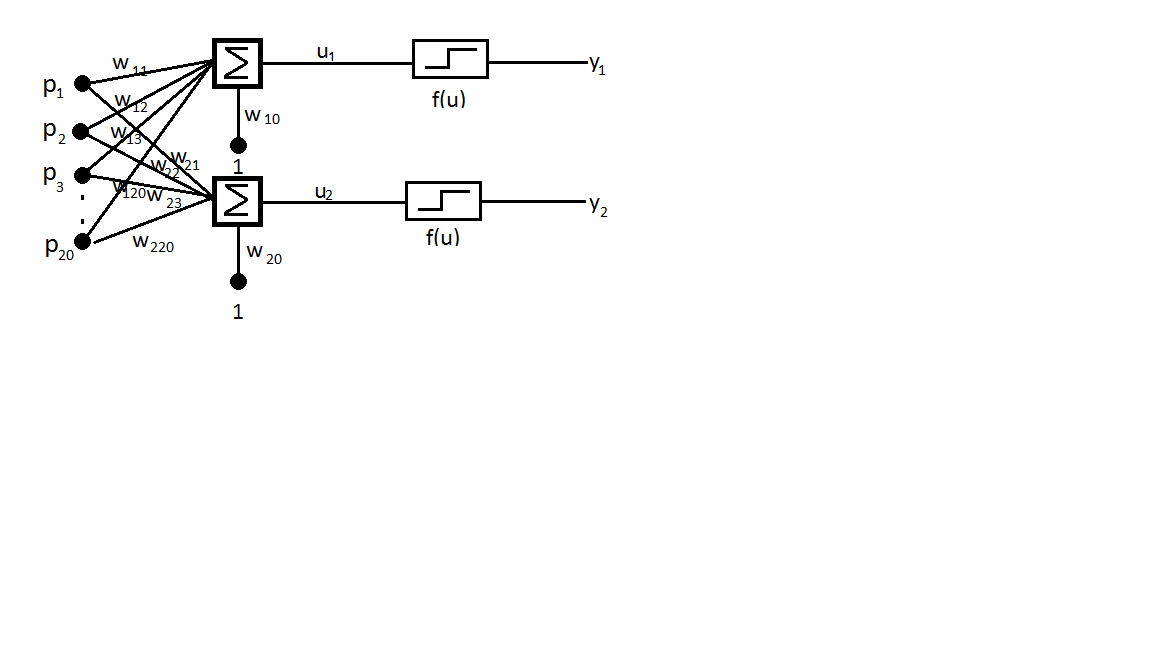
Celem budowanej sieci jest rozpoznawanie wielkości liter w tym celu stworzony został zestaw liter( 10 dużych i 10 małych), które są reprezentowane w postaci dwuwymiarowej tablicy 4x5 pikseli dla jednej litery.

Tab. 1. Nauczany alfabet.

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

A więc ciąg uczący składa się z 20 wektorów uczących, natomiast mamy dwa wektory wyjściowe, odpowiednio dla dużej litery 1 0, a dla małej 0 1. Dlatego wybrano sieć jednowarstwową składającą się z dwóch neuronów. Stworzona dwie różniące się między sobą sieci.

Pierwsza sieć została wykonana w sposób analogiczny do poprzedniego zagadnienia i została przedstawiona na poniższym rysunku.



Rys. 1 Pierwsza sieć neuronowa

Sieć wykorzystuje unipolarną, binarną funkcję aktywacji, natomiast proces uczenia przebiega według następującego schematu:

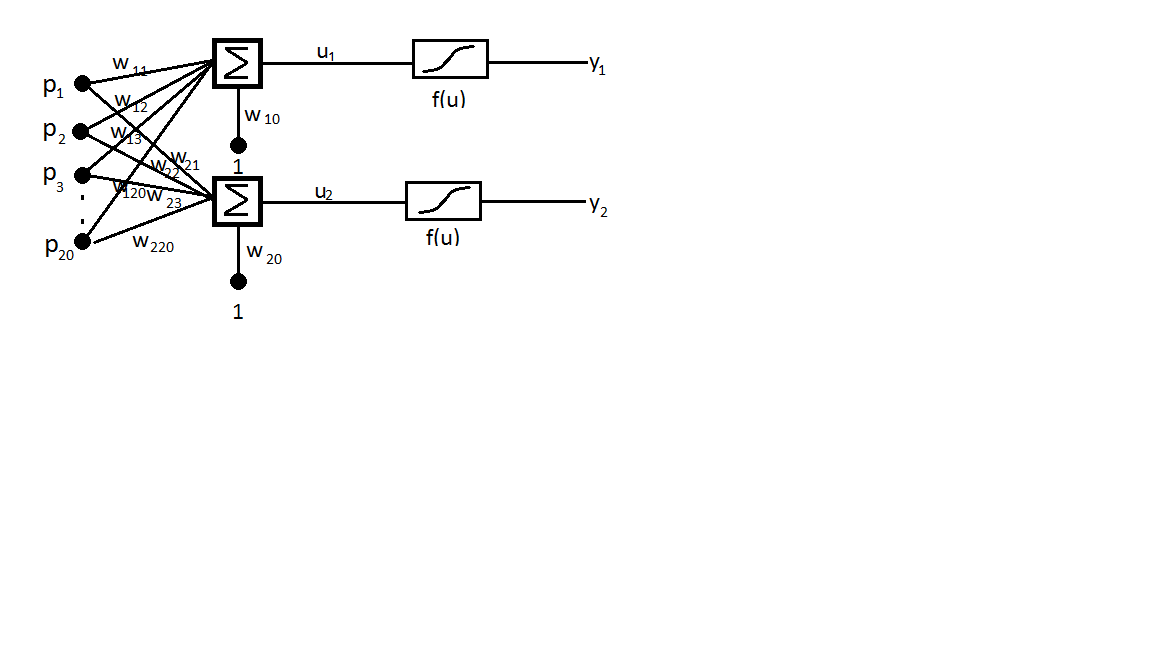
1. Wybór > 0 oraz Emax > 0  
2. Na wstępie wagi dobrano w sposób losowy.  
3. Obliczana jest suma ilorazów sygnałów wejściowych i wag-   
4. Obliczana jest wartość wyjściowa za pomocą funkcji binarnej, unipolarnej funkcji aktywacji.

5. 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:

6. Obliczenie łącznego błędu epoki:

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

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-   
5. Obliczana jest wartość wyjściowa 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**

Pierwszy etap testów miał na celu sprawdzenie jak zachowuje się sieć w zależności od zadanej liczby epok i wyznaczenia maksymalnego błędu średniokwadratowego poniżej którego można stwierdzić, że sieć poprawnie wykonuje zadanie. Kolejno stworzone sieci badano w zależności od zmiennego współczynnika nauczania. Aby możliwe było porównanie wyników obu sieci przeprowadzano badania dla liczby epok równej 100, po czym wykonano testowanie sieci z już zmodyfikowanymi wagami. Następnie sprawdzono ile dokładnie potrzeba epok, aby można było uznać sieć za poprawnie działającą wykorzystując obserwacje z pierwszego testu.

Tab. 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Zależność błędu średniokwadratowego od liczby epok dla współczynnika uczenia 0,5 | | | | |
| Liczba epok | Współczynnik E dla pierwszej sieci | Współczynnik E dla drugiej sieci | Procent dobrych odpowiedzi dla pierwszej sieci | Procent dobrych odpowiedzi dla drugiej sieci |
| 5 | 6,5 | 4,89596 | 75 | 75 |
| 10 | 4,5 | 4,3754 | 82,5 | 80 |
| 15 | 4 | 3,76292 | 90 | 85 |
| 20 | 2,5 | 3,17436 | 92,5 | 87,5 |
| 25 | 0,5 | 3,14749 | 100 | 87,5 |
| 30 | 0 | 2,59597 | 100 | 90 |
| 50 | 0 | 1,77287 | 100 | 92,5 |
| 60 | 0 | 1,56718 | 100 | 97,5 |
| 70 | 0 | 1,00682 | 100 | 100 |
| 80 | 0 | 1,06191 | 100 | 100 |
| 100 | 0 | 0,619017 | 100 | 100 |

Tab. 3. Wyniki uczenia i testowania pierwszej sieci.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sieć pierwsza | | | | |
| Faza uczenia | | | | Faza testowania |
| Współczynnik uczenia | Liczba epok potrzebnych, aby osiągnąć E < 1.0 | Współczynnik E po wykonaniu 100 epok uczenia | Spadek E między 1 epoką a 100 | Procent dobrych odpowiedzi po 100 epokach |
|
| 0,001 | 740 | 10,5 | 1,5 | 47,5 |
| 0,005 | 154 | 6,5 | 4,5 | 85 |
| 0,01 | 84 | 0 | 8 | 100 |
| 0,05 | 28 | 0 | 12 | 100 |
| 0,1 | 25 | 0 | 12,5 | 100 |
| 0,5 | 22 | 0 | 13 | 100 |
| 1 | 20 | 0 | 16 | 100 |

