

## Podstawy Sztucznej Inteligencji

### Sprawozdanie z Projektu 4

### Uczenie sieci regułą Hebba

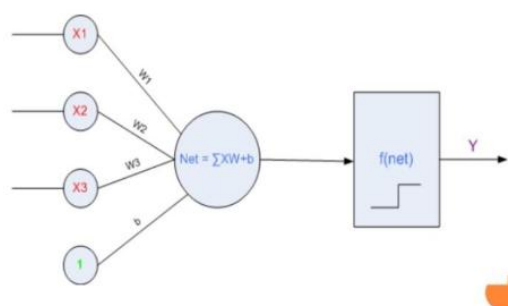
#### 1. Cel ćwiczenia:

Celem ćwiczenia jest poznanie działania reguły Hebba na przykładzie rozpoznawania emotikon.

#### 2. Reguła Hebba

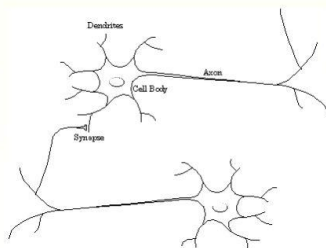
Jest to jedna z najpopularniejszych metod samo uczenia sieci neuronowych. Polega ona na tym, że sieci pokazuje się kolejne przykłady sygnałów wejściowych, nie podając żadnych informacji o tym, co z tymi sygnałami należy zrobić. Sieć obserwuje otoczenie i odbiera różne sygnały, nikt nie określa jednak, 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

NEURON HEBB



### Reguła Hebba

- "Kiedy akson komórki A jest dostatecznie blisko by pobudzić komórkę B i wielokrotnie w sposób trwały bierze udział w jej pobudzaniu, procesy wzrostu lub zmian metabolicznych zachodzą w obu komórkach tak, że sprawność neuronu A jako jednej z komórek pobudzających B, wzrasta."

