Temat: ***Uczenie​ ​ sieci​ ​ regułą​ ​ Hebba.***

Data oddania: 20.10.2017

Prowadzący: dr inż. Dorota Wilk - Kołodziejczyk

Przedmiot: Podstawy sztucznej inteligencji

Autor: Przemysław Adamus 286084

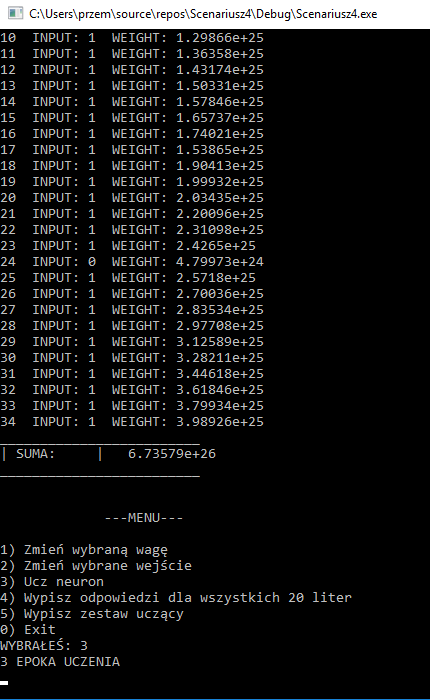
Kierunek: Inżynieria Obliczeniowa sem.V

Spis treści:

1. Cel projektu;
2. Wstęp i opis projektu;
3. Budowa sieci;
4. Zestaw uczący;
5. Syntetyczny​ ​opis​ ​budowy​​ ​wykorzystanego​ ​algorytmu​ ​uczenia w projekcie;
6. Opis wykorzystanych funkcji;
7. Wyniki i analiza;
8. Wnioski;
9. Przykładowa odpowiedź programu;
10. Źródła;
11. Kod programu.
12. **CEL:**

Celem ćwiczenia było​ ​ poznanie​ ​ działania​ ​ reguły​ ​ Hebba​ ​ dla​ ​ sieci​ ​ jednowarstwowej​ ​ na przykładzie​ ​ grupowania​ ​ liter​ ​ alfabetu.

1. **WSTĘP I KRÓTKI OPIS PROJEKTU:**

Do rozwiązania zadanego problemu wykorzystałem wiedzę i projekt C++ z wcześniejszego scenariusza. Do rozwiązywania zadania zbudowałem sieć jednowarstwową. Pojedynczy perceptron z ilości wejść odpowiadającej liczbie pikseli użytej matrycy. Zadany problem rozwiązałem przy użyciu środowiska C++.

Zadaniem mojego programu było rozpoznanie wielkości podanej litery. Najpierw przygotowałem zestaw danych uczących. Umieściłem go w pliku .csv. Plik dane.csv zawiera zbiór 20 dużych liter zapisanych za pomocą 1 i 0. Do programu dane wczytuje za pomocą funkcji void loadData(bool\*\*). Dane uczące można znaleźć w punkcie 3 tego sprawozdania.

W sprawozdaniu zostały umieszczone rozwiązania, kody i wyniki zadania z środowiska C++. Stworzyłem klasę neuron wraz z całą gamą metod do działania na obiektach tej klasy. Znajduje się tam również funkcja do nauki neuronu. Klasę umieściłem w pliku Neurone.h – załącznik 1, a implementacje metod w pliku Neuron.cpp – załącznik2.

W funkcji main() możemy znaleźć proste menu do obsługi i działaniu na naszym neuronie - załącznik 3. Po każdym działaniu możemy zaobserwować czy zwracana wartość i suma membranowa różni się od poprzedniej.

Po zakończonej implementacji przetestowałem mój neuron.

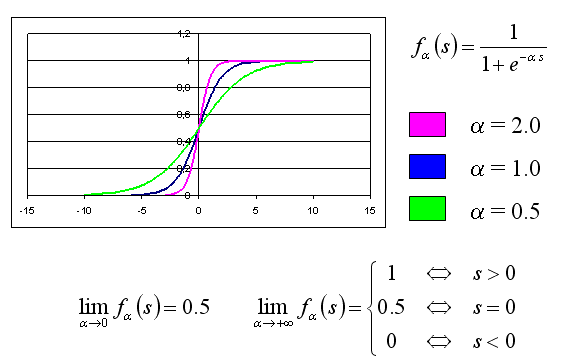
1. **BUDOWA WYKORZYSTANEGO PERCEPTRONU:**

Perceptronem nazywamy prosty element obliczeniowy, który sumuje ważone sygnały wejściowe i porównuję tę sumę z progiem aktywacji - w zależności od wyniku perceptron może być albo wzbudzony (wynik 1), albo nie (wynik 0).

