Akademia Górniczo-Hutnicza
WIMIiP, Inżynieria Obliczeniowa
grupa laboratoryjna 01
Agnieszka Kępka

13.12.2018,Kraków

Podstawy Sztucznej Inteligencji

Sprawozdanie numer 4

Uczenie sieci regułą Hebba

1. Cel ćwiczenia:
Celem ćwiczenia było poznanie działania reguły Hebba na przykładzie rozpoznawania emotikon.

2. Zadania do wykonania:

- Wygenerowanie danych uczących i testujących, zawierających 4 różne emotikony w kolorze czarno-białym o wymiarze 8x8 pikseli dla jednej emotikony.
- Przygotowanie sieci oraz reguły Hebba w wersji z i bez współczynnika zapominania.
- Uczenie sieci dla różnych współczynników uczenia i zapominania.
- Testowanie sieci.

3. Zagadnienia teoretyczne:

Reguła Hebba

Zasada działania reguły Hebba polega na interpretacji zachowań aktywnych neuronów. Jeśli aktywny neuron A jest cyklicznie pobudzany przez drugi neuron B, to staje się on jeszcze bardziej czuły na pobudzenie tego neuronu. Reguła Hebba składa się z następujących punktów:

- Jeżeli połączone synapsą neurony A i B są pobudzane jednocześnie (synchronicznie) to połączenie synaptyczne między nimi jest wzmacniane.
- Jeżeli neurony A i B połączone synapsą nie są pobudzane jednocześnie (asynchronicznie) to połączenie pomiędzy nimi jest osłabiane.

Reguła Hebba jest jedną z najpopularniejszych metod samouczenia 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 nie zostają określone 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. 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.
Proces samouczenia ma niestety wady. W porównaniu z procesem uczenia z nauczycielem samouczenie jest zwykle znacznie powolniejsze. Co więcej bez nauczyciela nie można z góry określić, który neuron wyspecjalizuje się w rozpoznawania której klasy sygnałów. Stanowi to pewną trudność przy wykorzystywaniu i interpretacji wyników pracy sieci. Co więcej - nie można określić, czy sieć uczona w ten sposób nauczy się wszystkich prezentowanych jej wzorców. Dlatego sieć przeznaczona do samouczenia musi być większa niż sieć wykonująca to samo zadanie, ale trenowana w sposób klasyczny z udziałem nauczyciela.

Metoda Hebba posiada wiele wad, są nimi m. in.:
- niska efektywność uczenia,
- przemnożony wpływ początkowych wartości wag sieci,
- wagi w przypadku tej reguły mogą rosnąć bez ograniczeń,
- proces uczenia sieci nigdy nie zakończy się samodzielnie,
- nie ma pewności, że jednej klasie wzorców będzie odpowiadał jeden neuron,
- nie ma również gwarancji, że wszystkie klasy wzorców będą miały oddzielne reprezentacje w postaci oddzielnych zbiorów aktywnych neuronów.

Bardzo istotną kwestią jest wybór początkowych wartości wag neuronów sieci przeznaczonej do samouczenia. Wartości te mają bardzo silny wpływ na ostateczne zachowanie sieci, ponieważ proces uczenia jedynie pogłębia i doskonali pewne tendencje istniejące w sieci od samego początku. Od jakości początkowych właściwości sieci silnie zależy do czego sieć dojdzie na końcu procesu uczenia.

