Jarosław Dutka

Inżynieria Obliczeniowa, rok III

WIMIiIP, AGH

**Sprawozdanie nr 4**

„Uczenie sieci regułą Hebba”

Celem ćwiczenia było poznanie budowy i działania reguły dla sieci jednowarstwowej na przykładzie grupowania liter z i bez współczynnika zapominania.

1. **Przebieg ćwiczenia**

W celu realizacja zadania postanowiono przebudować program napisany podczas realizacji scenariusza nr 2. Program ten był napisany w języku C++ z wykorzystaniem środowiska MS Visual Studio 2015.

W ramach ćwiczeń wygenerowano 10 małych oraz 10 dużych liter alfabetu łacińskiego. Litery wykonano w rozmiarze 4 x 6 pikseli. 

Rys. 1.1 Przykładowa litera wraz z numeracją pól.

Zaimplementowano klasę **Neuron**, która zawiera zmienną typu int size, która zawiera ilość wejść do neuronu. Ponadto klasa zawiera wskaźnik typu bool \* input, który wskazuje na tablicę wejść perceptronu oraz wskaźnik typu double \* weight, który wskazuje na tablicę przechowującą wagi poszczególnych wejść neuronu.

Klasa Neuron zawiera także konstruktor domyślny, który ustawia zmienną size = 24, co odpowiada wielkości matrycy, która przechowuje dane uczące. Konstruktor domyślny losuje także poszczególne wagi z zakresu od 0 do 0.5.

Zaimplementowano dwie metody do uczenia sieci. Obie są wariantami reguły Hebba. Pierwsza z nich – **teach**() - jest wersją reguły bez współczynnika zapominania. Nowa waga jest wyliczana z formuły: ***waga[i] = waga[i] + wejście[i] \* sumaMembranowa***.

Drugi z nich – **forgetTeach**() - jest wariantem reguły Hebba z współczynnikiem zapominia. W tym przypadku wagi są wyliczane ze wzoru: ***waga[i] = waga[i] \* (1 – współczynnikZapominania) + wejście[i] \* sumaMembranowa***.

Ponadto klasa zawiera gamę metod, które służą do odczytu/zapisu danych i obsługi działania klasy.

Do projektu dodano plik **main.cpp**, w którym napisano interfejs programu. Zawierał on trzy funkcje:

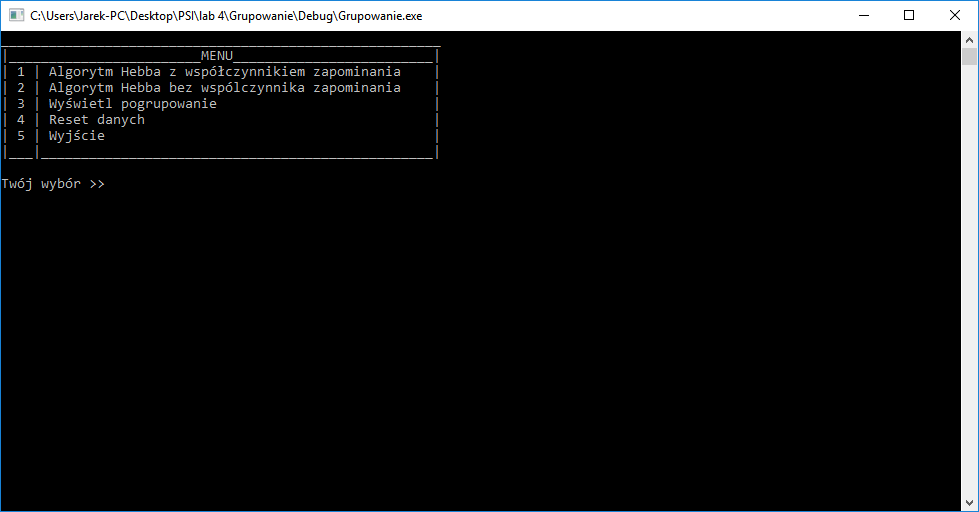
* **int mian()**
* **void loadData(bool \*\* tab)**
* **void assign(bool \*letter, bool\*\* tab, Neuron \*ptr)**