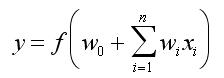
Suma membranowa jest to suma wszystkich iloczynów wejść i ich wag. Jest ona przekazywana dalej do funkcji aktywacji.

Funkcja aktywacji w sztucznej inteligencji określa funkcje, według której obliczana jest wartość wyjścia neuronów sieci neuronowej. Do najczęściej używanych funkcji aktywacji należą: funkcja liniowa, funkcja progowa unipolarna, funkcja progowa bipolarna, funkcja sigmoidalna unipolarna, funkcja sigmoidalna bipolarna.

Rezultat wiadomość zwrotna od neuronu. Może być przekazana dalej, lub być ostatecznym wynikiem. Rezultat jest wartością logiczną więc powinien zawierać tylko informacjię *true* lub *false.*



Ogólny wzór na wartość wyjściową neuronu przedstawia się następująco:



gdzie *w0* jest dodatkową wagą pełniącą rolę analogiczną do progu aktywacji w perceptronie.

1. **ZESTAW UCZĄCY:**

0;1;1;1;0;

1;0;0;0;1;

1;0;0;0;1;

1;1;1;1;1;

1;0;0;0;1;

1;0;0;0;1;

1;0;0;0;1;

1;1;1;1;0;

1;0;0;0;1;

1;0;0;0;1;

1;1;1;1;0;

1;0;0;0;1;

1;0;0;0;1;

1;1;1;1;0;

0;1;1;1;0;

1;0;0;0;1;

1;0;0;0;0;

1;0;0;0;0;

1;0;0;0;0;

1;0;0;0;1;

0;1;1;1;0;

1;1;1;1;0;

1;0;0;0;1;

1;0;0;0;1;

1;0;0;0;1;

1;0;0;0;1;

1;0;0;0;1;

1;1;1;1;0;

1;1;1;1;1;

1;0;0;0;0;

1;0;0;0;0;

1;1;1;1;0;

1;0;0;0;0;

1;0;0;0;0;

1;1;1;1;1;

1;1;1;1;1;

1;0;0;0;0;

1;0;0;0;0;

1;1;1;1;0;

1;0;0;0;0;

1;0;0;0;0;

1;0;0;0;0;

0;1;1;1;1;

1;0;0;0;0;

1;0;0;0;0;

1;0;1;1;1;

1;0;1;0;1;

1;0;0;0;1;

0;1;1;1;0;

1;0;0;0;1;

1;0;0;0;1;

1;0;0;0;1;

1;1;1;1;1;

1;0;0;0;1;

1;0;0;0;1;

1;0;0;0;1;

0;1;1;1;0;

0;0;1;0;0;

0;0;1;0;0;

0;0;1;0;0;

0;0;1;0;0;

0;0;1;0;0;

0;1;1;1;0;

1;1;1;1;1;

1;0;0;0;0;

1;0;0;0;0;

1;0;0;0;0;

1;0;0;0;0;

1;0;0;0;1;

1;1;1;1;0;

1;0;0;0;1;

1;0;0;1;0;

1;0;1;0;0;

1;1;0;0;0;

1;0;1;0;0;

1;0;0;1;0;

1;0;0;0;1;

1;0;0;0;0;

1;0;0;0;0;

1;0;0;0;0;

1;0;0;0;0;

1;0;0;0;0;

1;0;0;0;1;

1;1;1;1;1;

1;1;0;1;1;

1;0;1;0;1;

1;0;1;0;1;

1;0;0;0;1;

1;0;0;0;1;

1;0;0;0;1;

1;0;0;0;1;

1;0;0;0;1;

1;1;0;0;1;

1;0;1;0;1;

1;0;1;0;1;

1;0;1;0;1;

1;0;0;1;1;

1;0;0;0;1;

0;1;1;1;0;

1;0;0;0;1;

1;0;0;0;1;