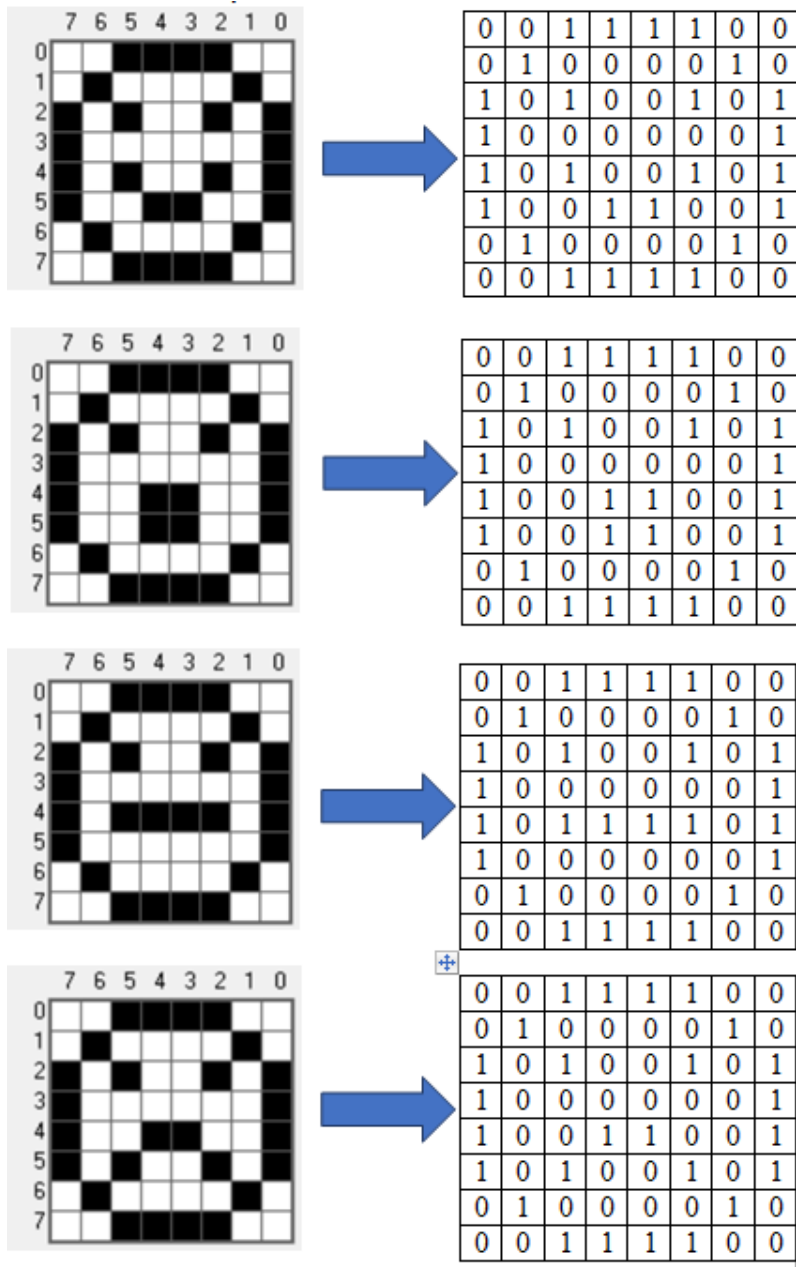


D. O. Hebb, 1949

### 3. Opis wykonanego zadania:

- a) Wygenerowanie danych uczących i testujących, zawierających 4 różne emotikony np. Czarno-białe, wymiar 8x8 pikseli dla jednej emotikony.

Poniżej 4 emotikony, które powstały na macierzy wielkości 8x8. Białe pole - wartość 0, natomiast czarne - wartość 1.



Stworzyłam także tablicę 64 wartości (0,1), które odpowiadają wielkości macierzy, ponieważ na każdym polu może znaleźć się tylko wartość 0-puste lub 1-pełne.

Zmienna *output* zawiera wyjście gdzie 1 oznacza, że dany emotikon powstał, a 0 jest przeciwnieństwem.

- b) Przygotowanie (implementacja lub wykorzystanie gotowych narzędzi) sieci oraz reguły Hebba w wersji z i bez współczynnika zapominania.

Skorzystałam z gotowych narzędzi pakietu Matlab, które zawierają już przygotowane funkcje tworzące sieć i wykorzystujące regułę Hebba.

Funkcja `newff(PR,[S1 S2...snl],{TF1 TF2...tfnl},BTF,BLF,PF)`

PR – macierz wejściowa z

[S1 S2...snl]–liczba neuronów w kolejnych warstwach

{TF1 TF2...tfnl}–funkcje aktywacji neuronów w kolejnych warstwach ( u mnie tansig czyli tangens hiperboliczny

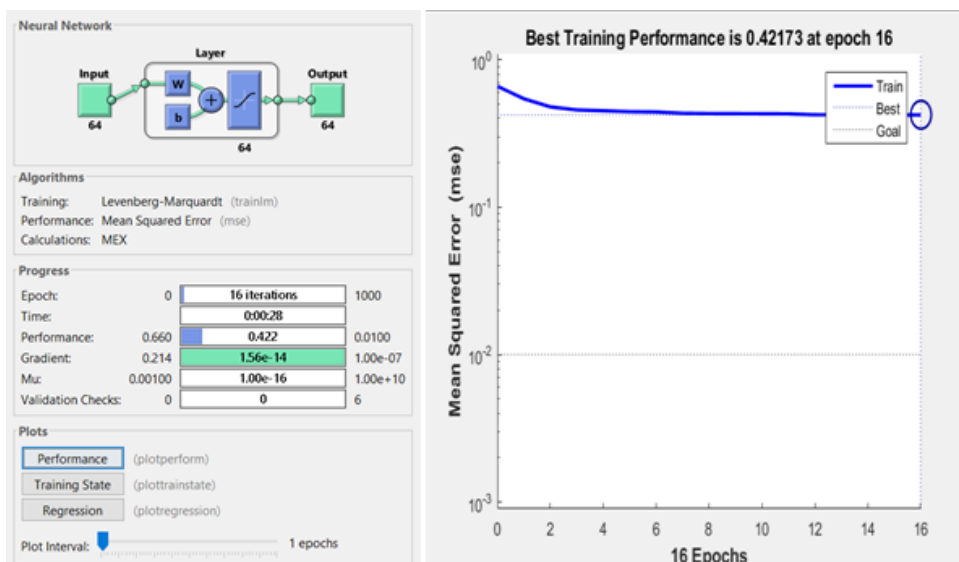
BTF,BLF,PF – funkcje wykorzystywane do treningu sieci