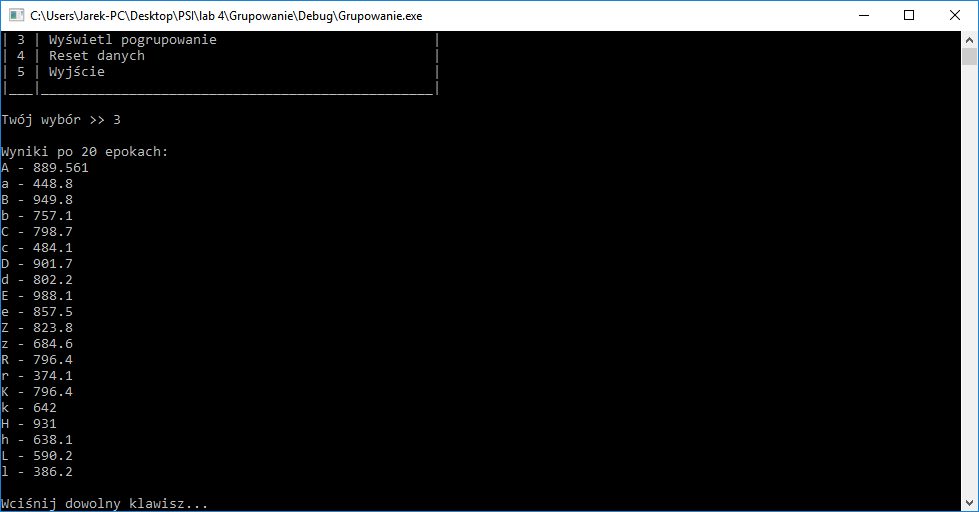
W funkcja **main**() zostało zaimplementowane menu do obsługi aplikacji, z którego można wybrać sposób uczenia (z współczynnikiem zapominania jak również bez), funkcje wyświetlania pogrupowanych liter oraz resetowania danych w aplikacji. Po poprawnym wykonaniu programu funkcja zwraca warość 0.

Funkcja **loadData**(bool \*\* tab) odpowiada za wczytanie danych uczących do aplikacji. Dane uczące zostały załączone do projektu w pliku **„dane.txt”**.

Funkcja **assign(bool \*letter, bool\*\* tab, Neuron \*ptr)** służy do nadania neuronom wejść w etapie uczenia.

Screenshoty interfejsu





1. **Regułą Hebba**

Reguła Hebba jest to jedna z najpopularniejszych przykładów samouczenia się sieci neuronowych. Polega ona  na tym, że sieci pokazuje się kolejne przykłady sygnałów wejściowych, nie podając żadnych informacji o tym, co z tymi sygnałami należy zrobić. Sieć obserwuje otoczenie i odbiera różne sygnały, nikt nie określa jednak, jakie znaczenie mają pokazujące się obiekty i jakie są pomiędzy nimi zależności. Sieć na podstawie obserwacji występujących sygnałów stopniowo sama odkrywa, jakie jest ich znaczenie i również sama ustala zachodzące między sygnałami zależności.

Matematyczna interpretacja tej koncepcji wygląda 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).

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. **Testowanie aplikacji**
2. **Wersja bez współczynnika zapominania**

* Współczynnik uczenia 0,005

**Wyniki po 10 epokach:**

Jeżeli przyjmiemy podział na dwie grupy: I grupa – (0:50), II grupa – (50: 80) grupowanie wygląda następująco:

|  |  |
| --- | --- |
| **I grupa:**  a - 30.95  c - 33.85  z - 47.62  r - 25.61  k - 44.74  h - 44.33  L - 41.4  l - 26.98 | **II grupa:**  A - 61.4736  B - 66.11  b - 52.49  C - 56.17  D - 62.75  d - 55.72  E - 68.71  e - 58.95  Z - 57.42  R - 54.65  K - 54.65  H - 64.33 |

**Wyniki po 20 epokach:**

Przyjmuje podział na dwie grupy: I grupa – (0 – 1000), II grupa – (powyżej 1000).

