Weronika Muszyńska, gr. 03

**Sprawozdanie z ćwiczenia nr 4**

**Temat ćwiczenia:**

Uczenie sieci regułą Hebba

**Celem ćwiczenia** jest poznanie działania reguły Hebba dla sieci jednowarstwowej na

przykładzie grupowania liter alfabetu.

**Syntetyczny opis budowy sieci jednowarstwowej**

Sieć neuronowa – sieć, w którym wyjścia każdego neuronu są połączone, poprzez wagi, z wejściami wszystkich neuronów, także z jego własnym wejściem.

Sieci jednokierunkowe to sieci neuronowe, w których nie występuje sprzężenie zwrotne, czyli pojedynczy wzorzec lub sygnał przechodzi przez każdy neuron dokładnie raz w swoim cyklu. Najprostszą siecią neuronową jest pojedynczy perceptron progowy, opracowany przez McCullocha i Pittsa w roku 1943**.**

Elementarna konfoguracja m neuronów połączonych z n wejściami wygląda następująco :



***Rys.1 Schemat połączeń jednowarstwowej sieci neuronowej.***

Wektory wejściowy i wyjściowy mają postać :

x= y =

Wagi wij łączą neuron I z wejściem j. Pobudzeniei-tego neuronu można zapisać jako :

neti = i=1,2,…,m

**Reguła Hebba**

Reguła Hebba przyjmuje, że sygnałem uczącym jest sygnał wyjściowy neuronu, czyli

r=yi = f()

Przyrost ∆wi  wektora wag wynosi

∆wi  = cyix = cf()x

Każda ze składowych wij zmienia się więc o

∆wi j = cyixj , j= 1,2,...,n

Wymaga się wstępnego ustawienia wag na wartości przypadkowe z otoczenia wi=0.

Opierając się na zasadzie tworzenia się odruchów warunkowych, Hebb wprowadził następującą zasadę. Jeśli aktywny neuron A jest cyklicznie pobudzany przez neuron B, to staje się on jeszcze bardziej czuły na pobudzenie tego neuronu. Jeśli przez *x*A i *x*Boznaczymy stany aktywacji neuronów A i B, a przez *w*AB - wagę ich połączenia synaptycznego, to powyższą regułę można zapisać w postaci następującego równania:

Rozmiar: 2068 bajtów

gdzie  oznacza pewną stałą dodatnią, sterującą procesem uczenia.

Reguła Hebba posiada istotną wadę, mianowicie prowadzi do procesu rozbieżnego. Aby to udowodnić, rozważmy liniowy element przetwarzający. Jego stan aktywacji *x* w chwili *k* jest równy:

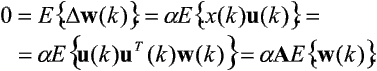
Rozmiar: 1571 bajtów

gdzie **w**(*k*) oznacza wektor wag wejściowych połączeń synaptycznych elementu, **u**(*k*) - wektor wejściowy należący do pewnego zbioru  wektorów uczących, wzięty z tego zbioru zgodnie z rozkładem prawdopodobieństwa, oraz podany na wejście elementu w chwili *k*-tej.

Zgodnie z ogólną regułą kolejne prezentacje wzorców ze zbioru  zmieniają wektor wag o przyrost **W**(*k*) określony zależnością:

Rozmiar: 1616 bajtów

Dalej będziemy zakładać, że elementy ciągu **u**(*k*) są niezależne (tzn. wynik losowania wzorca ze zbioru  w danej chwili nie zależy od wyników losowań w innych chwilach). Warunkiem zbieżności procesu uczenia jest zerowanie się (od pewnego momentu) średniej zmiany wag:



gdzie:

Rozmiar: 1706 bajtów

jest macierzą korelacji obrazów wejściowych, a E. operatorem wartości oczekiwanej.