1;0;0;0;1;

1;0;0;0;1;

1;0;0;0;1;

0;1;1;1;0;

1;1;1;1;0;

1;0;0;0;1;

1;0;0;0;1;

1;1;1;1;0;

1;0;0;0;0;

1;0;0;0;0;

1;0;0;0;0;

1;1;1;1;0;

1;0;0;0;1;

1;0;0;0;1;

1;1;1;1;0;

1;0;1;0;0;

1;0;0;1;0;

1;0;0;0;1;

1;1;1;1;1;

1;0;1;0;1;

0;0;1;0;0;

0;0;1;0;0;

0;0;1;0;0;

0;0;1;0;0;

0;0;1;0;0;

1;0;0;0;1;

1;0;0;0;1;

1;0;0;0;1;

1;0;0;0;1;

1;0;1;0;1;

1;0;1;0;1;

0;1;0;1;0;

1;1;1;1;1;

0;0;0;0;1;

0;0;0;1;0;

0;0;1;0;0;

0;1;0;0;0;

1;0;0;0;0;

1;1;1;1;1

1. **ALGORYTM UCZENIA - ALGORYTM HEBBA:**

Model ten ma identyczną strukturę jak w przypadku modelu typu Adaline oraz neuronu sigmoidalnego, ale charakteryzuje się specyficzną metodą uczenia, znaną pod nazwą reguły Hebba. Reguła ta występuje z nauczycielem jak i bez nauczyciela. Hebb zauważył podczas badań działania komórek nerwowych, i połączenie pomiędzy dwiema komórkami jest wzmacniane, jeśli w tym samym czasie obie komórki są aktywne.

Zaproponował on algorytm, zgodnie z którym modyfikację wag przeprowadza się następująco:

**Wi(t+1) = Wi(t) + nyxi**

Oznaczenia:

• **i** - numer wagi neuronu,

• **t** - numer iteracji w epoce,

• **y** - sygnał wyjściowy neuronu,

• **x** - wartość wejściowa neuronu,

• **n** - współczynnik uczenia (0,1).

W przypadku pojedynczego neuronu w trakcie uczenia będziemy modyfikować wartość wag proporcjonalnie zarówno do wartości sygnału podanego na i-te wejście, jak i sygnału wyjściowego y z uwzględnieniem współczynnika uczenia. Zauważmy, że w przypadku tym nie podajemy wzorcowej wartości wyjściowej, stosujemy więc tu metodę uczenia bez nauczyciela. Niewielka modyfikacja prowadzi do drugiej metody uczenia neuronu Hebba- z nauczycielem (d-sygnał wzorcowy).

Pewną wadą omawianego przez nas algorytmu jest to, iż wartości wag mogą wzrastać do dowolnie dużych liczb. Dlatego powstała wzbogacona wersja tego algorytmu uczenia. Dodano do niej współczynnik zapominania, dzięki któremu możemy kontrolować wartość wag. Modyfikacja wag w algorytmie Hebba z współczynnikiem zapominania przebiega następująco:

**Wi(t+1) = (1-z)\*Wi(t) + nyxi**

Oznaczenia:

* **i** - numer wagi neuronu,
* **t** - numer iteracji w epoce,
* **y** - sygnał wyjściowy neuronu,
* **x** - wartość wejściowa neuronu,
* **z** - współczynnik zapominania (0,1),
* **n** - współczynnik uczenia (0,1).