|  |  |
| --- | --- |
| **I grupa:**  a - 529.6  b - 894.2  C - 942.7  c - 571.5  d - 947.2  Z - 971.6  z - 809  R - 939.4  r - 441.7  K - 939.4  k - 757.7  h - 753.6  L - 696.8  l - 455.8 | **II grupa:**  A - 1050.74  B - 1121  D - 1064  E - 1166  e - 1012  H - 1099 |

**Wyniki po 30 epokach:**

Przyjmuję dwie grupy: I grupa – (o – 14 000), II grupa(powyżej 13 000).

|  |  |
| --- | --- |
| **I grupa:**  a - 7527  b - 1.27e+04  c - 8115  z - 1.148e+04  r - 6272  k - 1.076e+04  h - 1.07e+04  L - 9900  l - 6479 | **II grupa:**  A - 14922.4  B - 1.593e+04  C - 1.34e+04  D - 1.512e+04  d - 1.345e+04  Z - 1.381e+04  R - 1.335e+04  K - 1.335e+04  E - 1.657e+04  e - 1.438e+04  H - 1.562e+04 |

1. **Wersja z współczynnikiem zapominania**

* **Współczynnik zapominania = 0,01**

**Wyniki po 10 epokach:**

Przyjmuję dwie grupy: I grupa - (0, 6), II grupa (powyżej 6).

|  |  |
| --- | --- |
| **I grupa:**  a - 3.35  b - 5.613  C - 5.822  c - 3.648  d - 5.99  z - 5.085  R - 5.794  r - 2.856  K - 5.794  k - 4.677  h - 4.587  L - 4.29  l - 2.796 | **II grupa:**  A - 6.54156  B - 7.029  D - 6.664  E - 7.305  e - 6.267  Z - 6.018  H - 6.848 |

**Wyniki po 20 epokach:**

Przyjmuję dwie grupy: I grupa - (0, 12), II grupa (powyżej 12).

|  |  |
| --- | --- |
| **I grupa:**  a - 6.948  b - 11.5  C - 11.71  c - 7.459  d - 12.64  z - 10.42  R - 11.3  r - 6.152  K - 11.3  k - 9.069  h - 9.068  L - 8.296  l - 5.303 | **II grupa:**  A - 13.3524  B - 14.89  D - 14.06  E - 14.96  e - 12.34  Z - 12.46  H - 13.8 |

**Wyniki po 80 epokach:**

Przyjmuję dwie grupy: I grupa - (0, 2000), II grupa (powyżej 2000).

|  |  |
| --- | --- |
| **I grupa:**  a - 1149  b - 1942  c - 1240  z - 1754  r - 959.2  k - 1647  h - 1637  L - 1514  l - 992.2 | **II grupa:**  A - 2283.11  B - 2438  C - 2051  D - 2314  d - 2057  E - 2537  e - 2198  Z - 2114  R - 2044  K - 2044  H - 2391 |

1. **Wnioski**

Do testów użyto zestawu dużych i małych liter, co pozwoliło na naturalny podział zbioru uczącego na dwa zestawy: litery małe i litery duże. Algorytm bez współczynnika zapominania rozpoznał z 10 % błędem wielkość liter już po 30 epokach uczenia. Jednak współczynniki jakie obliczał dla poszczególnych liter były już relatywnie duże, niewygodne dla oka człowieka. Ten przykład pokazuje, że maszyna jest sama w stanie rozpoznać zestaw cech danych znaków, tylko obserwując dane wejściowe przez wystarczającą liczbę epok.

Żeby liczby, obliczane przez algorytm były mniejszego rzędu, wprowadzono algorytm uczenia ze współczynnikiem zapominania. Podobny rząd wielkość współczynników jak po 30 epokach algorytmu bez współczynnika występuje dopiero przy 100 epoce. Podczas testów, po 80 epokach, program był w stanie rozróżnić wielkość liter z 10 % dokładnością.

Algorytm miał problem z przyporządkowaniem do grupy małych liter dla znaków „e” oraz „d”. Według mnie, jest to skutkiem wyglądu tych znaków. Znak „d” jest podobny do zniekształconych dużych liter takich jak D, B, A. Podobnie ze znakiem „e”. Jednak przy wystarczająco dużej ilości epok, algorytm jest w stanie poradzić sobie także z tym problemem.