4. Opis wykorzystanych metod oraz zmiennych:

Wykorzystanie funkcji **newff(start, wyjscia_s, {'tansig'}, 'trainlm', 'learnh');**, która tworzy sieć neuronową złożoną z jednej warstwy, a jej argumentami są:
– **start** – wartości minimalne i maksymalne dla elementów wejściowych sieci dla funkcji tworzących sieć neuronową, składa się z 4 par, które odpowiednio oznaczają wartość maksymalną - 1 oraz minimalną - 0 na wejściu,

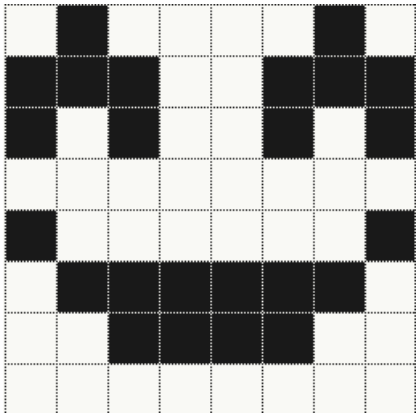
– **wyjscia_s** – liczba elementów wektora wyjściowego sieci
– **tansig** – parametru określającego funkcję aktywacji neuronów (w tym przypadku tangens hiperboliczny),
– **trainlm** – funkcja szkolenia sieciowego, która aktualizuje wartości wagi i odchylenia,
– **learnh** – jest wagą uczenia dla funkcji Hebba.

Inne użyte zmienne oraz funkcje:
– **net** – zmienna, do której będzie przypisywana nowa tworzona sieć neutronowa,
– **WEJSCIE** – dane uczące sieci neuronowej
– **WYJSCIE** – zmienna przechowujące dane wyjściowe odpowiadające danym uczącym (gdzie 1 odpowiada trafienie w dużą literę, a 0 oznacza chybienie),
– **lp.dr** – wartość współczynnika zapominania dla metody Hebba,
– **lp.lr** – wartość współczynnika uczenia dla metody Hebba,
– **wagiHebba** – dostosowanie wartości wag dla metody Hebba za pomocą learnh,
– **net.trainParam.x** – określenie parametrów treningu sieci, gdzie x oznacza określenie:
- **net.trainParam.epochs** – maksymalnej liczby epok,
- **net.trainParam.goal** – celu wydajności sieci,
- **net.trainParam.lr** – wartości współczynnika uczenia się sieci.

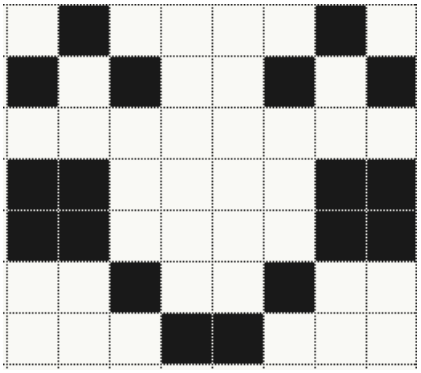
– **train(net, WEJSCIE, wagiHabba)** – uczenie (trening) sieci net z wykorzystaniem danych wejściowych i wartości określonych wcześniej wag dla reguły Hebba,
– ***_testowe** – zmienna dla danych testujących (w miejscu gwiazdki odpowiedni emotikon),
– **efekt** – przypisanie wcześniej zdefiniowanych wag Hebba do zmiennej efekt (wykorzystanie zmiennej do późniejszego wypisania przykładowego działania reguły Hebba),
– **efekt_1 = sim(net, *_testowe)** – symulacja sieci net, parametrami funkcji sim są odpowiednio sieć oraz emotikon, który ma być rozpoznany przez sieć, na końcu wynik testu zostaje wpisany do zmiennej efekt_1,
– **disp('*litera* = '), disp(sum(efekt(*liczba odpowiadająca danej literze*, ':')** – wypisanie przykładowego działania reguły Hebba,
– **disp('*litera* = '), disp(efekt(*liczba odpowiadająca danej literze*)** – wypisanie przykładowego działania sieci.

Umieszczanie emotikonów w tablicy polegało na zinterpretowaniu danego pola przez sieć.
Tablica składa się z 64 pól, które odpowiednio są czarne -1 albo białe -0. Czarne pole oznacza, że w danym miejscu występuje fragment emotikony.