Po podaniu do sieci neuronowej każdego kolejnego zestawu sygnałów wejściowych tworzy się w tej sieci pewien rozkład sygnałów wyjściowych - niektóre neurony sieci są pobudzone bardzo silnie, inne słabiej, a jeszcze inne mają sygnały wyjściowe wręcz ujemne. Interpretacja tych zachowań może być taka, że niektóre neurony „rozpoznają” podawane sygnały jako „własne” (czyli takie, które są skłonne akceptować), inne traktują je „obojętnie”, zaś jeszcze u innych neuronów wzbudzają one wręcz „awersję”. Po ustaleniu się sygnałów wyjściowych wszystkich neuronów w całej sieci - wszystkie wagi wszystkich neuronów są zmieniane, przy czym wielkość odpowiedniej zmiany wyznaczana jest na podstawie iloczynu sygnału wejściowego, wchodzącego na dane wejście (to którego wagę zmieniamy) i sygnału wyjściowego produkowanego przez neuron, w którym modyfikujemy wagi. Łatwo zauważyć, że jest to właśnie realizacja postulatu Hebba - w efekcie opisanego wyżej algorytmu połączenia między źródłami silnych sygnałów i neuronami które na nie silnie reagują są wzmacniane.

*Rys.2 Reguła uczenia Hebba bez nauczyciela*

Sygnałem uczącym jest sygnał wyjściowy neuronu:



Korekta wektora wag:





Z zasady tej wynika, iż dodatnia wartość składnika korelacyjnego yixj powoduje wzrost wagi wij, czyli silniejszą reakcję neuronu przy kolejnej prezentacji tego samego obrazu wejściowego. Obrazy wejściowe, które często się powtarzają dają zatem silniejszą odpowiedź na wyjściu.

Regułę uczenia Hebba nazywa się także także uczeniem korelacyjnym. Celem jest takie dopasowanie wag, aby uzyskać najlepszą korelację między sygnałami wejściowymi, a zapamiętanym w formie wartości wag wzorcem, na który określony neuron ma zareagować.

Uczenie neuronu z zastosowaniem reguły Hebba może być prowadzone również z nauczycielem. W uczeniu z nauczycielem wartość sygnału wyjściowego a zastępuje się wartością zadaną t.



**Zestaw danych uczących**

Dane uczące składają się z tablicy 20 liczb reprezentujących 20 wielkich liter:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

new double[25] {1,1,1,1,1,1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1};

new double[25] {1,1,1,0,0,1,0,0,1,0,1,1,1,0,0,1,0,0,1,0,1,1,1,0,0},

new double[25] {1,1,1,1,1,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,1,1},

new double[25] {1,1,1,0,0,1,0,1,0,0,1,0,1,0,0,1,0,1,0,0,1,1,1,0,0},

new double[25] {1,1,1,1,1,1,0,0,0,0,1,1,1,1,1,1,0,0,0,0,1,1,1,1,1},

new double[25] {1,1,1,1,1,1,0,0,0,0,1,1,1,1,1,1,0,0,0,0,1,0,0,0,0},

new double[25] {1,1,1,1,0,1,0,0,0,0,1,0,1,1,0,1,0,0,1,0,1,1,1,1,0},

new double[25] {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},

new double[25] {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},

new double[25] {1,1,1,0,0,0,0,1,0,0,0,0,1,0,0,1,0,1,0,0,1,1,1,0,0},

new double[25] {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},

new double[25] {1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,1,0,0},

new double[25] {1,1,0,1,1,1,0,1,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1},

new double[25] {1,1,0,0,1,1,1,1,0,1,1,0,1,1,1,1,0,0,1,1,1,0,0,0,1},

new double[25] {1,1,1,1,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,1,1,1,1},