1. **Listing kodu**

**Neuron.h**

#pragma once

#include <iostream>

#include <vector>

#include <conio.h>

#include <windows.h>

#include <cstdlib>

#include <fstream>

#include <string>

#include <ctime>

#include <iomanip>

class Neuron

{

private:

int size; //ilość wejść

bool \*input; //wejścia

double \*weight; //wagi

public:

Neuron(); //konstruktor

~Neuron(); //destruktor

void setSize(int s); /\*ustawia ilość wejść do neuronu\*/

void setInput(int i, bool value); /\*ustawia wartość wejścia o indeksie i wartością boolean\*/

void setWeight(int i, double value); /\*ustawia wartość o indeksie i wagi\*/

int getSize(); /\*zwraca ilość wejść do neuronu\*/

bool getInput(int i); /\*zwraca wartość boolean wejścia o indeksie i\*/

double getWeight(int i); /\*zwraca wartość wagi o indeksie i\*/

void setNeuronInputs(bool\*); /\*ustawia wartość wejść neuronu danymi z tablicy, na którą wskazuje wskaźnik\*/

void teach(bool \*\*); /\*funkcja uczenia neuronu bez współczynnika zapominia\*/

void forgetTeach(bool \*\*); /\*funkcja uczenia neuronu z współczynnikiem zapominania\*/

double getMembraneSum(); /\*oblicza sumę membranową (suma += waga[i] \* input[i])\*/

void showNeuron(); /\*wyświetlanie neuronu\*/

};

**Neuron.cpp**

#include "Neuron.h"

#define learningRate 0.005

#define forgetRate 0.01

#define numberOfData 20

using namespace std;

#pragma region Constructors&Destructors

Neuron::Neuron()

{

size = 24;

input = new bool[size];

weight = new double[size];

srand(time(NULL));

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

{

input[i] = 0;

weight[i] = (rand() % 100) / 200.0; //losowanie od 0 do 0.5

}

}

Neuron::~Neuron()

{

delete[] input;

delete[] weight;

}

#pragma endregion

#pragma region Setters

void Neuron::setSize(int s)

{

size = s;

}

void Neuron::setInput(int i, bool value)

{

input[i] = value;

}

void Neuron::setWeight(int i, double value)

{

weight[i] = value;

}

#pragma endregion

#pragma region Getters

int Neuron::getSize()

{

return size;

}

bool Neuron::getInput(int i)

{

return input[i];

}

double Neuron::getWeight(int i)

{

return weight[i];

}

#pragma endregion

#pragma region Functions

void Neuron::setNeuronInputs(bool \* tab)

{

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

{

input[i] = tab[i];

}

}

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

{

long double answer = 0.0;

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

{

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

{

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

}

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

{

answer = getMembraneSum();

weight[j] += (learningRate \* input[j] \* answer);

}

}

}

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

{

long double answer = 0.0;

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

{

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

{

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

}

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

{

answer = getMembraneSum();

weight[j] = weight[j] \* (1 - forgetRate) + (learningRate \* input[j] \* answer);

}

}

}

double Neuron::getMembraneSum()

{

if (size == 0)

return -1;

else

{

long double sum = 0.0;

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

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

return sum;

}

}

void Neuron::showNeuron()

{

{

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

{

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

}

cout << "SUMA = " << getMembraneSum() << endl<<endl;

}

}

#pragma endregion

**Main.cpp**

#include "Neuron.h"

#define numeberOfData 20

#define numberOfPixels 24

using namespace std;

void loadData(bool \*\*tab); /\*wczytanie danych uczących\*/

void assign(bool\* letter, bool \*\*tab, Neuron \*ptr);

int main()

{

setlocale(LC\_ALL, "");

int epochs = 0; //licznik epok

int userAnswer; /\*ilość epok jaką chce przeprowadzić użytkownik\*/

/\*Tworzenie dynamicznej tablicy dwuwymiarowej do przechowania danych uczących\*/

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

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

{

data[i] = new bool[numberOfPixels];

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Neuron neuron; /\*stworzenie domyślnego neuronu\*/

