Magdalena Górka

Sprawozdanie nr 4

**Temat ćwiczenia: Uczenie sieci za pomocą reguły Hebba**

**Cel ćwiczenia.**

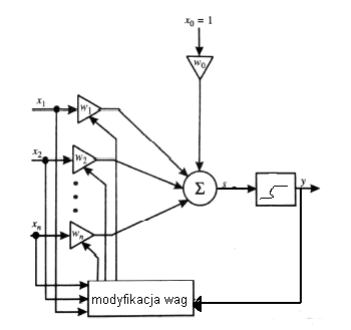
Celem ćwiczenia jest zapoznanie się z regułą Hebba na przykładzie rozpoznawania emotikon.

**Syntetyczny opis budowy oraz wykorzystanego algorytmu uczenia.**

W projekcie użyty został model neuronu Hebba.

Model sieci neuronowej Hebba ma podobną strukturę w porównaniu do modelu Adaline, jednakże różnią się one samą regułą Hebba. Reguła ta występuje w dwóch wersjach: nauka z nauczycielem i bez nauczyciela. Przy implementacji można zastosować, lecz nie trzeba, współczynnik zapominania, który wspomaga uczenie sieci.

*Model Hebba:*



Wadą omawianego algorytmu jest to, iż wartości wag mogą wzrastać do dowolnie dużych liczb, ponieważ w każdym cyklu uczącym następuje proces sumowania.

W celu poprawy stabilności procesu uczenia, podczas aktualizacji wag można wartości zmniejszonej o współczynnik zapominania γ:



Współczynnik zapominania γ zawiera się zwykle w przedziale (0,1) i stanowi najczęściej niewielki procent stałej uczenia η. Przyjęcie dużej wartości γ sprawia, że neuron zapomina większość tego, co zdołał się nauczyć w przeszłości. Najbardziej optymalna wartość współczynnika zapominania to γ = 0.1.

Łączny błąd jednej epoki obliczany jest z następującego wzoru:



Neuron uczy się dopóki błąd nie będzie mniejszy od ustalonego progu.

**Algorytm uczenia**

Proces uczenia przebiega według następującego schematu:

1. Wybór współczynnika uczenia η z zakresu (0; 1>

2. Wylosowanie początkowych wartości wag z zakresu <0; 1)

3. Dla danego zbioru uczącego obliczamy odpowiedź sieci – dla każdego pojedynczego neuronu obliczana jest suma ilorazów sygnałów wejściowych oraz wag.

4. Modyfikacja wag odbywa się według następujących zależności:

a) Reguła Hebba (bez zapominania)

b) Reguła Hebba (z zapominaniem)

Gdzie:

η – współczynnik uczenia

x – zbiór danych wejściowych

γ – współczynnik zapominania

a – suma wyjściowa

5. Obliczenie łącznego błędu

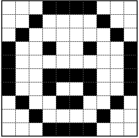
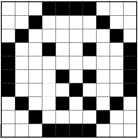
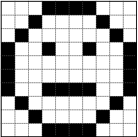
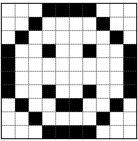


6. Uczenie odbywa się aż błąd nie będzie niższy niż określony próg

**Zastosowane dane**.

Zarówno dane uczące jak i testujące składają się z 4 emotikonek o wymiarach 10x10.

Emotikony uczące.



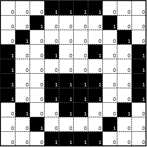
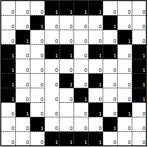
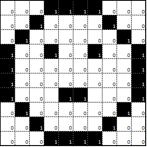
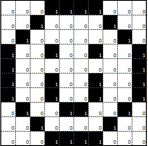








Zmodyfikowane emotikony testujące.