new double[25] {1,1,1,1,0,1,0,0,1,0,1,1,1,1,0,1,0,0,0,0,1,0,0,0,0},

new double[25] {1,1,1,1,0,1,0,0,1,0,1,1,1,1,0,1,0,1,0,0,1,0,0,1,0},

new double[25] {1,1,1,0,0,1,0,0,0,0,1,1,1,0,0,0,0,1,0,0,1,1,1,0,0},

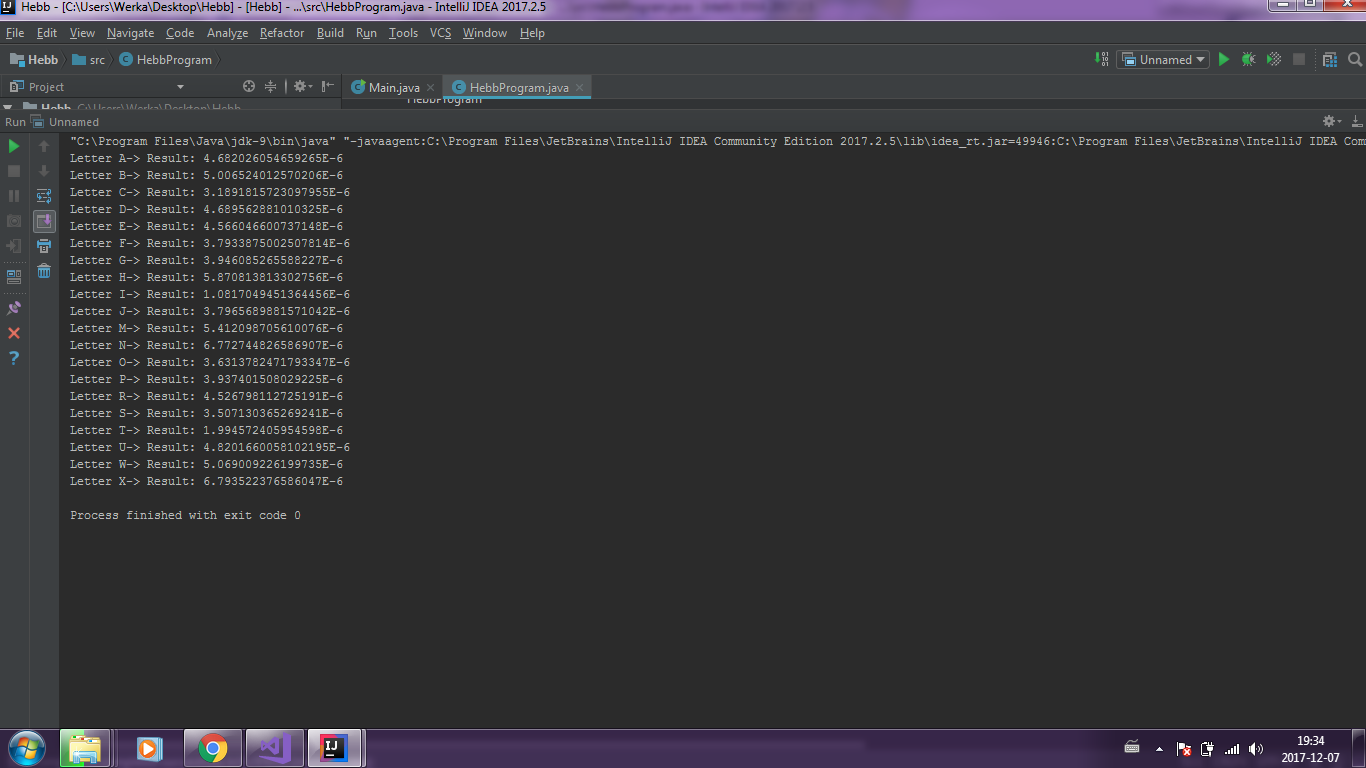
new double[25] {1,1,1,1,1,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0},