Funkcja `learnh(W,P,Z,N,A,T,E,gw,ga,D,LP,LS)`

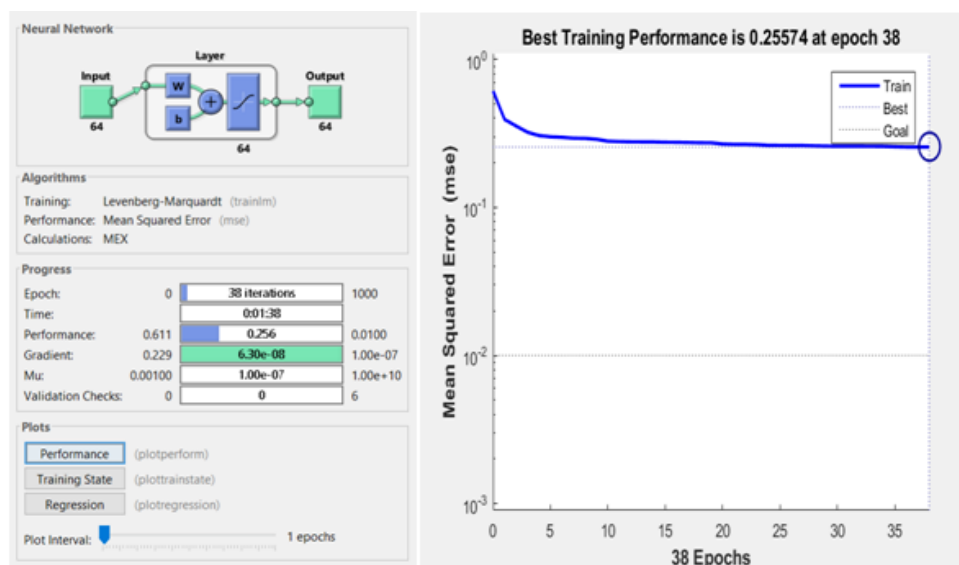
Gdzie w odpowiednie miejsca wstawiłam macierz wejściową, wyjściową i parametry służące regule Hebba.

#### 4. Testowanie sieci:

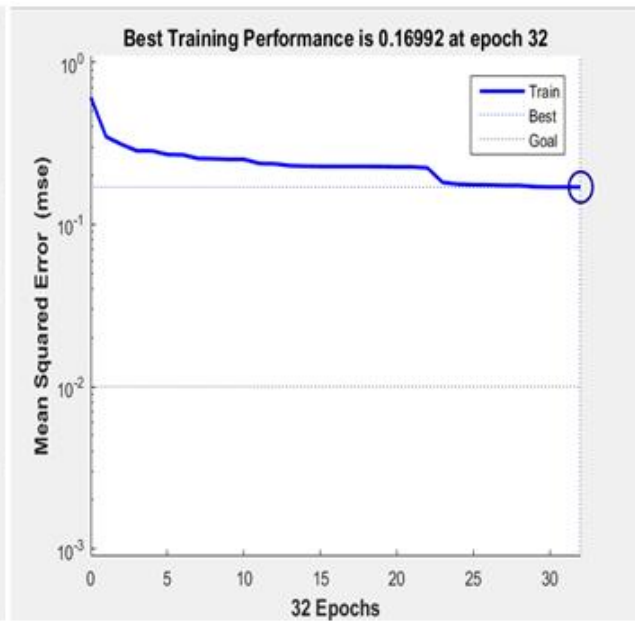
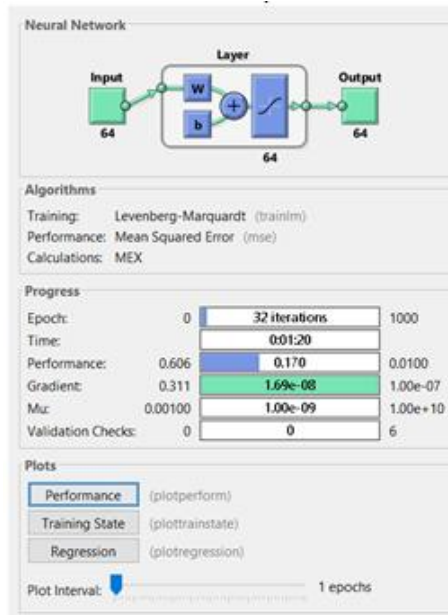
 Dla 0.01



 Dla 0.1



## Dla 0.5



### SMILE:

Próba	0.01/0.01
1	-0.7096
2	1
3	0.4374
4	-1
5	-0.8830

### SHOCK:

Próba	0.01/0.01
1	-0.1986
2	1
3	0.1491
4	-1
5	-0.2091

### CONFUSE:

Próba	0.01/0.01
1	-0.5272
2	1
3	0.6064
4	-1
5	-0.6268

## 5. Wnioski:

- ✚ Proces samouczenia ma wady. W porównaniu z procesem uczenia z nauczycielem samo uczenie jest zwykle znacznie powolniejsze.
- ✚ 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.
- ✚ Bardzo istotną kwestią jest wybór początkowych wartości wag neuronów sieci przeznaczonej do samouczenia. Wartości te 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 38 epok, ale wyniki były najdokładniejsze.
- ✚ Przy innych wprowadzonych wartościach sieć uczyła się szybciej, ale osiągała słabsze wyniki.
- ✚ Ujemne wartości są prawdopodobnie spowodowane faktem, iż neurony nie traktują wprowadzonych do nich wartości jako własne i wyliczają je na minusie.

