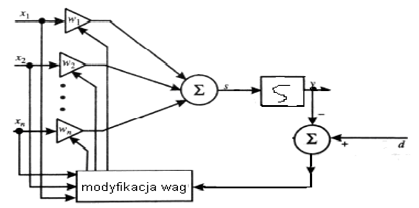
**Mateusz Bożek  
Scenariusz 4**  
**Temat :** Uczenie sieci za pomocą reguły Hebba  
  
**Cel ćwiczenia :** Celem ćwiczenia jest zapoznanie się z regułą Hebba na przykładzie rozpoznawania emitikon.  
  
**1.Syntetyczny opis budowy i wykorzystania sieci i algorytmów uczenia.**  
  
W projekcie użyta została reguła Hebba. W odróżnieniu od modelu Adaline, reguła Hebba ma podobną strukturę jednak rożni się samą regułą. Regułę Hebba możemy podzielić na dwa rodzaje :  
- wersja nauki z nauczycielem   
- wersja nauki bez nauczyciela  
Dodatkowo do reguły Hebba można dodać współczynnik zapominania jednak nie jest ona wymagana.  
  
**MODEL REGUŁY HEBBA**  
  
  
Do wykonania zadania został użyty poniższy algorytm :

Krok 1. Wybór współczynnika uczenia η z zakresu ( 0 ,1 )

Krok 2. Wylosowanie początkowych wartości wag z zakresu ( 0 , 1 )

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

Krok 4. Modyfikacja wag odbywa się według następujących zależności:

- Reguła Hebba (z zapominaniem)

η – współczynnik uczenia

x – zbiór danych wejściowych

γ – współczynnik zapominania

a – suma wyjściowa

- Reguła Hebba (bez zapominania)

η – współczynnik uczenia

x – zbiór danych wejściowych

γ – współczynnik zapominania

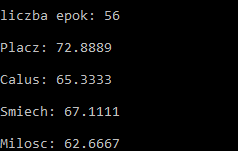
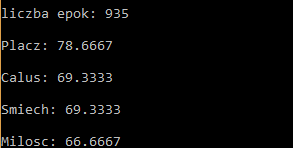
a – suma wyjściowa

Krok 5. Obliczenie łącznego błędu epoki



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

**2.Zestawienie wyników:***Reguła Hebba bez współczynnika zapominania*   
  
  
*Reguła Hebba z współczynnikiem zapominania*



**3.Analiza programu :**

**WYKRES LICZBY EPOK ZALEŻNY OD WSPÓŁCZYNNIKA UCZENIA**

Z powyższego wykresu wynika, że w przypadku reguły Hebba na liczbę epok potrzebną do nauki bardzo mocno wpływa współczynnik uczenia. Gdy współczynnik uczenia jest bardzo mały ( 0.001 ) potrzebna jest bardzo duża liczba epok do nauki ( liczba epok = 5000 ze względu na długi czas nauki ). Natomiast gdy wartość współczynnika rośnie do 0,1 liczba epok wyraźnie spada. Powyżej współczynnika 0,1 obie reguły ( bez współczynnika zapominania oraz ze współczynnikiem zapominania ) uczą się prawie identycznej liczbie epok.

**WYKRES ZALEŻNOŚCI MIĘDZY LICZBĄ EPOK A WSPÓŁCZYNNIKIEM ZAPOMINANIA**Z powyższego wykresu wynika, że im wyższy współczynnik zapominania tym większa liczba epok jest potrzebna do nauki. Dla małych wartości współczynnika zapominania ( do 0,25 ) liczba epok nie rośnie tak dramatycznie jak dla wartości od 0,5 do 1. Na wzrost liczby epok wpływ ma wysoka wartość współczynnika, ponieważ sieć zapomina informację, których nauczyła się wcześniej.  
  
  
Z powyższych wykresów jasno wynika, że sieć dobrze się nauczyła i jest w wstanie rozpoznać prawidłowo, która emotikona została podana. Przy każdej próbie testu, sieć prawidłowo rozpoznała emotikonę podaną do testu. W każdym teście wyniki są do siebie zbliżone. Może być to spowodowane tym, że emotikony są do siebie podobne.  
  