1. **WYNIKI :**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| W.NAUKI | 0.001 | | 0.0005 | | 0.01 | |
| L.EPOK | MAX | MIN | MAX | MIN | MAX | MIN |
| 1. | 7.95107 (N) | 4.29300 (I) | 6.20981 (N) | 4.20861 (I) | 1815.31 (B) | 1018.10 (I) |
| 5. | 84.2286 (B) | 46.0905 (I) | 15.8614 (B) | 9.58782 (I) | 5.53606e+13 (B) | 3.10442e+13 (I) |
| 10. | 1752 (B) | 955.32 (I) | 51.9370 (B) | 29.2783 (I) | 7.05537e+26 (B) | 3.95639e+26 (I) |
| 20. | 760619 (B) | 414671 (I) | 543.862 (B) | 297.778 (I) | … | … |
| 30. | 3.30247e+08 (B) | 1.80043e+08 (I) | 4481.43 (B) | 2446.96 (I) | … | … |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| W.NAUKI | 0.001 | | | | 0.05 | |
| W.ZAPOMINANIA | 0.001 | | 0.05 - większy | | 0.001 – mniejszy | |
| L.EPOK | MAX | MIN | MAX | MIN | MAX | MIN |
| 1. | 6.91979 (B) | 3.94806 (I) | 3.18709 (B) | 1.73024 (I) | 2.00851e+13 | 1.20969e+13 |
| 5. | 70.3168 (B) | 38.4979 (I) | 0.571051 (B) | 0.312282 (I) | 9.71993e+64 (T) | 1.13294e+65 (I) |
| 10. | 1323.74 (B) | 721.812 (I) | 0.0766283 (B) | 0.0420407 (I) | … | … |
| 20. | 471903 (B) | 257268 (I) | 0.00138715 (B) | 0.00076115 (I) | … | … |

1. **WNIOSKI:**

Na początku w wnioskach chciałbym zwrócić uwagę na powstałe grupy liter. Litery grupują się ze względu na podobieństwo. Jeśli dwie litery posiadają dużą liczbę wspólnych pikseli to wyniki perceptronu dla obu liter powinny być podobne (np. dla liter B i D). Natomiast jeśli litery nie mają, lub mają mało wspólnych pikseli to wyniki perceptronu dla tych liter powinny się różnić (np. dla liter I i A). Różnica między niepodobnymi literami powinna wzrastać wraz z ilością epok nauczania. Ilość grup i granica między nimi powinna tworzyć się naturalnie i samodzielnie. Może ona zmieniać wartości wraz z ilością epok, ale obiekty nie powinny zmieniać obrębu swojej grupy.

Analizując uzyskane odpowiedzi, mogę stwierdzić, że algorytm dzieli litery na co najmniej 3 grupy podobieństwa już po pierwszej epoce nauczania, ale granica dobrze widoczna jest po co najmniej 5 epokach.

Współczynnik nauczania odgrywa ogromną rolę w zastosowanej metodzie Hebba. Wpływa on bezpośrednio na wartości wag, które jak wiemy mogą rosnąć w nieskończoność. Nieznormalizowane wagi mogą pływać niekorzystnie na wydajność, ale dzięki nim można zauważyć wyraźnie granice między grupami.

Współczynnik nauczania jest w opozycji do współczynnika zapominania. To właśnie on dba, aby nasze wartości wag nie zmierzały za szybko do nieskończoności. Dobranie tych dwóch parametrów jest bardzo ważne dla reguły Hebba. Moim zdaniem najlepszy efekt uzyskujemy gdy te wartości są zbliżone do siebie. W tabelkach można zaobserwować odziałowywanie współczynników na siebie. I tak gdy:

* n > z to wagi zmierzają do nieskończoności;
* n < z to wagi zmierzają do 0;

gdzie n to współczynnik nauczania, a z to współczynnik zapominania.

Przetestowany program moim zdanie zawsze rozwiązywał problem podziały dużych liter alfabetu na grupy podobnych do siebie obiektów.

1. **PRZYKŁADOWA ODPOWIEDŹ PROGRAMU:**

*ODP DLA WSZYSTKICH LITER:*

*A - 90.9982*

*B - 100.438*

*C - 65.0909*

*D - 90.424*

*E - 90.0624*

*F - 70.4767*

*G - 90.5231*

*H - 86.1484*

*I - 53.5069*

*J - 75.2654*

*K - 70.1986*

*L - 60.1662*

*M - 90.0942*

*N - 95.612*

*O - 80.4511*

*P - 75.3477*

*R - 90.231*

*T - 63.5171*

*W - 80.4815*

*Z - 74.5145*

1. **ŹRÓDŁA:**

* <https://edux.pjwstk.edu.pl/mat/273/lec/wyklad3/w3.html>
* <https://www.wikipedia.pl>
* https://www.en.wikipedia.org
* <https://forum.pasja-informatyki.pl/73877/sztuczne-sieci-neuronowe>
* <http://www.if.uz.zgora.pl/~mdudek/siecineuronowe.pdf>
* <https://www.youtube.com/watch?v=Wa_9S20SkKw&t=995s>
* http://home.agh.edu.pl/~horzyk/pracedoktorskie/adrianhorzykpracadoktorska.pdf
* http://pracownik.kul.pl/files/31717/public/Model\_neuronu\_Hebba.pdf
* Materiały udostępnione przez dr inż. Dorota Wilk – Kołodziejczyk