## 6. Listing kodu wraz z komentarzami:

```
closeall; clearall; clc;

%wejscia do sieci z min i max wartosciami
minmax=[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
ilosc_wyj = 64;

%uzycie funkcji tworzacej siec
net = newff(minmax, ilosc_wyj,{'tansig'}, 'trainlm',
'learnh');

%kolumnowe wprowadzenie emotikon w formie 0-1
%      smile/shock/confuse/sad
input  = [ 0 0 0 0;
           0 0 0 0;
           1 1 1 1;
           1 1 1 1;
```

1 1 1 1;  
1 1 1 1;  
0 0 0 0;  
0 0 0 0;  
0 0 0 0;  
1 1 1 1;  
0 0 0 0;  
0 0 0 0;  
0 0 0 0;  
0 0 0 0;  
1 1 1 1;  
0 0 0 0;  
1 1 1 1;  
0 0 0 0;  
1 1 1 1;  
0 0 0 0;  
0 0 0 0;  
1 1 1 1;  
0 0 0 0;  
1 1 1 1;  
1 1 1 1;  
0 0 0 0;  
0 0 0 0;  
0 0 0 0;  
0 0 0 0;  
0 0 0 0;  
0 0 0 0;  
1 1 1 1;  
1 1 1 1;  
0 0 0 0;  
1 0 1 0;  
0 1 1 1;  
0 1 1 1;  
1 0 1 0;  
0 0 0 0;  
1 1 1 1;  
1 1 1 1;  
0 0 0 0;  
0 0 0 1;  
1 1 0 0;  
1 1 0 0;  
0 0 0 1;  
0 0 0 0;  
1 1 1 1;  
0 0 0 0;  
1 1 1 1;  
0 0 0 0;  
0 0 0 0;  
0 0 0 0;  
0 0 0 0;  
1 1 1 1;

```

        0 0 0 0;
        0 0 0 0;
        0 0 0 0;
        1 1 1 1;
        1 1 1 1;
        1 1 1 1;
        1 1 1 1;
        1 1 1 1;
        0 0 0 0;
        0 0 0 0;
    ];

    %zmienna zawierająca 1 gdy trafimy w emotikon i 0 gdy chybimy
    output = [ 1 0 0 0    %smile
               0 1 0 0    %shock
               0 0 1 0    %confuse
               0 0 0 1]; %sad

    %parametry reguły Hebba
    lp.dr = 0.5; %wsp. zapominania
    lp.lr = 0.9; %wsp. uczenia

    %użycie reguły Hebba
    hebb = learnh( [], input, [], [], output, [], [], [], [], [],
    lp, []);
    heb=hebb';

    net.trainParam.epochs = 1000;
    net.trainParam.goal = 0.01;

    %trenowanie sieci z użyciem reguły Hebba
    net = train(net, input, heb);

    %DANE TESTUJACE
    smile = [ 0 0 1 1 1 1 0 0;
              0 1 0 0 0 0 1 0;
              1 0 1 0 0 1 0 1;
              1 0 0 0 0 0 0 1;
              1 0 1 0 0 1 0 1;
              1 0 0 1 1 0 0 1;
              0 1 0 0 0 0 1 0;
              0 0 1 1 1 1 0 0];
    shock = [ 0 0 1 1 1 1 0 0;
              0 1 0 0 0 0 1 0;
              1 0 1 0 0 1 0 1;
              1 0 0 0 0 0 0 1;
              1 0 0 1 1 0 0 1;
              1 0 0 1 1 0 0 1;
              0 1 0 0 0 0 1 0;
              0 0 1 1 1 1 0 0];

```

```
confuse = [ 0 0 1 1 1 1 0 0;  
0 1 0 0 0 0 1 0;  
1 0 1 0 0 1 0 1;  
1 0 0 0 0 0 0 1;  
1 0 1 1 1 1 0 1;  
1 0 0 1 1 0 0 1;  
0 1 0 0 0 0 1 0;  
0 0 1 1 1 1 0 0];  
sad = [ 0 0 1 1 1 1 0 0;  
0 1 0 0 0 0 1 0;  
1 0 1 0 0 1 0 1;  
1 0 0 0 0 0 0 1;  
1 0 0 1 1 0 0 1;  
1 0 1 0 0 1 0 1;  
0 1 0 0 0 0 1 0;  
0 0 1 1 1 1 0 0];
```

```
%sprawdzenie poprawności wytrenowanej sieci
```

```
test = sim(net, smile);  
test1 = sim(net, shock);  
test2 = sim(net, confuse);  
test3 = sim(net, sad);
```

```
%wypisanie wartości
```

```
disp('SMILE ='), disp(test(1));  
disp('SHOCK ='), disp(test1(1));  
disp('CONFUSE ='), disp(test2(1));  
disp('SAD ='), disp(test3(1));
```