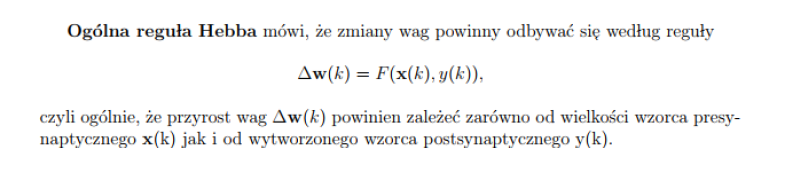
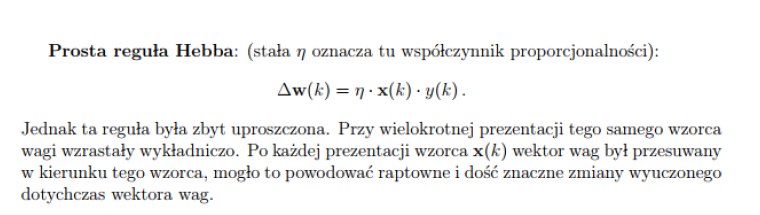
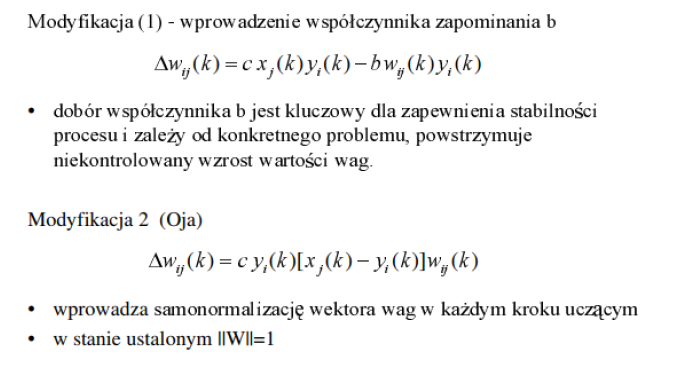
Sprawozdanie z zagadnienia nr4.

Opis syntetyczny:

Reguła Hebba jest jedną z popularnych metod samouczenia sieci neuronowych.  
Zasada działania polega na tym, że sieci przedstawia się kolejne przykłady sygnałów wejściowych , jednak nie mówi się informacji co z tymi sygnałami zrobić. Sieć odbiera różne sygnały i obserwuje otoczenie. Na podstawie napływających danych sieć dedukuje jakie mają one znaczenie i ustala zachodzące między nimi zależności.







**Dane uczące:**

Wygenerowane zostały matryce w rozmiarze 15x15 dla 4 emotikon: śmiech,płacz,miłość,całus.  
Przykładowa matryca:

**-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1**

**-1 -1 -1 -1 -1**  1 1 1 1 1 **-1 -1 -1 -1 -1**

**-1 -1 -1** 1 1 **-1 -1 -1 -1 -1** 1 1 **-1 -1 -1**

**-1 -1** 1 **-1 -1 -1 -1 -1 -1 -1 -1 -1** 1 **-1 -1**

**-1 -1** 1 **-1** 1  **-1 -1 -1 -1 -1** 1 **-1**  1 **-1 -1**

**-1** 1 -**1** 1 **-1 -1 -1 -1 -1 -1 -1** 1  **-1** 1 **-1**

**-1** 1 **-1 -1** 1 1 1  **-1** 1 1 1 **-1 -1** 1 **-1**

**-1** 1 -**1 -1** 1 **-1 -1 -1 -1 -1** 1 -**1 -1** 1 **-1**

**-1** 1 **-1** 1 1 **-1 -1 -1 -1 -1** 1 1 **-1**  1 **-1**

**-1** 1 1 1 **-1** 1 1 1 1 1 **-1** 1 1 1 **-1**

**-1 -1** 1 1 **-1** 1 1 1 1 1 **-1** 1 1 **-1 -1**

**-1 -1**  1 **-1 -1 -1** 1 1 1 **-1 -1 -1** 1 **-1 -1**

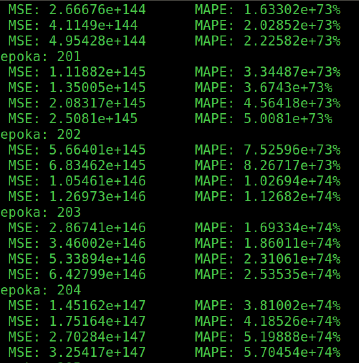
**-1 -1 -1** 1 1 **-1 -1 -1 -1 -1** 1 1 **-1 -1 -1**

**-1 -1 -1 -1 -1**  1 1 1 1 1 **-1 -1 -1 -1 -1**

**-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1**

Do testów używano zaszumionych matryc gdzie poprawność pojedynczych pikseli była na poziomie 60% i 80% względem oryginału.

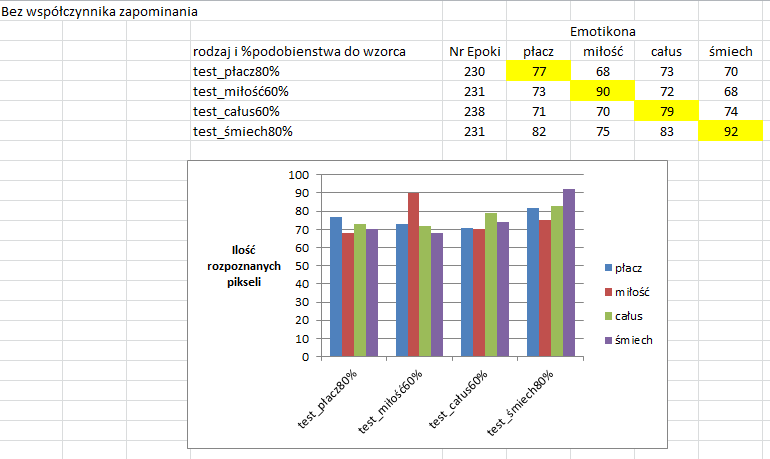
**Zestawienie danych + przykładowe wydruki:**

****

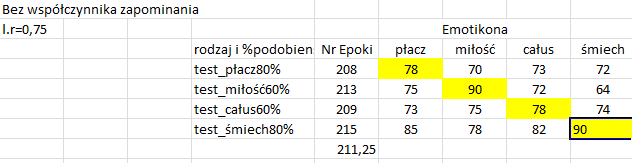
Przykładowy zrzut wykonany w trakcie wykonywania procesu uczenia, na którym widnieje błąd MSE i MAPE.

**Dane uzyskane :**

1. **Bez współczynnika zapominania**

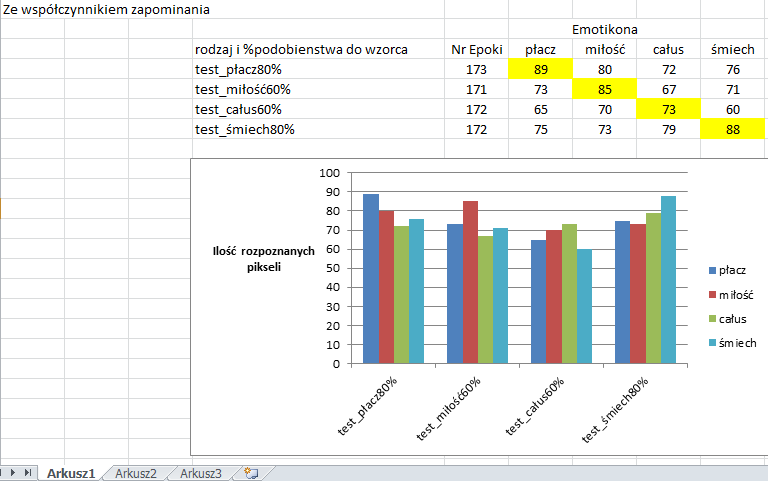
****

Przy współczynniku uczenia = 0,5 proces uczenia trwał około 233 epoki. Dla zniekształconych emotikon testowych, siec rozpoznała wszystkie obrazy poprawnie.