Tab. 4. Wyniki uczenia i testowanie drugiej sieci.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sieć druga | | | | |
| Faza uczenia | | | | Faza testowania |
| Współczynnik uczenia | Liczba epok potrzebnych, aby osiągnąć E < 1.0 | Współczynnik E po wykonaniu 100 epok uczenia | Spadek E między 1 epoką a 100 | Procent dobrych odpowiedzi po 100 epokach |
|
| 0,001 | > 10 000 | 5,00711 | 1,30724 | 55 |
| 0,005 | 6276 | 5,43404 | 2,2567 | 60 |
| 0,01 | 2990 | 4,52172 | 2,60596 | 65 |
| 0,05 | 599 | 3,14838 | 2,97046 | 80 |
| 0,1 | 309 | 2,49216 | 4,72793 | 92,5 |
| 0,5 | 72 | 0,64357 | 7,3345 | 100 |
| 1 | 61 | 0,270782 | 7,932438 | 100 |

1. **Analiza wyników**

Wykres 1. Zależność błędu średniokwadratowego od liczby epok dla obu modeli ( = 0,5).

Na powyższym wykresie możemy obserwować spadek błędu średnio kwadratowego wraz z liczbą epok uczenia. Na podstawie tabeli nr 2 można stwierdzić, że dla wartości błędu średniokwadratowego wynoszącej 1,00682 otrzymujemy sieć dającą 100% poprawnych odpowiedzi, a więc Emax ustala się na poziomie 1.0 co wykorzystano w dalszej części testów. Dodatkowo na podstawie powyższego wykresu widzimy, że pierwsza sieć dużo szybciej się uczy. Dla drugiej sieci wartość współczynnika β został ustalony 1.0 w innym wypadu otrzymywane rezultaty przez sieć nie były w żadnym stopniu zadowalające i nie porównywalne z siecią pierwszą. Zauważano również, że osiągnięcie błędu średniokwadratowego bliskiego 0 dla niej jest praktycznie nie możliwe, a dla pierwszej sieci osiągamy to już przy 30 epokach uczenia.

Wykres 2. Zależność liczby epok potrzebnych do nauczenia sieci w zależności od współczynnika uczenia dla obu modeli.

Na powyższym wykresie widzimy, że różnice liczby epok potrzebne do całkowitego nauczenia w poszczególnych sieciach są znaczne. Przy bardzo małych współczynnikach uczenia liczba epok jest co najmniej 10 razy większa. Dokładnie wartość potrzebnych epok do całkowitego nauczenia dla współczynnika uczenia równego 0,001 nie została zmierzona, ponieważ już dla 10 000 epok czas wykonywania obliczeń był długi a błąd średniokwadratowy był ponad 3. Zgodnie z regułą delty wagi były zwiększane o bardzo małe współczynniki, a więc i sieć uczyła się bardzo wolno.

Wykres 3. Zależność poprawnych odpowiedzi podczas testowania przy różnych współczynnikach uczenia dla różnych modeli.

Na podstawie wykresu można stwierdzić, że procent poprawnych odpowiedzi zwiększa się wraz ze wzrostem wartości współczynnika uczenia. Zauważalnym jest, że dla najniższego z testowanych współczynników różnica między obiema sieciami jest stosunkowo nieduża to już przy kolejnych, większych współczynnikach sieć z funkcją binarną wykazuje dużo większą skuteczność, więc proces uczenia przebiega szybciej. Natomiast przy współczynniku uczenia równym 0,5 otrzymujemy dwie poprawnie działające sieci, dla których procent poprawnych odpowiedzi wynosi 100%.

1. **Wnioski**

* Skuteczność procesu uczenia zależy od współczynnika uczenia. Wraz z jego wzrostem proces uczenia jest poprawniejszy. Jest to wytłumaczalne z tego względu, że im ta wartość jest większa tym przyrost wag, które na samym początku są niewielkie jest szybszy, a więc i proces uczenia przebiega szybciej. Współczynnik wpływa znacznie zarówno dla jednej jak i drugiej sieci.
* Pierwszy algorytm wykorzystujący binarną funkcję aktywacji działa dużo lepiej niż sieć wykorzystująca sigmoidalną funkcję. Oznacza to, że dla tak prostego działania sieci, jak rozpoznawanie wielkości liter lepsze efekty da funkcja binarna. Zauważano to także przy etapie wyboru współczynnika β, dla którego im jego wartość wyższa tym jego zachowanie bardziej przypomina działanie funkcji binarnej. Im jego wartość była niższa tym sieć działała gorzej, przy wartościach ok 0,5 postęp uczenia był prawie niezauważalny. Szybkość uczenia takiej sieci jest niewielki, co może być przydatne w bardziej zaawansowanych problemach i nie dopuści do przeuczenia sieci.
* Przy projektowaniu sieci neuronowej konieczne jest uwzględnienie modelu sieci. Przy naszym problemie lepiej spisywała się prostsza sieć, natomiast funkcje liniowe stosuje się przy sieciach wielowarstwowych stosowanych do bardziej złożonych zagadnień.