1. **KOD PROGRAMU:**

**Neuron.h**

#pragma once

#include <iostream>

#include <vector>

#include <conio.h>

#include <windows.h>

#include <cstdlib>

#include <fstream>

#include <string>

#include <ctime>

class Neuron

{

private:

int size; // ILOŚ WEJŚĆ

bool \*input; //WEJŚCIA

double \*weight; //WAGI

public:

Neuron(); // Konstruktor domyślny

Neuron(int, bool\*); //Konstruktor z parametrem

~Neuron(); // Destruktor

void showNeuron(); //Wyświetla neuron

//SET

void setSize(int s) { size = s; }

void setInput(int i, bool value) { input[i] = value; }

void setWeight(int i, double value) { weight[i] = value; }

//GET

int getSize() { return size; }

bool getInput(int i) { return input[i]; }

double getWeight(int i) { return weight[i]; }

void setNeuron(bool\*); //Ustawienie wejść neuronu

void teachNeuron(bool \*\*); // Uczy neuron z współczynnikiem zapominania

double getMembranePotential(); // Oblicznie sumy membranowej

long double power(int p); //POTĘGOWANIE NIEZBĘDNE DO WLICZNIA FUNKCJI AKTYWACJI UNIPOLARNEJ

double functionUse(); //FUNKCJA AKTYWACJI

};

**Neuron.cpp**

#include "Neuron.h"

#define constantLearning 0.05 // stała uczęnia

#define constantforgotten 0.001 // stała zapominania

#define amountOfData 20 // ilość danych wejściowych

//#define pixel 35 // ilość pikseli matrycy - u nas 7x5

Neuron::Neuron()

{

size = 20;

input = new bool[size];

weight = new double[size];

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

{

input[i] = 0;

weight[i] = 0.5;

}

}

Neuron::Neuron(int s, bool\* tab)

{

size = s;

input = new bool[size];

weight = new double[size];

srand(time(NULL));

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

{

input[i] = tab[i];

//weight[i] = (rand() % 100) / 100.0 - 0.5;

weight[i] = (rand() % 100) / 200.0;

}

}

Neuron::~Neuron()

{

delete[] input;

delete[] weight;

}

void Neuron::showNeuron()

{

std::cout << "\_\_\_\_\_MÓJ NEURON\_\_\_\_\_" << std::endl;

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

{

std::cout << i << " INPUT: " << input[i] << " WEIGHT: " << weight[i] << std::endl;

}

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

std::cout << "| SUMA: | " << getMembranePotential() << std::endl;

std::cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << std::endl;

}

void Neuron::teachNeuron(bool \*\*tab)

{

double answer = 0.0;

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

{

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

{

input[j] = tab[(size\*i)+j];

}

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

{

answer = getMembranePotential();

weight[j] = weight[j]\*(1- constantforgotten)+(constantLearning \* input[j] \* answer);

}

}

}

double Neuron::getMembranePotential() //OBLICZANIE SUMY MEMBRANOWEJ

{

if (size == 0) //JEŚLI WEJŚCIA NIE ISTNIEJĄ

return -1;

else

{

double sum = 0;

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

{

sum += input[i]\*weight[i] ;

}

return sum;

}

}

long double Neuron::power(int p) //POTĘGOWANIE NIEZBĘDNE DO WLICZNIA FUNKCJI AKTYWACJI UNIPOLARNEJ

{

const double w = getMembranePotential();

double n = 1;

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

{

n \*= p;

}

long double x = getMembranePotential() / n; //BO POTENGUJEMY PRZEZ -5 WIĘDZ DZIELIMY

return x;

}

//double Neuron::function\_use() //FUNKCJA AKTYWACJI

//{

// if (size == 0) //JEŚLI WEJŚCIA NIE ISTNIEJĄ

// return -1;

//

// else

// {

// float e = 2.71; //WARTOŚĆ LOGARYTMU

// float f = (1 / (1 + (power(e)))); //AKTYWACJA UNIPOLARNA

// return f;