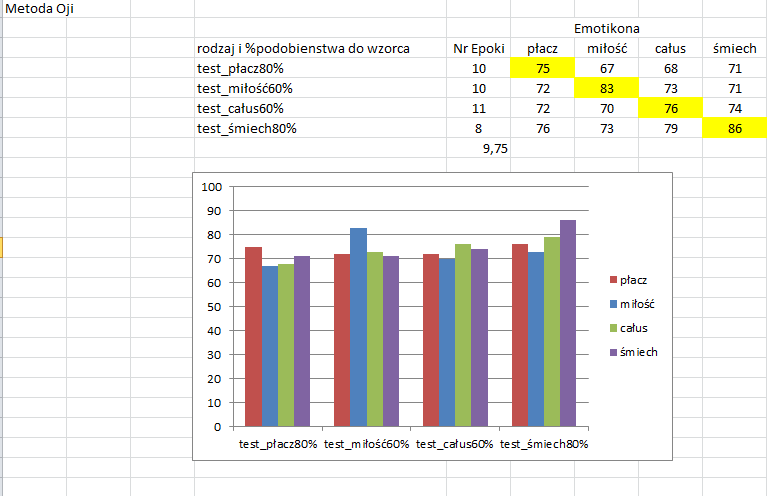


Dla porównania zwiększony współczynnik uczenia =0,75, który przyśpieszył proces uczenia o około 22 epoki.

1. **Ze współczynnikiem zapominania**

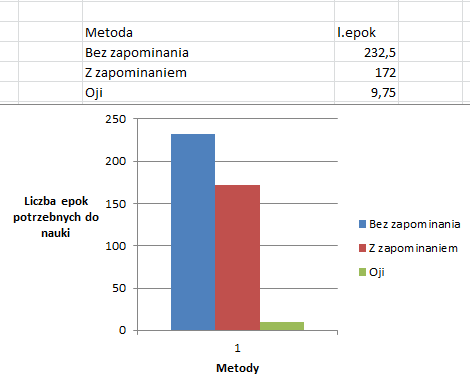
****Przy współczynniku uczenia=0,5 i współczynniku zapominania =0,05 czas uczenia trwał ok. 172 epoki. Rozpoznano wszystkie obrazy poprawnie, choć jak widać z inną dokładnością.

1. **Uczenie metodą Oji**

****

Przy współczynniku uczenia =0,5 sieć metoda Oji uczyła się w ok. 10 epok. Obrazy zostały rozpoznane poprawnie szybciej lecz z mniejszą precyzją.

**Zestawienie wyników:**Porównanie prędkości uczenia sieci w zależności od wybranej metody uczenia.

****

Zestawienie dokładności rozpoznawania obrazów w zależności od wybranej metody.

**Wnioski:**

**-** Metoda Hebba polega na wzmacnianiu połączeń między źródłami silnymi i osłabieniu tych słabszych

-Modyfikacja tej metody w metodę Oji w znacznym stopniu przyśpiesza proces uczenia się

-współczynnik zapominania ogranicza wzrost wektora wag (pozwala mieć nad nim kontrole)

**-** wielkości współczynników uczenia i zapominania mają wpływ na szybkość i dokładność uczenia  
-testy polegały na podaniu sieci zniekształconego obrazu i próbie jego rozpoznania  
-metodą najszybszą okazała się być metoda Oji ,lecz najdokładniejszą metoda Hebba bez współczynnika zapominania