1. **Kod programu**

main.cpp

#include <iostream>

#include <vector>

#include "Neuron.h"

#include <time.h>

#include <fstream>

using namespace std;

void ustawWejscie(Neuron&, bool , int, int, int, bool);

void ucz(Neuron&, bool , int, double, int, bool);

fstream WYNIK\_NAUKI\_PLIK;

fstream WYNIK\_TESTOWANIA\_PLIK;

fstream WYNIKI\_NAUKI\_DWA\_PLIK;

fstream WYNIK\_TESTOWANIA\_DWA\_PLIK;

fstream DANE\_DO\_NAUKI\_PLIK;

fstream DANE\_DO\_TESTOWANIA;

double E\_F;

double E\_F\_TEST;

double E;

double E\_TEST;

int LICZBA\_ZLYCH\_ODPOWIEDZI;

int OGOLNA\_LICZBA\_ZLYCH\_ODPOWIEDZI;

int LICZBA\_DOBRYCH\_ODPOWIEDZI;

int OGOLNA\_LICZBA\_DOBRYCH\_ODPOWIEDZI;

int POPRAWNA\_ODPOWIEDZ[2];

int main()

{

srand(time(NULL));

int liczbaWejsc = 21;

int liczbaWag = 2;

Neuron neuron(liczbaWejsc, liczbaWag, 1);

DANE\_DO\_NAUKI\_PLIK.open("dane\_do\_nauki.txt", ios::in);

DANE\_DO\_TESTOWANIA.open("dane\_do\_testowania.txt", ios::in);

WYNIK\_NAUKI\_PLIK.open("wynik\_uczenia.txt", ios::out);

WYNIKI\_NAUKI\_DWA\_PLIK.open("wyniki\_uczenia\_dwa.txt", ios::out);

do

{

cout << "1. Ucz" << endl;

cout << "2. Testuj" << endl;

cout << "3. Ucz 2" << endl;

cout << "4. Testuj 2" << endl;

cout << "5. Wyjscie" << endl;

int wybor;

cin >> wybor;

switch (wybor)

{

case 1:

LICZBA\_ZLYCH\_ODPOWIEDZI = 0;

LICZBA\_DOBRYCH\_ODPOWIEDZI = 0;

OGOLNA\_LICZBA\_ZLYCH\_ODPOWIEDZI = 0;

OGOLNA\_LICZBA\_DOBRYCH\_ODPOWIEDZI = 0;

int liczbaEpok;

cout << "Wprowadz liczbe epok: " << endl;

cin >> liczbaEpok;

WYNIK\_NAUKI\_PLIK << "nowa epoka: 1" << endl;

for (int i = 0, numerEpoki = 1; i <= liczbaEpok \* 20; i++)

{

ucz(neuron, false, liczbaWejsc, liczbaWag, false);

if (i % 20 == 0)

{

numerEpoki++;

WYNIK\_NAUKI\_PLIK << "Liczba zlych odpowiedzi: " << LICZBA\_ZLYCH\_ODPOWIEDZI << endl;

WYNIK\_NAUKI\_PLIK << "Liczba dobrych odpowiedzi: " << LICZBA\_DOBRYCH\_ODPOWIEDZI << endl;

WYNIK\_NAUKI\_PLIK << "EF: " << E\_F << endl;

E\_F = 0;

if (numerEpoki <= liczbaEpok)

{

WYNIK\_NAUKI\_PLIK << "nowa epoka: " << numerEpoki << endl;

}

OGOLNA\_LICZBA\_ZLYCH\_ODPOWIEDZI += LICZBA\_ZLYCH\_ODPOWIEDZI;

OGOLNA\_LICZBA\_DOBRYCH\_ODPOWIEDZI += LICZBA\_DOBRYCH\_ODPOWIEDZI;

LICZBA\_DOBRYCH\_ODPOWIEDZI = 0;

LICZBA\_ZLYCH\_ODPOWIEDZI = 0;

}

}

WYNIK\_NAUKI\_PLIK << "\nOgolna liczba zlych odpowiedzi: " << OGOLNA\_LICZBA\_ZLYCH\_ODPOWIEDZI << endl;