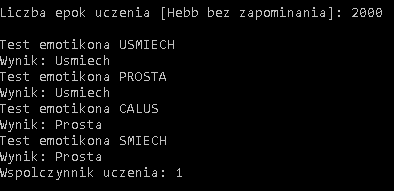


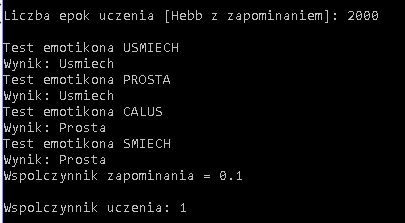




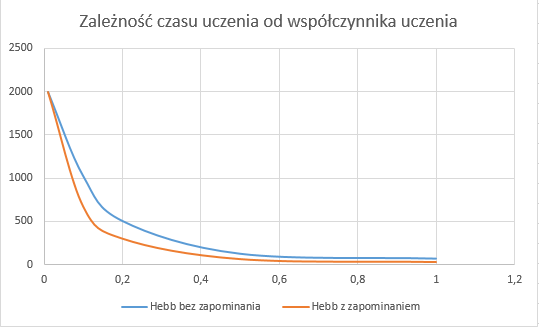


**Przykładowe wyjście programu.**





**Wykresy i analiza.**



Z powyższego wykresu wynika, że efektywność uczenia metodą Hebba jest bardzo związana z współczynnikiem uczenia. Im większy współczynnik uczenia, tym mniej epok potrzebnych do nauki, co jest niezależne od zastosowanej reguły. Dla bardzo małych współczynników uczenia, efektywność nauki dla reguły Hebba jest bardzo niska (tutaj ograniczona do 2000 epok, po czym nauka zostaje przerwana z powodu oszczędności czasu).

W przypadku reguły Hebba ze współczynnikiem zapominania można zauważyć, że posiada ona bardziej efektywne uczenie, jeśli współczynnik zapominania jest niewielki, co pozwala osiągnąć stabilność wag. Dla większych współczynników zapominania sieć uczy się mniej efektywnie, ponieważ to co zostało nauczone, zostaje zapomniane.

*Wykresy przedstawiające jak dobrze neuron radził sobie z rozpoznawaniem danych testowych (przeprowadzono 10 prób)*

Błędy sieci w rozpoznawaniu obrazów z zestawu testującego nie oznaczają tego, że sieć działa źle. Może to wskazywać na jej umiejętność rozpoznawania różnych cech z emotikon, których się nauczyła.

Jako przykład można stwierdzić, że emotikony Śmiech, Prosta i Uśmiech są do siebie podobne, dlatego nie możemy jednoznacznie stwierdzić, że sieć neuronowa podała niepoprawny wynik.

**Wnioski.**

Po dokonaniu analizy powyższych wykresów można jasno stwierdzić, że największy wpływ na proces nauki i rozpoznawania przez sieć podanej emotikony ma współczynnik uczenia. Powodem takiej sytuacji jest fakt, że większy współczynnik uczenia powoduje szybszy przyrost wag.

W modelu z regułą Hebba wpływ na efektywność uczenia sieci ma również obecność lub brak współczynnika zapominania. Ważne jest, żeby dobrać odpowiednio mały współczynnik zapominania tak aby sieć nie zapominała wszystkiego czego przed chwilą się nauczyła.

**KOD.**

#include <iostream>

#include "Neuron.h"

using namespace std;

int main() {

const double LEARNING\_RATE = 0.5;

//Neuron hDontForget(LEARNING\_RATE);

//hDontForget.learnDontForget();

Neuron hForget(LEARNING\_RATE);

hForget.learnForget();

cout << endl << "Wspolczynnik uczenia: " << LEARNING\_RATE << endl << endl;

system("pause");

return 0;

}

#include<ctime>

#include<string>

#include<fstream>

#include<iostream>

#include<string>

#define HOW\_MANY\_BITS 10\*10

#define HOW\_MANY\_EMOTICONS 4

using namespace std;

class Neuron {

public:

int emoticons[4][HOW\_MANY\_BITS]; //Tablica danych uczacych

int emoticonsTest[4][HOW\_MANY\_BITS]; //Tablica danych testujacych

//0 - Usmiech 1 - Prosta 2 - Calus 3 - Smieszek

double learningRate; //Wspolczynnik uczenia

double weights[4][HOW\_MANY\_BITS]; //Wagi

double sumOfInput[HOW\_MANY\_BITS]; //Suma: wagi \* dane uczace

double emoticonTest[HOW\_MANY\_BITS]; //Tablica do przechowywania wyników w czasie testowania

Neuron(double); //Konstruktor

void learnDontForget(); //Uczenie bez zapominania

void learnForget(); //Uczenie z zapominaniem

int activationFunction(double); //Funkcja aktywacji (progowa bipolarna)

void resetSum(); //Ustawienie sum na 0

void test(); //Testowanie rozpoznawania emotikon

void result(); //Wypisywanie wyniku testowania

};

#include "Neuron.h"

Neuron::Neuron(double learningRate) {

this->learningRate = learningRate;