// }

//}

double Neuron::functionUse() //FUNKCJA AKTYWACJI

{

long double x = getMembranePotential();

if (x >= 4.5) return 1.0;

//if (x >= 0.0) return 1.0;

return 0.0;

}

void Neuron::setNeuron(bool\* tab)

{

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

{

input[i] = tab[i];

}

}

**Main.cpp**

#include "Neuron.h"

#define n 35 // ilość pikseli matrycy - u nas 7x5

#define amountOfData 20 // ilość danych wejściowych

using namespace std;

void LoadData(bool\*\* tab)

{

double a = 0.0;

fstream file;

file.open("dane.csv", ios::in);

int no\_line = 1;

string line;

int i = 0;

int j = 0;

if (file.good() == false)

{

cout << "Error 0" << endl;

exit(0);

}

while (getline(file, line, ';'))

{

a = atoi(line.c\_str());

no\_line++;

tab[i][j] = a;

j++;

if (j % n == 0 && j != 0)

{

i++;

j = 0;

}

}

file.close();

}

//void LoadScore(bool\* tab)

//{

// double a = 0.0;

// fstream file;

// file.open("score.csv", ios::in);

// int no\_line = 1;

// string line;

// int i = 0;

//

// if (file.good() == false)

// {

// cout << "Error 0" << endl;

// exit(0);

// }

//

// while (getline(file, line, ';'))

// {

// a = atoi(line.c\_str());

// no\_line++;

// tab[i] = a;

// i++;

// }

// file.close();

//}

int main()

{

setlocale(LC\_ALL, ""); // Polskie litery w konsloi

//DANE POMOCNICZE

int index = 0;

double value = 0.0;

int counter = 0;

double answer = 0.0;

bool letter[n] = { 0 };

char alphabet[amountOfData] = { 'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','R','T','W','Z' };

// TABLICE Z DANYMI WEJŚCIOWYMI

bool\*\* data = new bool\*[amountOfData];

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

{

data[i] = new bool[n];

}

bool score[n] = { 0 };

bool input[n] = { 0 };

LoadData(data);

Neuron neuron(n, input); //tworzymy neuron z 25 wejsciami. Wagi ustawiamy losowo

while (true)

{

system("cls");

neuron.showNeuron();

cout << endl << endl;

cout << " ---MENU--- " << endl << endl;

cout << "1) Zmień wybraną wagę " << endl;

cout << "2) Zmień wybrane wejście " << endl;

cout << "3) Ucz neuron " << endl;

cout << "4) Wypisz odpowiedzi dla wszystkich 20 liter" << endl;

cout << "5) Wypisz zestaw uczący " << endl;

cout << "0) Exit " << endl;

cout << "WYBRAŁEŚ: ";

char z;

std::cin >> z;

switch (z)

{

case '1':

{

cout << "Podaj indeks wagi: ";

std::cin >> index;

cout << "Podaj wartosć nowej wagi: ";

std::cin >> value;

neuron.setWeight(index, value);

}

break;

case '2':

{

cout << "Podaj indeks wejścia: ";

std::cin >> index;

cout << "Na jaką wartość logiczną chcesz zmienić ? " << endl << "true -> 1" << endl << "false -> 0" << endl << "WYBRAŁEŚ: ";

std::cin >> value;

if (value == 1)

{

neuron.setInput(index, true);

}

else

{

neuron.setInput(index, false);

}

}

break;

case '0':

{

exit(0);

}

break;

case '4':

{

cout << endl << "ODP DLA WSZYSTKICH LITER: " << endl;

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

{

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

{

letter[j] = data[i][j];

}

neuron.setNeuron(letter);

cout << alphabet[i] << " - " << neuron.getMembranePotential() << endl;

//cout << alphabet[i] << " - " << endl;

}

\_getch();

}

break;

case '5':

{

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

{

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

{

cout << data[i][j];

if (j % 5 == 4 && j != 0)cout << endl;

}

}

cout << endl << "Wciśnij ENTER";

\_getch();

}

break;

case '3':

{

counter++;

neuron.teachNeuron(data);

cout << counter <<" EPOKA UCZENIA" << endl;

\_getch();

}

break;

}

}

//DELOKACJA PAMIĘCI

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

{

delete[] data[i];

}

delete[] data;

data = NULL;

neuron.~Neuron();

//system("pause");

return 0;

}