WYNIK\_NAUKI\_PLIK << "Ogolna liczba dobrych odpowiedz: " << OGOLNA\_LICZBA\_DOBRYCH\_ODPOWIEDZI << endl;

break;

case 2:

WYNIK\_TESTOWANIA\_PLIK.open("wynik\_testow.txt", ios::out);

LICZBA\_ZLYCH\_ODPOWIEDZI = 0;

LICZBA\_DOBRYCH\_ODPOWIEDZI = 0;

E\_F\_TEST = 0;

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

ucz(neuron, false, liczbaWejsc, liczbaWag, false);

WYNIK\_TESTOWANIA\_PLIK << "Liczba zlych odpowiedzi: " << LICZBA\_ZLYCH\_ODPOWIEDZI << endl;

WYNIK\_TESTOWANIA\_PLIK << "Liczba dobrych odpowiedzi: " << LICZBA\_DOBRYCH\_ODPOWIEDZI << endl;

WYNIK\_TESTOWANIA\_PLIK << "E: " << E\_F\_TEST << endl;

break;

case 3:

LICZBA\_ZLYCH\_ODPOWIEDZI = 0;

LICZBA\_DOBRYCH\_ODPOWIEDZI = 0;

OGOLNA\_LICZBA\_ZLYCH\_ODPOWIEDZI = 0;

OGOLNA\_LICZBA\_DOBRYCH\_ODPOWIEDZI = 0;

cout << "Wpisz liczbe epok: " << endl;

cin >> liczbaEpok;

WYNIKI\_NAUKI\_DWA\_PLIK << "nowa epoka: 1" << endl;

for (int i = 0, numerEpoki = 1; i <= liczbaEpok \* 20; i++)

{

ucz(neuron, false, liczbaWejsc, liczbaWag, true);

if (i % 20 == 0)

{

WYNIKI\_NAUKI\_DWA\_PLIK << "E: " << E << endl;

E = 0;

numerEpoki++;

if (numerEpoki <= liczbaEpok)

WYNIKI\_NAUKI\_DWA\_PLIK << "nowa epoka: " << numerEpoki << endl;

OGOLNA\_LICZBA\_ZLYCH\_ODPOWIEDZI += LICZBA\_ZLYCH\_ODPOWIEDZI;

OGOLNA\_LICZBA\_DOBRYCH\_ODPOWIEDZI += LICZBA\_DOBRYCH\_ODPOWIEDZI;

LICZBA\_DOBRYCH\_ODPOWIEDZI = 0;

LICZBA\_ZLYCH\_ODPOWIEDZI = 0;

}

}

break;

case 4:

WYNIK\_TESTOWANIA\_DWA\_PLIK.open("wynik\_testow\_dwa.txt", ios::out);

LICZBA\_ZLYCH\_ODPOWIEDZI = 0;

LICZBA\_DOBRYCH\_ODPOWIEDZI = 0;

E\_TEST = 0;

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

ucz(neuron, true, liczbaWejsc, liczbaWag, false);

WYNIK\_TESTOWANIA\_DWA\_PLIK << "Liczba zlych odpowiedzi: " << LICZBA\_ZLYCH\_ODPOWIEDZI << endl;

WYNIK\_TESTOWANIA\_DWA\_PLIK << "Liczba dobrych odpowiedzi: " << LICZBA\_DOBRYCH\_ODPOWIEDZI << endl;

WYNIK\_TESTOWANIA\_DWA\_PLIK << "E: " << E\_TEST << endl;

break;

case 5:

WYNIK\_NAUKI\_PLIK.close();

WYNIK\_TESTOWANIA\_PLIK.close();

WYNIKI\_NAUKI\_DWA\_PLIK.close();

WYNIK\_TESTOWANIA\_DWA\_PLIK.close();

DANE\_DO\_NAUKI\_PLIK.close();

DANE\_DO\_TESTOWANIA.close();

return 0;

}

} while (true);

return 0;

}

void ustawWejscie(Neuron& neuron, bool testowanie, int liczbaWejsc, int liczbaWag, bool drugi)

{

if (testowanie == false)

{

int\* wejscia = new int[liczbaWejsc];

int\* odpowiedzi = new int[liczbaWag];

if (drugi == false)

{

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

DANE\_DO\_NAUKI\_PLIK >> wejscia[i];

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

DANE\_DO\_NAUKI\_PLIK >> odpowiedzi[i];

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

neuron.ustawWejscie(i, wejscia[i]);

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

POPRAWNA\_ODPOWIEDZ[i] = odpowiedzi[i];