//Zerowanie tablic wag,sum oraz pomocnicza do testow

for (int i = 0; i<HOW\_MANY\_EMOTICONS; i++)

for (int j = 0; j < HOW\_MANY\_BITS; j++) {

weights[i][j] = 0;

sumOfInput[i] = 0;

emoticonTest[i] = 0;

}

//Losowanie wag poczatkowych z przedziału od 0 do 1

srand(time(NULL));

for (int i = 0; i<HOW\_MANY\_BITS; i++) {

weights[0][i] = (double)rand() / (double)RAND\_MAX;//dla emotki 1 dla wszytkich "komponentow" itd dalej dla 2,3,4 emotki

weights[1][i] = (double)rand() / (double)RAND\_MAX;

weights[2][i] = (double)rand() / (double)RAND\_MAX;

weights[3][i] = (double)rand() / (double)RAND\_MAX;

}

//Wczytanie danych uczacych

int whichEmoticon = 0, whichBit = 0, bit;//pomocnicze

fstream file;

file.open("dane.txt");

if (file.is\_open()) {

while (!file.eof()) {

//Jesli wczytano juz wszystkie bity (100=10x10) danej emotikony, przejscie do wczytywania bitow kolejnej

if (whichBit == 225) {

whichBit = 0; whichEmoticon++;

}

//Jesli wczytano wszystkie (4) emotikony, zakoncz wczytywanie

if (whichEmoticon == HOW\_MANY\_EMOTICONS)

break;

file >> bit;

emoticons[whichEmoticon][whichBit] = bit;

whichBit++;

}

file.close();

}

//Wczytanie danych testujacych

fstream file2;

file2.open("dane\_test.txt");

if (file2.is\_open()) {

while (!file2.eof()) {

if (whichBit == 225) {

whichBit = 0; whichEmoticon++;

}

if (whichEmoticon == HOW\_MANY\_EMOTICONS)

break;

file2 >> bit;

emoticonsTest[whichEmoticon][whichBit] = bit;

whichBit++;

}

file2.close();

}

}

//Funkcja aktywacji - progowa unipolarna

int Neuron::activationFunction(double sum) {

return (sum >= 0) ? 1 : 0;//1 gdy x>=wartosc progowa, 0 gdy x<wart.prog, wart.prog=0 - moze byc inna

}

//Ustawienie sum (wejscie\*waga) na wartosc poczatkowa = 0

void Neuron::resetSum() {

int i = 0;

for (i = 0; i<HOW\_MANY\_BITS; i++)

sumOfInput[i] = 0.;

}

//Uczenie regula Hebba (bez zapominania)

void Neuron::learnDontForget() {

int epoch = 0; //Epoka

double globalError = 0.; //Blad globalny

double localErr = 0.; //Blad lokalny

double sum = 0.; //Przechowuje sume wejscie\*waga (sumOfInput)

resetSum(); //Ustawienie wartosci poczatkowych dla wektora sumOfInput

do {

cout << "Epoka: " << epoch << endl;

for (int i = 0; i<HOW\_MANY\_EMOTICONS; ++i) {

globalError = 0.;

for (int j = 0; j<HOW\_MANY\_BITS; ++j) {

sum = sumOfInput[j];

sumOfInput[j] = (weights[i][j] \* emoticons[i][j]);

//Aktualizacja wag

weights[i][j] = weights[i][j] + this->learningRate\*sumOfInput[j] \* emoticons[i][j];

if (localErr == abs(sum - sumOfInput[j]))

break;

localErr = abs(sum - sumOfInput[j]);

globalError = globalError + pow(localErr, 2);

}

}

//cout<<"localError="<<localErr<<endl;

cout<<"globalError="<<globalError<<endl;

epoch++;

} while (globalError != 0 && epoch < 2000);

cout << endl << "Liczba epok uczenia [Hebb bez zapominania]: " << epoch << endl;

test();

}

//Uczenie reguła Hebba (z zapominaniem)

void Neuron::learnForget() {

int epoch = 0; //Epoka

double FORGET\_RATE = 0.1; //Wspolczynnik zapominania

double globalError = 0.; //Blad globalny

double localErr = 0.; //Blad lokalny

double sum = 0.; //Przechowuje sume wejscie\*waga (sumOfInput)

resetSum(); //Ustawienie wartosci poczatkowych dla wektora sumOfInput

do {

//cout << "Epoka: " << epoch << endl;

for (int i = 0; i<HOW\_MANY\_EMOTICONS; ++i) {

globalError = 0.;

for (int j = 0; j<HOW\_MANY\_BITS; ++j) {

sum = sumOfInput[j];

sumOfInput[j] = (weights[i][j] \* emoticons[i][j]);