new double[25] {1,0,0,1,0,1,0,0,1,0,1,0,0,1,0,1,0,0,1,0,1,1,1,1,0}

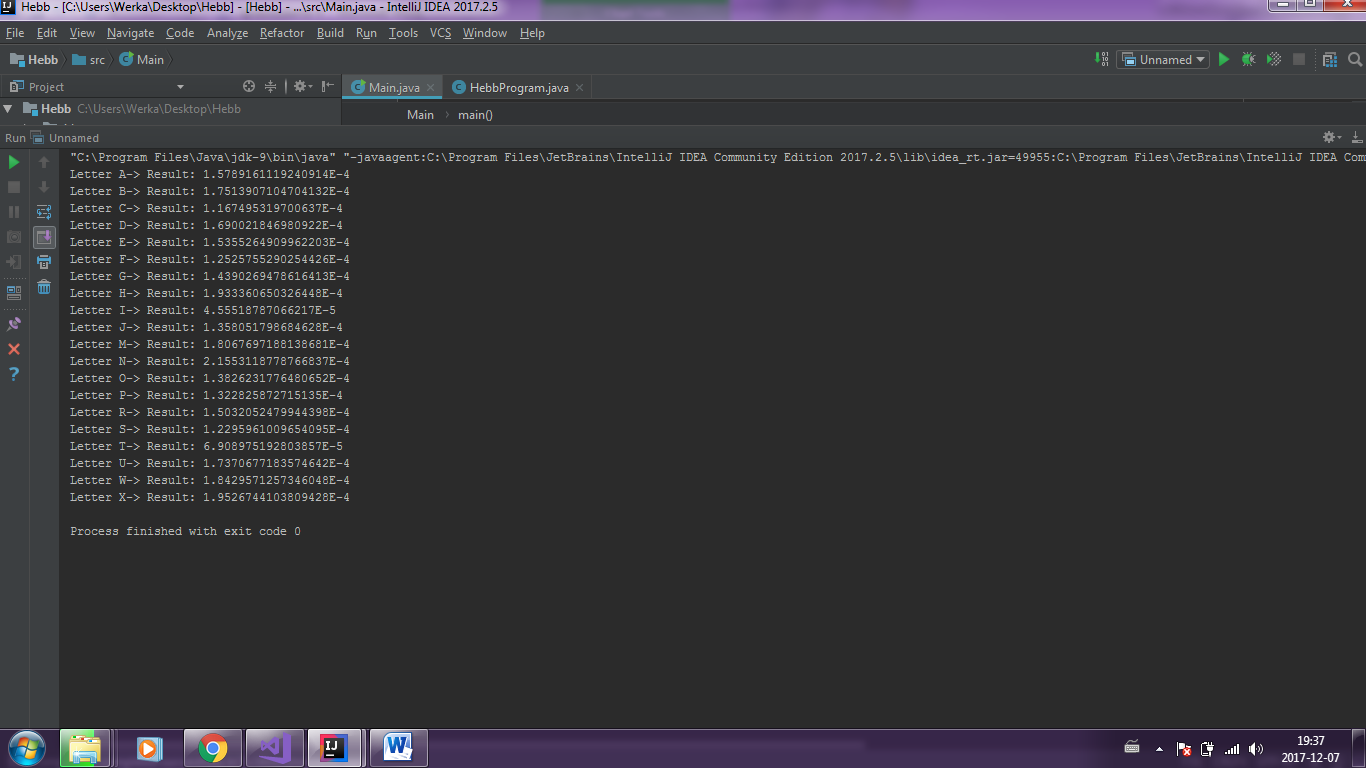
W programie użyłam funkcję jako funkcję aktywacji

Każda litera reprezentowana jest jako tablica zawierająca 5x5 liczb reprezentującą piksele