WYNIK\_NAUKI\_PLIK << "IN: ";

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

WYNIK\_NAUKI\_PLIK << neuron.pobierzWejscie(i) << " ";

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

WYNIK\_NAUKI\_PLIK << " RA: " << POPRAWNA\_ODPOWIEDZ[i];

}

else

{

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

DANE\_DO\_NAUKI\_PLIK >> wejscia[i];

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

DANE\_DO\_NAUKI\_PLIK >> odpowiedzi[i];

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

neuron.ustawWejscie(i, wejscia[i]);

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

POPRAWNA\_ODPOWIEDZ[i] = odpowiedzi[i];

WYNIKI\_NAUKI\_DWA\_PLIK << "IN: ";

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

WYNIKI\_NAUKI\_DWA\_PLIK << neuron.pobierzWejscie(i) << " ";

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

WYNIKI\_NAUKI\_DWA\_PLIK << " RA: " << POPRAWNA\_ODPOWIEDZ[i];

}

delete wejscia;

delete odpowiedzi;

}

else

{

int\* wejscia = new int[liczbaWejsc];

int\* odpowiedzi = new int[liczbaWag];

if (drugi == false)

{

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

DANE\_DO\_TESTOWANIA >> wejscia[i];

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

DANE\_DO\_TESTOWANIA >> odpowiedzi[i];

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

neuron.ustawWejscie(i, wejscia[i]);

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

POPRAWNA\_ODPOWIEDZ[i] = odpowiedzi[i];

WYNIK\_TESTOWANIA\_PLIK << "IN: ";

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

WYNIK\_TESTOWANIA\_PLIK << neuron.pobierzWejscie(i) << " ";

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

WYNIK\_TESTOWANIA\_PLIK << " RA: " << POPRAWNA\_ODPOWIEDZ[i];

}

else

{

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

DANE\_DO\_TESTOWANIA >> wejscia[i];

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

DANE\_DO\_TESTOWANIA >> odpowiedzi[i];

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

neuron.ustawWejscie(i, wejscia[i]);

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

POPRAWNA\_ODPOWIEDZ[i] = odpowiedzi[i];

WYNIK\_TESTOWANIA\_DWA\_PLIK << "IN: ";

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

WYNIK\_TESTOWANIA\_DWA\_PLIK << neuron.pobierzWejscie(i) << " ";

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

WYNIK\_TESTOWANIA\_DWA\_PLIK << " RA: " << POPRAWNA\_ODPOWIEDZ[i];

}

delete wejscia;

delete odpowiedzi;

}

}

void ucz(Neuron& neuron, bool testowanie, int liczbaWejsc, int liczbaWag, bool dwa)

{

ustawWejscie(neuron, testowanie, liczbaWejsc, liczbaWag, dwa);

neuron.sumaWejsc();

neuron.obliczWyjscie(dwa, testowanie);

double\* odpowiedziNeuronu = neuron.pobierzWartoscWyjsciowa();

int\* macierzWejsc = new int[liczbaWejsc];

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

macierzWejsc[i] = neuron.pobierzWejscie(i);

for (int i = 0, int j = 1; i < liczbaWag; i++, j--)

{

if (odpowiedziNeuronu[i])

{

if (POPRAWNA\_ODPOWIEDZ[i] == 0)

{

LICZBA\_ZLYCH\_ODPOWIEDZI++;

if (testowanie == false)

{

neuron.obliczNoweWagi(POPRAWNA\_ODPOWIEDZ[i], odpowiedziNeuronu[i], macierzWejsc, i, dwa);

if (dwa == true)

{

E\_F += 0.5 \* pow(((double)POPRAWNA\_ODPOWIEDZ[i] - neuron.pobierzWartoscWyjsciowa(i)), 2);

WYNIK\_NAUKI\_PLIK << " A: 1 - Nowe wagi";

}

else

{

E += 0.5 \* pow(((double)POPRAWNA\_ODPOWIEDZ[i] - neuron.pobierzWartoscWyjsciowa(i)), 2);

WYNIKI\_NAUKI\_DWA\_PLIK << " Nowe wagi";

}

}

else

{

if (dwa == true)

{

E\_F\_TEST += 0.5 \* pow(((double)POPRAWNA\_ODPOWIEDZ[j] - neuron.pobierzWartoscWyjsciowa(i)), 2);

WYNIK\_TESTOWANIA\_PLIK << " A: 1 -";

}

else

{

E\_TEST += 0.5 \* pow(((double)POPRAWNA\_ODPOWIEDZ[i] - neuron.pobierzWartoscWyjsciowa(i)), 2);