//Aktualizacja wag

weights[i][j] = weights[i][j] \* (1 - FORGET\_RATE) + this->learningRate\*sumOfInput[j] \* emoticons[i][j];

if (localErr == abs(sum - sumOfInput[j])) break;

localErr = abs(sum - sumOfInput[j]);

globalError = globalError + pow(localErr, 2);

}

}

//cout<<"localError="<<localErr<<endl;

//cout<<"globalError="<<globalError<<endl;

epoch++;

cout<<globalError<<endl;

} while (globalError != 0 && epoch < 2000);

cout << endl << "Liczba epok uczenia [Hebb z zapominaniem]: " << epoch << endl;

test();

cout <<endl<< "Wspolczynnik zapominania = " << FORGET\_RATE <<endl;

}

//Przetestowanie nauczenia danej emotikony

void Neuron::test() {

double globalError = 0.;

double localErr = 0.;

double sum = 0.;

int epoch = 0;

resetSum();

cout << endl << "Test emotikona USMIECH" << endl;

do {

epoch++;

for (int i = 0; i<HOW\_MANY\_BITS; i++) {

sum = sumOfInput[i];

sumOfInput[i] = (weights[0][i] \* emoticonsTest[0][i]);

emoticonTest[i] = activationFunction(sumOfInput[i]);

localErr = abs(sum - sumOfInput[i]);

globalError = globalError + pow(localErr, 2);

}

} while (globalError != 0 && epoch<1);

result();

resetSum();

cout << endl << "Test emotikona PROSTA" << endl;

do {

epoch++;

for (int i = 0; i<HOW\_MANY\_BITS; i++) {

sum = sumOfInput[i];//suma wejsc

sumOfInput[i] = (weights[0][i] \* emoticonsTest[1][i]);//s=w\*dane\_ucz

emoticonTest[i] = activationFunction(sumOfInput[i]);//f(s)

localErr = abs(sum - sumOfInput[i]);

globalError = globalError + pow(localErr, 2);

}

} while (globalError != 0 && epoch<1);

result();

resetSum();

cout << endl << "Test emotikona CALUS" << endl;

do {

epoch++;

for (int i = 0; i<HOW\_MANY\_BITS; i++) {

sum = sumOfInput[i];

sumOfInput[i] = (weights[0][i] \* emoticonsTest[2][i]);

emoticonTest[i] = activationFunction(sumOfInput[i]);

localErr = abs(sum - sumOfInput[i]);

globalError = globalError + pow(localErr, 2);

}

} while (globalError != 0 && epoch<1);

result();

resetSum();

cout << endl << "Test emotikona SMIECH" << endl;

do {

epoch++;

for (int i = 0; i<HOW\_MANY\_BITS; i++) {

sum = sumOfInput[i];

sumOfInput[i] = (weights[0][i] \* emoticonsTest[3][i]);

emoticonTest[i] = activationFunction(sumOfInput[i]);

localErr = abs(sum - sumOfInput[i]);

globalError = globalError + pow(localErr, 2);

}

} while (globalError != 0 && epoch<1);

result();

}

//Sprawdzenie czy dobrze rozpoznaje emotikony

void Neuron::result() {

int tmp[HOW\_MANY\_EMOTICONS];

for (int i = 0; i<HOW\_MANY\_EMOTICONS; i++)

tmp[i] = 0;

for (int i = 0; i<HOW\_MANY\_BITS; ++i) {

if (emoticonTest[i] == emoticons[0][i]) tmp[0]++;

if (emoticonTest[i] == emoticons[1][i]) tmp[1]++;

if (emoticonTest[i] == emoticons[2][i]) tmp[2]++;

if (emoticonTest[i] == emoticons[3][i]) tmp[3]++;

}

if (tmp[0] > tmp[1] && tmp[0] > tmp[2] && tmp[0] > tmp[3])

cout << "Wynik: Usmiech";

if (tmp[1] > tmp[0] && tmp[1] > tmp[2] && tmp[1] > tmp[3])

cout << "Wynik: Prosta";

if (tmp[2] > tmp[0] && tmp[2] > tmp[1] && tmp[2] > tmp[3])

cout << "Wynik: Calus";

if (tmp[3] > tmp[0] && tmp[3] > tmp[1] && tmp[3] > tmp[2])

cout << "Wynik: Smieszek";

}