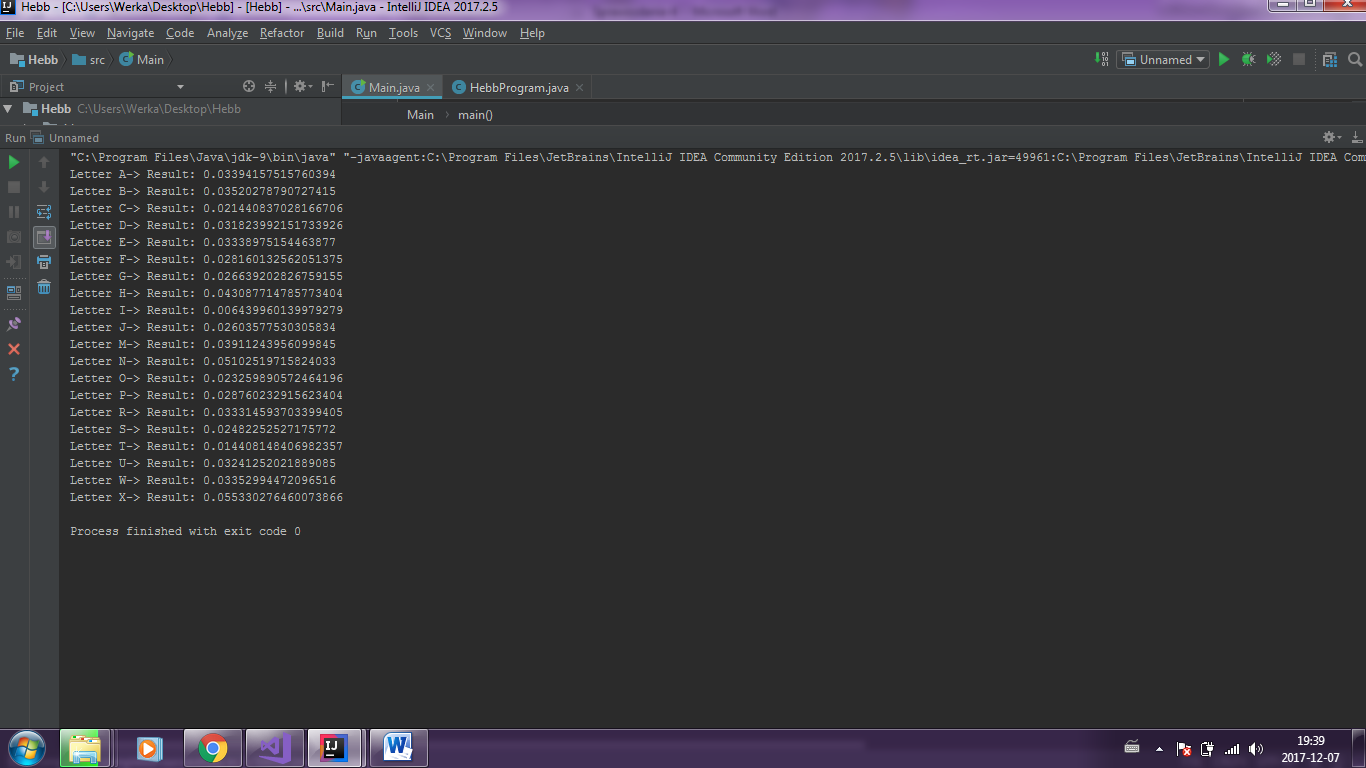
**Uzyskane wyniki :**



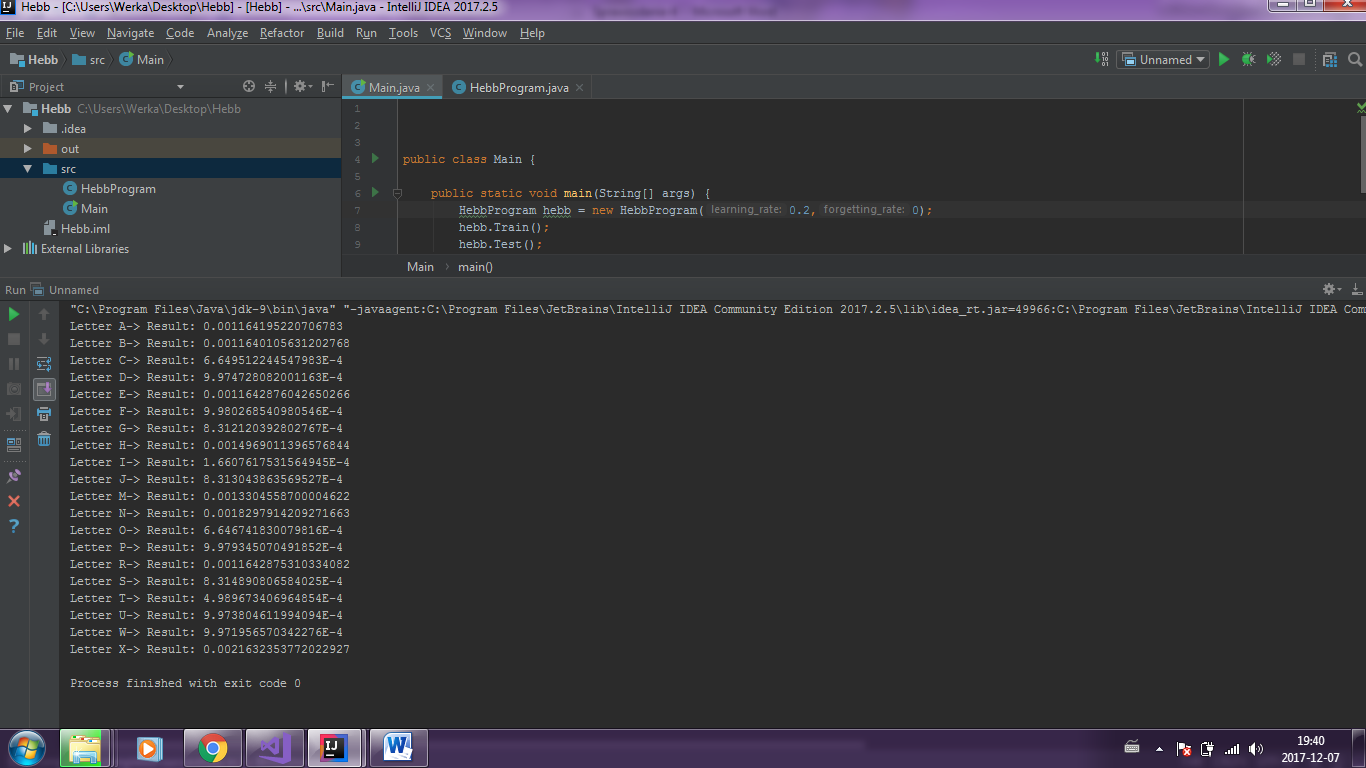
**Rys. 3** **wspóczynnik uczenia wynosi 0.1 i współczynnik zapominania jest równy 0.1**



**Rys. 4** **wspóczynnik uczenia wynosi 0.1 i współczynnik zapominania jest równy 0.2**



**Rys. 5** **wspóczynnik uczenia wynosi 0.2 i współczynnik zapominania jest równy 0.1**



**Rys. 6** **wspóczynnik uczenia wynosi 0.2 i współczynnik zapominania jest równy 0**

**Zestawienie wyników :**

**Rys. 7 Wyniki dla liter od A do X, gdzie wspóczynnik uczenia wynosi 0.1 i współczynnik zapominania jest równy 0.1**

**Rys. 8 Wyniki dla liter od A do X, gdzie wspóczynnik uczenia wynosi 0.1 i współczynnik zapominania jest równy 0.2**

**Rys.9 Wyniki dla liter od A do X, gdzie wspóczynnik uczenia wynosi 0.2 i współczynnik zapominania jest równy 0.1**

**Rys. 10 Wyniki dla liter od A do X, gdzie wspóczynnik uczenia wynosi 0.2 i współczynnik zapominania jest równy 0**

**Wnioski:**

Uczenie sieci wypada gorzej, gdy nie stosuje się współczynnika zapominania, ponieważ wagi rosną wtedy bardzo szybko i uniemożliwiona jest ich stabilizacja.

Im mniejszy współczynnik uczenia tym dokładniejszy wynik. Aby otrzymać wiarygodne dane musimy odpowiednio dobrać współczynniki uczenia a także bezwładności

Wraz ze zwiększaniem wartości współczynnika uczenia zmniejsza się liczba iteracji, co ma wpływ na działanie programu i powoduje uzyskanie błędnych wyników. Aby zredukować liczbę błędnych wyników można zwiększyć liczbę iteracji.

Dobór wag ma bezpośredni wpływ na działanie perceptronów w sieci, w zależności od ich wartości zmienia się poprawność wyników i liczba iteracji. Wartości wag mają największy wpływ na efekt końcowy, często wagi są ustalane losowo.