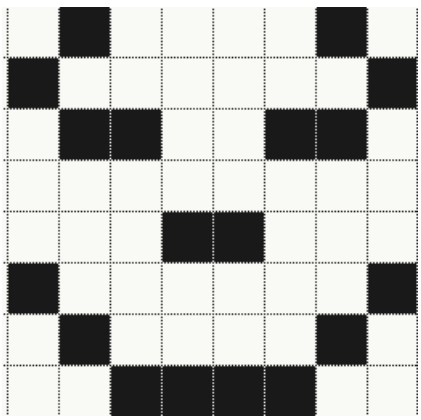
A



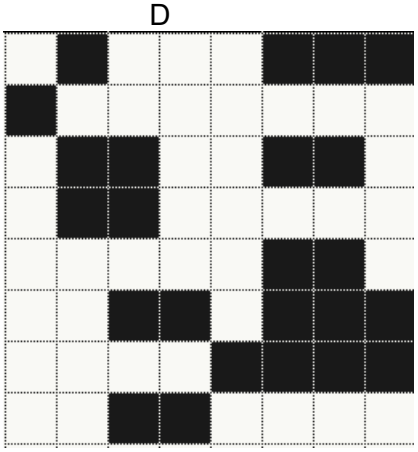
B



C



D



Tak stworzone emotikony zostały zapisane kolumnowo przy zmiennej „WEJŚCIE”, są to dane uczące dla sieci neuronowej.

5.Uczenie sieci dla różnych współczynników uczenia i zapominania:

EMOTIKON A			
Wartość współczynnika uczenia / wartość współczynnika zapominania			
Próba	0.01 / 0.01	0.1 / 0.1	0.5 / 0.5
1	3.5527e-14	-8.8818e-16	1.7764e-15
2	-2.2204e -16	1	-8.8818e-16
3	1.7764e-14	-1	2.6645e-15
4	-1	-4.4409e-16	0
5	1.3323e-14	-1.1102e-15	0

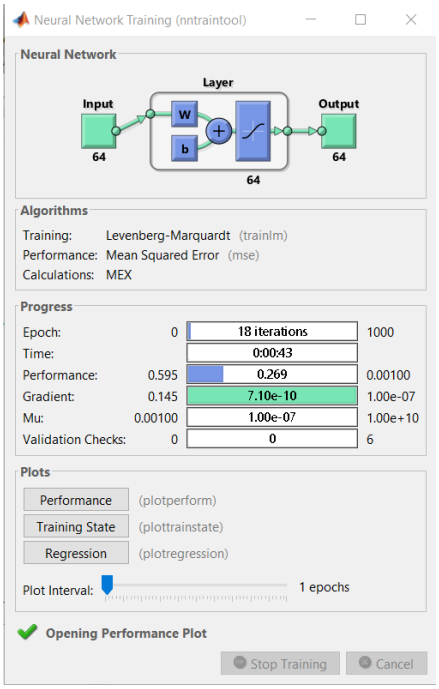
EMOTIKON B			
Wartość współczynnika uczenia / wartość współczynnika zapominania			
Próba	0.01 / 0.01	0.1 / 0.1	0.5 / 0.5
1	0.01	0.1	0.5
2	1	0.1	0.5
3	1	0.1	0.5
4	-1	-1	0.5
5	0.01	0.1	0.5