  
  
Powyższy wykres jest zbiorem czterech powyższych testów. Z powyższego wykresu wynika, że sieć lepiej sobie radzi gdy użyta jest reguła Hebba bez współczynnika zapomnienia. **4.Wnioski:**  
  
Po dokonaniu analizy powyższych wykresów można jasno stwierdzić, że największy wpływ na proces nauki i rozpoznawania przez sieć podanej emotikony ma współczynnik uczenia. Współczynnik uczenia im jest większy tym sieć uczy się lepiej i co najważniejsze szybciej. Powodem takiej sytuacji jest fakt, iż im wartość jest większa tym przyrost wag jest szybszy. Reguła Hebba co ważne, odporna jest na zaszumienie. Przy dużym zaszumieniu sieć daje złe wyniki. Na wynik uczenia sieci wpływ także ma rodzaj reguły - bez współczynnika zapominania lub z współczynnikiem zapominania. Reguła bez współczynnika jest efektywniejsza jednak tylko w przypadku gdy wartość tego współczynnika nie jest duża ( do 0,3 ).  
  
  
  
  
  
**5.Listing programu:  
  
Hebbe.h**

#include<stdio.h>

#include<ctime>

#include<stdlib.h>

#include<string>

#include<Eigen/Dense>

#include<fstream>

#include<iostream>

using namespace std;

using namespace Eigen;

#define ROZMIAR 15\*15

#define N 10

#define norma sqrt(225)

#define FILENAME "out/test.xlsx"

#pragma once

class Hebbe {

public:

int epoka; /\* liczba epok \*/

double wspolczynnik\_uczenia; /\* wspolczynnik uczenia \*/

double wspolczynnik\_zapominania; /\* wspolczynnik zapominania \*/

double blad\_globalny;

double blad\_lokalny;

double MSE;

double MAPE;

double pom = 0; // zmienne pomocnicze

double pomt = 0;

MatrixXf dane\_uczace\_emotikony= MatrixXf(225, 4); // dane uczace: 0,x - smiech, 1,x - calus, 2,x - milosc, 3,x - placz

MatrixXf dane\_uczace\_normalizacja = MatrixXf(225, 4); // dane uczace znormalizowane: dane\_uczace\_emotikony/sqrt(255)

MatrixXf wejscie= MatrixXf(225, 1); // dane testowe

MatrixXf macierzSmiechu= MatrixXf(225, 1);

MatrixXf macierzPocalunek= MatrixXf(225, 1);

MatrixXf macierzMilosc= MatrixXf(225, 1);

MatrixXf macierzPlacz = MatrixXf(225, 1);

MatrixXf wagi = MatrixXf(225, 4); // wagi

VectorXf suma\_wag = VectorXf(255); // suma wag

VectorXf atest = VectorXf(255); // wyniki dla danych testujacych

VectorXf deltaw= VectorXf(225); // poprawka wag

Hebbe(); // konstruktor

void zeruj(); // funkcja do zerowania wag

void losowanie\_wag(); // funkcja do losowania wag

void generuj\_wejscie(); // funkcja do generowania wejscia

void ustaw\_wektor\_uczenia(); // funkcja pobierajaca dane z pliku

int fa(double); // funkcja aktywacyjna

void test(); // funkcja testujaca

void porownaj(); // funkcja z wynikami

};  
  
**Hebbe.cpp**#include "Hebbe.h"

Hebbe::Hebbe() {

this->epoka = 0; // liczba epok

this->wspolczynnik\_uczenia = 0.1; // wspolczynnik uczenia

this->wspolczynnik\_zapominania = 0.05; // wspolczynnik zapominania

this->blad\_globalny = 0;

this->blad\_lokalny = 0;

this->MSE = 0;

this->MAPE = 0;

}

void Hebbe::losowanie\_wag() {

srand(time(NULL));

for (int i = 0;i<ROZMIAR;i++) {

wagi(i, 0) = (float)rand() / (float)RAND\_MAX; // losowanie wag

wagi(i, 1) = (float)rand() / (float)RAND\_MAX;

wagi(i, 2) = (float)rand() / (float)RAND\_MAX;

wagi(i, 3) = (float)rand() / (float)RAND\_MAX;