WYNIK\_TESTOWANIA\_DWA\_PLIK << " A: 1 -";

}

}

}

else

{

LICZBA\_DOBRYCH\_ODPOWIEDZI++;

if (testowanie == false)

{

if (dwa == true)

{

E\_F += 0.5 \* pow(((double)POPRAWNA\_ODPOWIEDZ[j] - neuron.pobierzWartoscWyjsciowa(i)), 2);

WYNIK\_NAUKI\_PLIK << " A: 1 + zostawiam wagi";

}

else

{

neuron.obliczNoweWagi(POPRAWNA\_ODPOWIEDZ[i], odpowiedziNeuronu[i], macierzWejsc, i, dwa);

WYNIKI\_NAUKI\_DWA\_PLIK << " nowe wagi";

E += 0.5\* pow(((double)POPRAWNA\_ODPOWIEDZ[i] - neuron.pobierzWartoscWyjsciowa(i)), 2);

}

}

else

{

if (dwa == true)

{

E\_F\_TEST += 0.5 \* pow(((double)POPRAWNA\_ODPOWIEDZ[j] - neuron.pobierzWartoscWyjsciowa(i)), 2);

WYNIK\_TESTOWANIA\_PLIK << " A: 1 +";

}

else

{

E\_TEST += 0.5 \* pow(((double)POPRAWNA\_ODPOWIEDZ[i] - neuron.pobierzWartoscWyjsciowa(i)), 2);

WYNIK\_TESTOWANIA\_DWA\_PLIK << " A: 1 +";

}

}

}

}

else

{

if (POPRAWNA\_ODPOWIEDZ[i] == 1)

{

LICZBA\_ZLYCH\_ODPOWIEDZI++;

if (testowanie == false)

{

neuron.obliczNoweWagi(POPRAWNA\_ODPOWIEDZ[i], odpowiedziNeuronu[i], macierzWejsc, i, dwa);

if (dwa == true)

{

E\_F += 0.5 \* pow(((double)POPRAWNA\_ODPOWIEDZ[i] - neuron.pobierzWartoscWyjsciowa(i)), 2);

WYNIK\_NAUKI\_PLIK << " A: 0 - nowe wagi";

}

else

{

E += 0.5 \* pow(((double)POPRAWNA\_ODPOWIEDZ[i] - neuron.pobierzWartoscWyjsciowa(i)), 2);

WYNIKI\_NAUKI\_DWA\_PLIK << " A: 0 - nowe wagi";

}

}

else

{

if (dwa == true)

{

E\_F\_TEST += 0.5 \* pow(((double)POPRAWNA\_ODPOWIEDZ[i] - neuron.pobierzWartoscWyjsciowa(i)), 2);

WYNIK\_TESTOWANIA\_PLIK << " A: 0 -";

}

else

{

E\_TEST += 0.5 \* pow(((double)POPRAWNA\_ODPOWIEDZ[i] - neuron.pobierzWartoscWyjsciowa(i)), 2);

WYNIK\_TESTOWANIA\_DWA\_PLIK << " A: 0 -";

}

}

}

else

{

LICZBA\_DOBRYCH\_ODPOWIEDZI++;

if (testowanie == false)

{

if (dwa == true)

{

E\_F += 0.5 \* pow(((double)POPRAWNA\_ODPOWIEDZ[i] - neuron.pobierzWartoscWyjsciowa(i)), 2);

WYNIK\_NAUKI\_PLIK << " A: 0 + zostawiam wagi";

}

else

{

neuron.obliczNoweWagi(POPRAWNA\_ODPOWIEDZ[i], odpowiedziNeuronu[i], macierzWejsc, i, dwa);

WYNIKI\_NAUKI\_DWA\_PLIK << " A: 0 + nowe wagi";

E += 0.5 \* pow(((double)POPRAWNA\_ODPOWIEDZ[i] - neuron.pobierzWartoscWyjsciowa(i)), 2);

}

}

else

{

if (dwa == true)

{

E\_F\_TEST += 0.5 \* pow(((double)POPRAWNA\_ODPOWIEDZ[i] - neuron.pobierzWartoscWyjsciowa(i)), 2);

WYNIK\_TESTOWANIA\_PLIK << " A: 0 +";

}

else

{

E\_TEST += 0.5 \* pow(((double)POPRAWNA\_ODPOWIEDZ[i] - neuron.pobierzWartoscWyjsciowa(i)), 2);

WYNIK\_TESTOWANIA\_DWA\_PLIK << " A: 0 +";

}

}

}

}

if (i == (liczbaWag - 1))