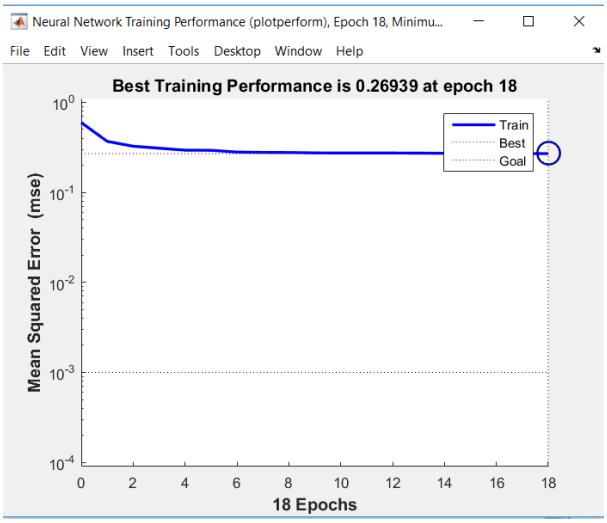
EMOTIKON C			
Wartość współczynnika uczenia / wartość współczynnika zapominania			
Próba	0.01 / 0.01	0.1 / 0.1	0.5 / 0.5
1	-1	1.7764e-15	1
2	-1	1	-4.4409e-16
3	1	2.2204e-16	2.6645e-15
4	2.2204e-16	2.4425e-15	4.4409e-16
5	1	-2.4869e-14	-4.4409e-16

EMOTIKON D			
Wartość współczynnika uczenia / wartość współczynnika zapominania			
Próba	0.01 / 0.01	0.1 / 0.1	0.5 / 0.5
1	4.2188e-15	-8.8818e-16	-1.3323e-15
2	-2.6645e -15	-1	-2.2204e-16
3	-2.2204e-15	-2.4869e-14	4.4409e-16
4	-1.7764e-15	1	-2.2204e-16
5	7.2493e-09	0	1

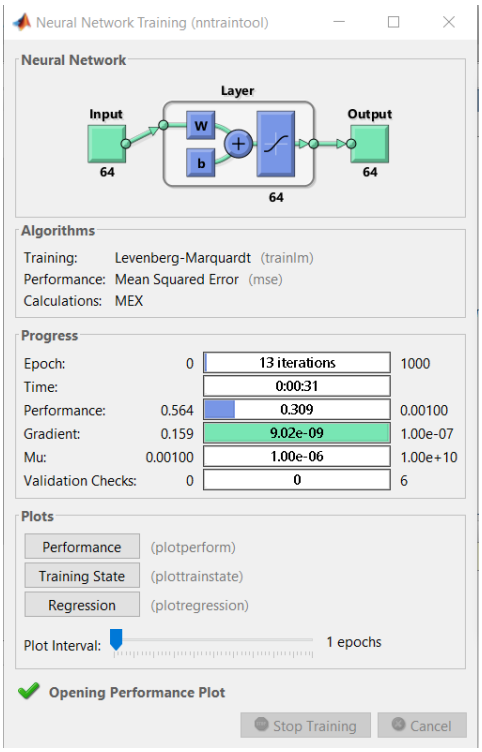
6. Testowanie sieci:

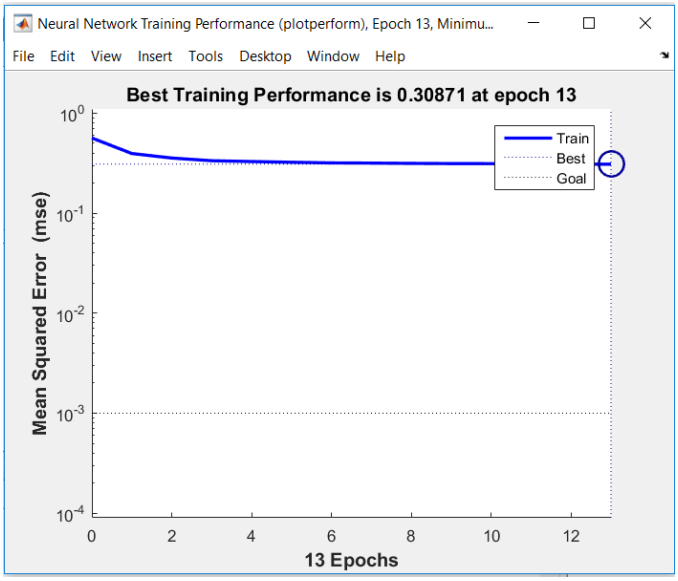
Wartość 0.01 / 0.01



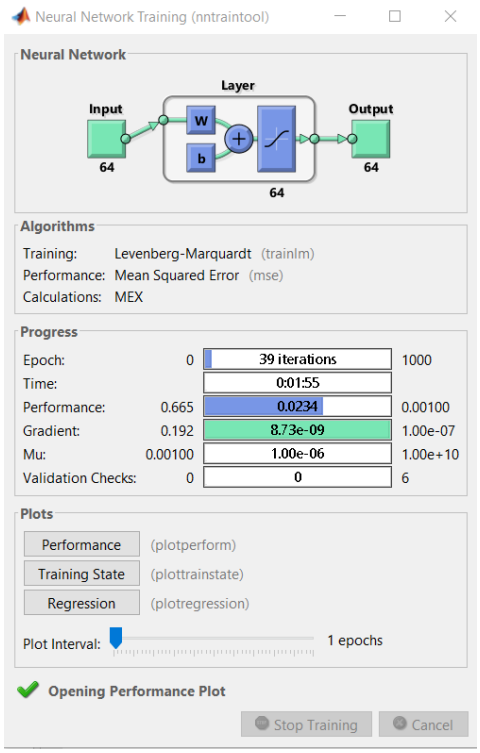


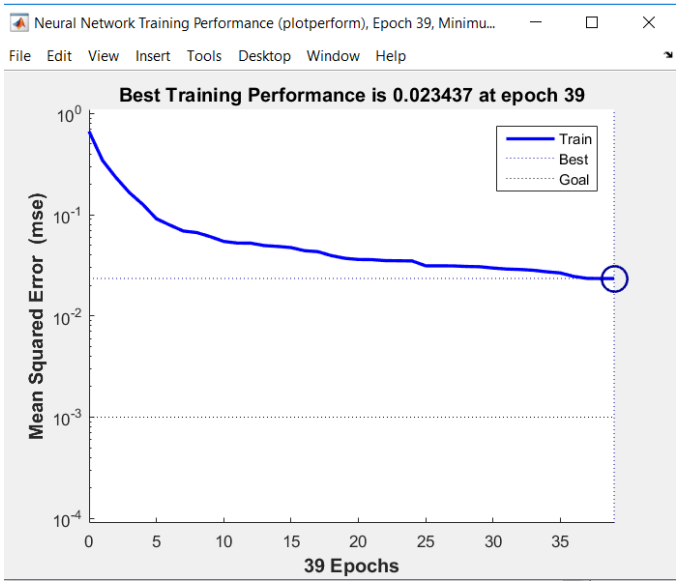
Wartość 0.1 / 0.1





Wartość 0.5 / 0.5





7. Wnioski:

Reguła Hebba jest dobrą alternatywą w porównaniu do poprzednich metod, ponieważ pozawala na uczenie sieci bez nauczyciela (jest jedną z najbardziej zbliżonych metod do prawdziwej, biologicznej sieci neuronowej).

Interpretacja danych przez algorytm nie gwarantuje poprawności. Reguła Hebba opiera się na uczeniu bez nauczyciela, przez co sieć jest zmuszona samodzielnie decydować o skutkach w oparciu o dane wejściowe. To zdecydowanie zwiększa możliwość wystąpienia błędu w działaniu algorytmu.

Ze względu na użyty algorytm można zauważyć, że sieć w porównaniu do wcześniejszych metod użytych na zajęciach zdecydowanie potrzebuje więcej czasu na naukę. Ma na to wpływ przymus nauki bez nauczyciela. Na podstawie tej informacji można wnioskować, że uzyskane wyniki nie świadczą o błędzie - tylko o zbyt wczesnym ich odczytaniu.

Wagi neuronów sieci przeznaczonej do samouczenia mają bardzo silny wpływ na ostateczne zachowanie sieci.