}

}

void Hebbe::generuj\_wejscie() {

int temp;

srand(time(NULL));

for (int i = 0;i<ROZMIAR;i++) {

temp = rand() % 2 - 1; // generowanie losowych danych uczacych

if (temp == 0) wejscie(i) = 1;

else wejscie(i) = -1;

}

for (int i = 0;i<ROZMIAR;i++) { // generowanie zdeformowanych emotikon

macierzSmiechu(i) = dane\_uczace\_emotikony(i, 0);

macierzPocalunek(i) = dane\_uczace\_emotikony(i, 1);

macierzMilosc(i) = dane\_uczace\_emotikony(i, 2);

macierzPlacz(i) = dane\_uczace\_emotikony(i, 3);

}

for (int i = 0;i<45;i++) {

temp = rand() % 225;

macierzMilosc(temp) = !macierzMilosc(temp);

temp = rand() % 225;

macierzSmiechu(temp) = !macierzSmiechu(temp);

}

for (int i = 0;i<90;i++) {

temp = rand() % 225;

macierzPocalunek(temp) = !macierzPocalunek(temp);

temp = rand() % 225;

macierzPlacz(temp) = !macierzPlacz(temp);

}

}

void Hebbe::ustaw\_wektor\_uczenia() { // wczytywanie danych z pliku

int d, j, tmp;

fstream fd;

fd.open("dane\_do\_uczenia.txt");

if (fd.is\_open()) {

j = 0, d = 0;

while (!fd.eof()) {

if (j == 225) {

j = 0; d++;

}

if (d == 4) break;

fd >> tmp;

dane\_uczace\_emotikony(j, d) = tmp;

dane\_uczace\_normalizacja(j, d) = dane\_uczace\_emotikony(j, d) / norma;

++j;

}

fd.close();

}

}

void Hebbe::zeruj() {

for (int i = 0;i<ROZMIAR;i++) {

suma\_wag(i) = 1;

}

}

void Hebbe::test() {

fstream fout; // plik w formacie do excela z zapisywaniem wyników

fout.open(FILENAME, ios::app);

double blad\_globalny = 0;

double blad\_lokalny = 0;

double MSE = 0;

double MAPE = 0;

double pom = 0;

int epoka = 0;

generuj\_wejscie();

do {

epoka++;

for (int j = 0;j<ROZMIAR;++j) {

pom = suma\_wag(j);

//suma\_wag(j) = (wagi(j, 0)\*macierzPlacz(j, 0));

//suma\_wag(j) = (wagi(j, 0)\*macierzMilosc(j, 0));

suma\_wag(j) = (wagi(j, 0)\*macierzPocalunek(j, 0));

//suma\_wag(j) = (wagi(j, 0)\*macierzSmiechu(j, 0));

atest(j) = fa(suma\_wag(j));

blad\_lokalny = abs(pom - suma\_wag(j));

blad\_globalny = blad\_globalny + pow(blad\_lokalny, 2);

}

MSE = pow(blad\_globalny, 2) / (ROZMIAR);

MAPE = (blad\_globalny \* 10 / ROZMIAR);

fout << "MSE: " << MSE << "\tMAPE: " << MAPE << "%\n";

} while (blad\_globalny != 0 && epoka<1);

fout.close();

porownaj();

}

int Hebbe::fa(double a) { // funkcja aktywacji

return (a >= 0) ? 1 : -1;

}

void Hebbe::porownaj() {

int tmp[4];

int result;

fstream fout; // plik w formacie do excela z zapisywaniem wynikow

fout.open(FILENAME, ios::app);

for (int i = 0;i<4;i++) tmp[i] = 0;

for (int i = 0;i<ROZMIAR;++i) {

if (atest(i) == dane\_uczace\_emotikony(i, 0)) tmp[0]++;

if (atest(i) == dane\_uczace\_emotikony(i, 1)) tmp[1]++;

if (atest(i) == dane\_uczace\_emotikony(i, 2)) tmp[2]++;

if (atest(i) == dane\_uczace\_emotikony(i, 3)) tmp[3]++;