Neuron \*ptr = &neuron;

bool letter[numberOfPixels] = { 0 }; /\*tablica do przechowywania wyników dla poszczególnych liter\*/

char charLetters[numeberOfData] = { 'A','a','B','b','C','c','D','d','E','e','Z','z','R','r','K','k','H','h','L','l' };/\*litery do wydruku\*/

loadData(data); /\*ładowanie danych uczących do tablicy data\*/

neuron.showNeuron();

cout << endl << "Wciśnij dowolny klawisz...";

\_getch();

char znak = '0'; /\*zmienna char do obsługi menu\*/

while (true)

{

system("cls");

cout <<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout <<"|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_MENU\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|" << endl;

cout <<"| 1 | Algorytm Hebba z współczynnikiem zapominania |" << endl

<< "| 2 | Algorytm Hebba bez wspólczynnika zapominania |" << endl

<< "| 3 | Wyświetl pogrupowanie |" << endl

<< "| 4 | Reset danych |" << endl

<< "| 5 | Wyjście |" << endl

<< "|\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|"<<endl;

cout<<endl << "Twój wybór >> "; cin >> znak;

switch (znak)

{

case '1': /\*uczenie z współczynnikiem zapominania\*/

{

assign(letter, data, ptr);

cout << "Ile epok nauczania przeprowadzić? >> "; cin >> userAnswer;

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

{

neuron.forgetTeach(data);

epochs++;

}

neuron.showNeuron();

cout << endl << "Wciśnij dowolny klawisz...";

\_getch();

}

break;

case '2': /\*uczenie bez współczynnika nauczania\*/

{

assign(letter, data, ptr);

cout << "Ile epok nauczania przeprowadzić? >> "; cin >> userAnswer;

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

{

neuron.teach(data);

epochs++;

}

neuron.showNeuron();

cout << endl << "Wciśnij dowolny klawisz...";

\_getch();

}

break;

case '3': /\*wyświetl wyniki po uczeniu\*/

{

cout << endl << "Wyniki po " << epochs << " epokach:" << endl;

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

{

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

{

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

}

neuron.setNeuronInputs(letter);

cout << charLetters[i] << " - " << neuron.getMembraneSum() << setprecision(4) << endl;

}

cout << endl << "Wciśnij dowolny klawisz...";

\_getch();

}

break;

case '4': /\*zresetuj program\*/

{

system("cls");

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

{

letter[i] = 0;

}

neuron = Neuron();

epochs = 0;

loadData(data);

cout << "Resetowanie ustawień i losowanie wag..." << endl;

Sleep(1000);

system("cls");

cout << "Resetowanie ustawień zakończone sukcesem!" << endl;

Sleep(1000);

}

break;

case '0':/\*wykjscie z programu\*/

{

system("cls");

cout << "Wyjście z programu...";

Sleep(1500);

exit(0);

}

break;

default: /\*nepoprawny klawisz\*/

{

cout << "Wciśnięto nieprawidłowy klawisz!" << endl;

cout << endl << "Wciśnij dowolny klawisz...";

\_getch();

}

}

}

//Dealokacja pamięci

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

{

delete[] data[i];

}

delete[] data;

data = NULL;

neuron.~Neuron();

return 0;

}

void loadData(bool \*\*tab)

{

double a = 0.0;

ifstream plik;

plik.open("dane.txt");

string line;

int numberLine = 1;

int i = 0;

int j = 0;

if (!plik.good()) //jesli nie mozna otworzyc pliku to wypisz blad

{

cerr << "Nie można otworzyć pliku źródłowego!" << endl

<< "Error #" << plik.fail() << endl << endl;

system("PAUSE");

exit(1); //jesli nie udalo sie otworzyc pliku to zakoncza dzialanie calego programu

}

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

{

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

numberLine++;

tab[i][j] = a;

j++;

if (j % 24 == 0 && j != 0)

{

i++;

j = 0;

}

}

plik.close();

}

void assign(bool \*letter, bool\*\* tab, Neuron \*ptr)

{

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

{

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

{

letter[j] = tab[i][j]; /\*dla ustawienia wejsc neuronu robie to przypisanie\*/

}

}

ptr->setNeuronInputs(letter);

}