Najlepiej dobranymi parametrami uczenia były wartości zbliżone 0.1. Najwięcej trafień padało gdy trenowaliśmy sieć dla takich wartości.

Wiązało się też to z najdłuższym procesem uczenia, ponieważ zajmowało to 39 epok, ale wyniki były najdokładniejsze.

Przy innych wprowadzonych wartościach sieć uczyła się szybciej, ale osiągała słabsze wyniki.

Kod programu:

```
close all; clear all; clc;

%wejscia do sieci oraz minimalne oraz maksymalne wartosci wejsc
%(64 par 0&1 - osobno dla kazdej z danych uczacych)

start=[0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1;
    0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1;
    0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1;
    0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1;
    0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1;
    0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1; 0 1;
    0 1; 0 1; 0 1; 0 1];

%ilosc wyjsc z sieci (jedna warstwa - 64 neutronow na wyjsciu)
wyjscia_s = 64;

%uzycie funkcji newff
net = newff(start, wyjscia_s, {'tansig'}, 'trainlm', 'learnh');


 %kolumnowa reprezentacja binarna 4 emotikonow dla tablicy 8x4
          %A B C D
WEJSCIE = [0 0 0 0 ;   %tu sa wiersze nr 8
           1 1 1 1 ;
           0 0 0 0 ;
           0 0 0 0 ;
           0 0 0 0 ;
           0 1 0 1 ;
           1 1 1 1 ;
           0 0 0 1 ;

          %A B C D     % tu są wiersze nr 7
           1 1 1 1 ;
           1 0 0 0 ;
           1 1 0 0 ;
           0 0 0 0 ;
           0 0 0 0 ;
           1 1 0 0 ;
           1 0 0 0 ;
           1 1 1 0 ;

          %A B C D     % tu są wiersze nr 6
           1 0 0 0 ;
           0 0 1 1 ;
           1 0 1 1 ;
           0 0 0 0 ;
           0 0 0 0 ;
           1 0 1 1 ;
           0 0 1 1 ;
           1 0 0 0 ;

          %A B C D     % tu są wiersze nr 5
           0 0 0 0 ;
           0 0 0 1 ;
           0 0 0 1 ;
           0 0 0 0 ;
           0 0 0 0 ;
           0 0 0 0 ;
           0 0 0 0 ;
           0 0 0 0 ;

          %A B C D     % tu są wiersze nr 4
           1 1 0 0 ;
           0 1 0 0 ;
           0 0 0 0 ;
           0 0 1 0 ;
           0 0 1 0 ;
           0 0 0 1 ;
```

```
           0 1 0 1 ;
           1 1 0 0 ;

          %A B C D     % tu są wiersze nr 3
           0 1 1 0 ;
           1 1 0 0 ;
           1 0 0 1 ;
           1 0 0 1 ;
           1 0 0 0 ;
           1 0 0 1 ;
           1 1 0 1 ;
           0 1 1 1 ;

          %A B C D     % tu są wiersze nr 2
           0 0 0 0 ;
           0 0 1 0 ;
           1 1 0 0 ;
           1 0 0 0 ;
           1 0 0 1 ;
           1 1 0 1 ;
           0 0 1 1 ;
           0 0 0 1 ;

          %A B C D     % tu są wiersze nr 1
           0 0 0 0 ;