}

cout << "\nPlacz: " << (double(tmp[3]) / 225.)\*100. << endl;

cout << "\nCalus: " << (double(tmp[1]) / 225.)\*100. << endl;

cout << "\nSmiech: " << (double(tmp[0]) / 225.)\*100. << endl;

cout << "\nMilosc: " << (double(tmp[2]) / 225.)\*100. << endl;

fout.close();

}  
  
**Source.cpp**#include "Hebbe.h"

int main() {

Hebbe hebbe;

cout << "Rozpoczecie nauki wedlug reguly Hebba." << endl;

hebbe.zeruj(); /\* ustawienie wartosci domyslnych dla wektora a \*/

hebbe.losowanie\_wag(); /\* wygenerowanie wag \*/

hebbe.ustaw\_wektor\_uczenia(); /\* ustawienie wektorow uczacych \*/

do {

cout << "epoka: " << hebbe.epoka << endl;

// BEZ WSPOLCZYNNIKA ZAPOMINANIA

for (int i = 0;i<4;++i) {

hebbe.blad\_globalny = 0;

for (int j = 0;j<ROZMIAR;++j) {

hebbe.pom = hebbe.suma\_wag(j); // ustalenie wag

hebbe.suma\_wag(j) = (hebbe.wagi(j, i)\*hebbe.dane\_uczace\_emotikony(j, i)); // ustalenie sumy wag = waga \* dana wejsciowa

hebbe.wagi(j, i) = hebbe.wagi(j, i) + hebbe.wspolczynnik\_uczenia\*hebbe.suma\_wag(j)\*hebbe.dane\_uczace\_emotikony(j, i); //zmiana wagi

if (hebbe.blad\_lokalny == abs(hebbe.pom - hebbe.suma\_wag(j))) break; //

hebbe.blad\_lokalny = abs(hebbe.pom - hebbe.suma\_wag(j)); // ustalenie bledu lokalnego

hebbe.blad\_globalny = hebbe.blad\_globalny + pow(hebbe.blad\_lokalny, 2); // ustalenie bledu globalnego

}

hebbe.MSE = pow(hebbe.blad\_globalny, 2) / ROZMIAR;

hebbe.MAPE = (hebbe.blad\_globalny \* 10) / ROZMIAR;

cout << " MSE: " << hebbe.MSE << "\tMAPE: " << hebbe.MAPE << "%\n";

}

// ZE WSPOLCZYNNIKIEM ZAPOMINANIA

for( int i=0;i<4;++i){

hebbe.blad\_globalny = 0;

for( int j=0;j<ROZMIAR;++j){

hebbe.pom = hebbe.suma\_wag(j);

hebbe.suma\_wag(j) = (hebbe.wagi(j,i)\*hebbe.dane\_uczace\_emotikony(j,i));

hebbe.wagi(j,i) = hebbe.wagi(j,i)\*hebbe.wspolczynnik\_zapominania + hebbe.wspolczynnik\_uczenia\*hebbe.suma\_wag(j)\*hebbe.dane\_uczace\_emotikony(j,i); // zmiana wag

if(hebbe.blad\_lokalny==abs(hebbe.pom- hebbe.suma\_wag(j))) break;

hebbe.blad\_lokalny = abs(hebbe.pom - hebbe.suma\_wag(j));

hebbe.blad\_globalny = hebbe.blad\_globalny + pow(hebbe.blad\_lokalny,2);

}

hebbe.MSE = pow(hebbe.blad\_globalny,2)/(ROZMIAR);

hebbe.MAPE = (hebbe.blad\_globalny\*10/ROZMIAR);

cout << "i:" << i << " MSE: " << hebbe.MSE << "\tMAPE: " << hebbe.MAPE << "%\n";

}

hebbe.epoka++;

} while (hebbe.blad\_globalny != 0 && hebbe.epoka<1000);

cout << "\nliczba epok: " << hebbe.epoka << endl;

hebbe.test();

system("pause");

return 0;

}