**Listing kodu:**

void

generateInput(){

srand(time(NULL));

int i, j, tmp;

for (i = 0; i<SIZE; i++){

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

if (tmp == 0) input(i) = 1;

else input(i) = -1;

}

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

inputSmiech80pr(i) = vec\_emoticons(i, 0);

inputCalus60pr(i) = vec\_emoticons(i, 1);

inputMilosc80pr(i) = vec\_emoticons(i, 2);

inputPlacz60pr(i) = vec\_emoticons(i, 3);

}

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

j = rand() % 225;

inputMilosc80pr(j) = !inputMilosc80pr(j);

j = rand() % 225;

inputSmiech80pr(j) = !inputSmiech80pr(j);

}

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

j = rand() % 225;

inputCalus60pr(j) = !inputCalus60pr(j);

j = rand() % 225;

inputPlacz60pr(j) = !inputPlacz60pr(j);

}

}

void

generateWeights(){

srand(time(NULL));

int i;

for (i = 0; i<SIZE; i++){

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

w(i, 1) = (float)rand() / (float)RAND\_MAX; // losowanie wag

w(i, 2) = (float)rand() / (float)RAND\_MAX; // losowanie wag

w(i, 3) = (float)rand() / (float)RAND\_MAX; // losowanie wag

}

}

int

main(){

int i, j;

int epoka = 0; // liczba epok

const double LEARNING\_RATE = 0.5; /\* wspolczynnik uczenia \*/

const double FORGET\_RATE = 0.05; /\* wspolczynnik zapominania \*/

double globalError = 0.; /\* bledy \*/

double localError = 0.;

double MSE = 0.;

double MAPE = 0.;

double pom = 0.; /\* zmienne pomocnicze \*/

double pomt = 0.;

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

//UCZENIE WG HEBBA

generateWeights(); /\* wygenerowanie wag \*/

setLearnVectors(); /\* ustawienie wektorow uczacych \*/

do{

cout << "epoka: " << epoka << endl;

/\* BEZ WSPOLCZYNNIKA ZAPOMINANIA

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

globalError = 0.;

for (j = 0; j<SIZE; ++j){

pom = a(j);

a(j) = (w(j, i)\*vec\_emoticons(j, i));

w(j, i) = w(j, i) + LEARNING\_RATE\*a(j)\*vec\_emoticons(j, i);

if (localError == abs(pom - a(j))) break;

localError = abs(pom - a(j));

globalError = globalError + pow(localError, 2);

}

MSE = pow(globalError, 2) / SIZE;

MAPE = (globalError \* 10) / SIZE;

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

}\*/

// ZE WSPOLCZYNNIKIEM ZAPOMINANIA

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

globalError = 0.;

for(j=0;j<SIZE;++j){

pom = a(j);

a(j) = (w(j,i)\*vec\_emoticons(j,i));

w(j,i) = w(j,i)\*FORGET\_RATE + LEARNING\_RATE\*a(j)\*vec\_emoticons(j,i);

if(localError==abs(pom-a(j))) break;

localError = abs(pom - a(j));

globalError = globalError + pow(localError,2);

}

MSE = pow(globalError,2)/(SIZE);

MAPE = (globalError\*10/SIZE);

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

}

// REGUŁA OJI

/\*for(i=0;i<4;++i){

globalError = 0.;

for(j=0;j<SIZE;++j){

pom = a(j);

a(j) = (w(j,i)\*vec\_emoticons(j,i));

w(j,i) = w(j,i) + (LEARNING\_RATE\*a(j)\*(vec\_emoticons(j,i)-a(j)\*w(j,i)));

if(localError==abs(pom-a(j))) break;

localError = abs(pom - a(j));

globalError = globalError + pow(localError,2);

}

MSE = pow(globalError,2)/(SIZE);

MAPE = (globalError\*10/SIZE);

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

}\*/

epoka++;

} while (globalError != 0 && epoka<1000);

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

test();

return 0;

}