           0 0 0 0 ;
           0 0 1 1 ;
           0 1 1 1 ;
           0 1 1 0 ;
           0 0 1 0 ;
           0 0 0 0 ;
           0 0 0 0 ];

%zmienna, ktora reprezentuje, czy uzytkownik "trafil"
% - 1 oznacza trafienie, 0 - chybienie
WYJSCIE = [1 0 0 0 ;  %A
           0 1 0 0 ;  %B
           0 0 1 0 ;  %C
           0 0 0 1 ]; %D


%PARAMETRY ALGORYTMU HEBBA
% * wspolczynnik zapominania
lp.dr = 0.1;
% * wspolczynnik uczenia
lp.lr = 0.1;

%dostosowanie parametrów sieci do metody Hebba
wagiHebba = learnh([], WEJSCIE, [], [], WYJSCIE, [], [], [], [], [], lp, []);


%PARAMETRY TRENINGU SIECI:
% * maksymalna ilosc epok
net.trainParam.epochs = 1000;
% * cel wydajnosci sieci
net.trainParam.goal = 0.001;
% * wskaznik uczenia sieci
net.trainParam.lr=0.5;
whebb=wagiHebba';
net = train(net, WEJSCIE, whebb);

%dane testowe
a_testowe= [0; 1; 0; 0; 0; 0; 1; 0;  %8
            1; 1; 1; 0; 0; 1; 1; 1;  %7
            1; 0; 1; 0; 0; 1; 0; 1;  %6
            0; 0; 0; 0; 0; 0; 0; 0;  %5
            1; 0; 0; 0; 0; 0; 0; 1;  %4
            0; 1; 1; 1; 1; 1; 1; 0;  %3
            0; 0; 1; 1; 1; 1; 0; 0;  %2
            0; 0; 0; 0; 0; 0; 0; 0]; %1
```

```
b_testowe=[0; 1; 0; 0; 0; 0; 1; 0;  %8
           1; 0; 1; 0; 0; 1; 0; 1;  %7
           0; 0; 0; 0; 0; 0; 0; 0;  %6
           0; 0; 0; 0; 0; 0; 0; 0;  %5
           1; 1; 0; 0; 0; 0; 1; 1;  %4
           1; 1; 0; 0; 0; 0; 1; 1;  %3
           0; 0; 1; 0; 0; 1; 0; 0;  %2
           0; 0; 0; 1; 1; 0; 0; 0]; %1



c_testowe=[0; 1; 0; 0; 0; 0; 1; 0;  %8
           1; 0; 0; 0; 0; 0; 0; 1;  %7
           0; 1; 1; 0; 0; 1; 1; 0;  %6
           0; 0; 0; 0; 0; 0; 0; 0;  %5
           0; 0; 0; 1; 1; 0; 0; 0;  %4
           1; 0; 0; 0; 0; 0; 0; 1;  %3
           0; 1; 0; 0; 0; 0; 1; 0;  %2
           0; 0; 1; 1; 1; 1; 0; 0]; %1


d_testowe=[0; 1; 0; 0; 0; 1; 1; 1;  %8
           1; 0; 0; 0; 0; 0; 0; 0;  %7
           0; 1; 1; 0; 0; 1; 1; 0;  %6
           0; 1; 1; 0; 0; 0; 0; 0;  %5
           0; 0; 0; 0; 0; 1; 1; 0;  %4
           0; 0; 1; 1; 0; 1; 1; 1;  %3
           0; 0; 0; 0; 1; 1; 1; 1;  %2
           0; 0; 1; 1; 0; 0; 0; 0]; %1

efekt = wagiHebba;
%symulacja sieci net
efekt_1 = sim(net, a_testowe);

%wypisywanie wartosci reguly Hebba, wypisywanie kolejnych wierszy
disp('Jednokrotne wykorzystanie reguly Hebba: ')
disp('A = '), disp(sum(efekt(1, ':')));
disp('B = '), disp(sum(efekt(2, ':')));
disp('C = '), disp(sum(efekt(3, ':')));
disp('D = '), disp(sum(efekt(4, ':')));


%wypisywanie wartosci
disp('Dzialanie algorytmu z wykorzystaniem r. Hebba dla wszystkich emotikon: ')
disp('A = '), disp(efekt_1(1));
disp('B = '), disp(efekt_1(2));
disp('C = '), disp(efekt_1(3));
disp('D = '), disp(efekt_1(4));
```