Dane uczące wpływają na poprawność uczenia perceptronu, przy zbyt małej ilości otrzymujemy błędne wyniki. Aby uzyskać lepsze wyniki powinniśmy dostarczyć wystarczającą liczbę danych wejściowych

Bardzo ważny jest odpowiedni dobór współczynnika uczenia, współczynnika zapominania, wag oraz liczby danych uczących, gdyż mają bezpośredni wpływ na działanie sieci.

**Źródła:**

1. **S. Osowski. Sieci neuronowe w ujęciu algorytmicznym. WNT, Warszawa 1997.**
2. **J. Żurada, M. Barski, W. Jędruch. Sztuczne sieci neuronowe. PWN, Warszawa 1996.**

**Listing kodu :**

**Main.java :**

**public class Main {**

**public static void main(String[] args) {**

**HebbProgram hebb = new HebbProgram(0.1,0.1);**

**hebb.Train();**

**hebb.Test();**

**}**

**}**

**HebbProgram.java:**

**import java.util.Random;**

**public class HebbProgram {**

**public double weight[];**

**double learning\_rate;**

**double forgetting\_rate;**

**public static final char[] litery = {'A','B','C','D','E','F','G','H','I','J','M','N','O','P','R','S','T','U','W','X'};**

**public static final int[][] dane\_uczace = {**

**{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},**

**{0,1,1,1,0,1,0,0,0,1,1,0,0,0,0,1,0,1,1,1,1,0,0,0,1,1,0,0,0,1,0,1,1,1,0},**

**{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},**

**{1,1,1,1,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,1,0,0,0,1,0,1,1,1,0},**

**{1,0,0,0,1,1,1,0,1,1,1,0,1,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1},**

**{1,0,0,0,1,1,0,0,0,1,1,1,0,0,1,1,0,1,0,1,1,0,0,1,1,1,0,0,0,1,1,0,0,0,1},**

**{0,1,1,1,0,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,0,1,1,1,0},**

**{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},**

**{0,1,1,1,0,1,0,0,0,1,1,0,0,0,0,0,1,1,1,0,0,0,0,0,1,1,0,0,0,1,0,1,1,1,0},**

**{1,1,1,1,1,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0},**

**{1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,0,1,1,1,0},**

**{1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,0,0,1,1,0,1,0,1,1,0,1,0,1,0,1,0,1,0},**

**{1,0,0,0,1,1,0,0,0,1,0,1,0,1,0,0,0,1,0,0,0,1,0,1,0,1,0,0,0,1,1,0,0,0,1}**

**};**

**public HebbProgram(double learning\_rate, double forgetting\_rate) {**

**weight = new double[35];**

**this.learning\_rate = learning\_rate;**

**this.forgetting\_rate = forgetting\_rate;**

**initializeWeight();**

**}**

**public void Train() {**

**for (int n = 0; n < 20; n++) {**

**double sum;**

**int[] tab;**

**tab = dane\_uczace[n];**

**sum= countSum(tab, weight);**

**for (int i = 0; i < 35; i++) {**

**weight[i] = weight[i] \* forgetting\_rate + learning\_rate \* activationFunction(sum) \* (tab[i] - weight[i]);**

**}**

**}**

**}**

**public void Test(){**

**for (int n = 0; n <20 ; n++) {**

**double result;**

**int[] tab;**

**tab = dane\_uczace[n];**

**double sum = countSum(tab, weight);**

**result = activationFunction(sum);**

**System.out.println("Litera "+ litery[n] +"-> Result: "+ result);**

**}**

**}**

**public double countSum(int[] table, double[] weight){**

**double sum = 0.0;**

**for(int x=0; x < table.length; x++)**

**sum += table[x] \* weight[x];**

**return sum;**

**}**

**private void initializeWeight() {**

**Random random = new Random();**

**for (int i = 0; i < 35; i++) {**

**weight[i] = random.nextDouble();**

**}**

**}**

**public double activationFunction(double s){**

**double result = (1 - Math.exp(-s)) / (1 + Math.exp(-s));**

**return result;**

**}**

**}**