{

if (testowanie == false)

{

if (dwa == false)

{

WYNIK\_NAUKI\_PLIK << endl;

}

else

{

WYNIKI\_NAUKI\_DWA\_PLIK << endl;

}

}

else

{

if (dwa == false)

{

WYNIK\_TESTOWANIA\_PLIK << endl;

}

else

{

WYNIK\_TESTOWANIA\_DWA\_PLIK << endl;

}

}

}

}

}

Neuron.cpp

#include "Neuron.h"

#include <time.h>

#include <math.h>

void Neuron::stworzWejscia(int liczbaWejsc, int liczbaWag)

{

vector<double> rzad;

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

{

rzad.clear();

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

{

if (i == 0)

\_wejscia.push\_back(0);

rzad.push\_back(obliczPierwszeWagi());

}

\_wagi.push\_back(rzad);

}

}

void Neuron::sumaWejsc()

{

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

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

{

sum[i] = 0;

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

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

\_sumaWszystkichWejsc[i] = sum[i];

}

}

void Neuron::obliczWyjscie(bool drugi, bool test)

{

if (drugi == false || test == true)

{

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

{

if (\_sumaWszystkichWejsc[i] > 0)

{

\_wartoscWyjsciowa[i] = 1;

}

else

{

\_wartoscWyjsciowa[i] = 0;

}

}

}

else // second algorithm

{

double beta = 1;

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

\_wartoscWyjsciowa[i] = (1.0 / (1.0 + (exp(-beta \* \_sumaWszystkichWejsc[i]))));

}

}

void Neuron::obliczNoweWagi(int prawidlowaOdpowiedz, int odpNeuron, int\* wejscia, int numerWej, bool drugi)

{

if (drugi == false)

{

\_wagi[numerWej][0] += \_wspolczynnikUczenia \* (prawidlowaOdpowiedz - odpNeuron);

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

\_wagi[numerWej][j] += \_wspolczynnikUczenia \* (prawidlowaOdpowiedz - odpNeuron) \* wejscia[j];

}

else // second algorithm

{

\_wagi[numerWej][0] += \_wspolczynnikUczenia \* (prawidlowaOdpowiedz - \_wartoscWyjsciowa[numerWej]);

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

{

\_wagi[numerWej][j] += \_wspolczynnikUczenia \* (odpNeuron - \_wartoscWyjsciowa[numerWej] ) \* (\_wartoscWyjsciowa[numerWej] \* (1 - \_wartoscWyjsciowa[numerWej])) \*wejscia[j];

}

}

}

double Neuron::obliczPierwszeWagi()

{

double a = 1.0;

double b = -1.0;

double w = ((double(rand()) / double(RAND\_MAX)) \* (a - b)) + b;

return w;

}

Neuron.h

#pragma once

#include <iostream>

using namespace std;

class Neuron

{

public:

void stworzWejscia(int, int);

void stworzWejscie() { \_wejscia.push\_back(0); }

void stworzWage(int index) { \_wagi[index].push\_back(0); }

int pobierzIloscWejsc() { return \_wejscia.size(); }

int pobierzIloscWag() { return \_wagi.size(); }

double pobierzWejscie(int index) { return \_wejscia[index]; }

void ustawWejscie(int index, double value) { \_wejscia[index] = value; }

double pobierzWage(int index1, int index2) { return \_wagi[index1][index2]; }

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

double pobierzSumeWszystkichWejsc(int index) { return \_sumaWszystkichWejsc[index]; }

double pobierzWartoscWyjsciowa() { return \_wartoscWyjsciowa; }

double pobierzWartoscWyjsciowa(int index){ return \_wartoscWyjsciowa[index]; }

double obliczPojedynczeWejscie(int index, int synapseIndex) { return \_wejscia[index] \* \_wagi[synapseIndex][index]; }

void sumaWejsc();

void obliczWyjscie(bool, bool);

void obliczNoweWagi(int, int, int\* , int, bool);

Neuron(int liczbaWejsc, int liczbaWag, double wspolczynnikUczenia)

{

stworzWejscia(liczbaWejsc, liczbaWag);

\_wspolczynnikUczenia = wspolczynnikUczenia;

\_sumaWszystkichWejsc = new double[liczbaWag];

\_wartoscWyjsciowa = new double[liczbaWag];

}

private:

double obliczPierwszeWagi();

vector<double> \_wejscia;

vector< vector<double> > \_wagi;

double \*\_sumaWszystkichWejsc;

double \*\_wartoscWyjsciowa;

double \_wspolczynnikUczenia;

int \_rozmiar;

};