Aleksandra Spiecha

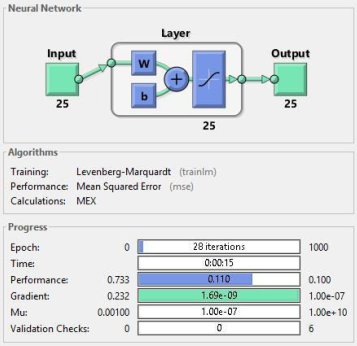
Podstawy sztucznej inteligencji

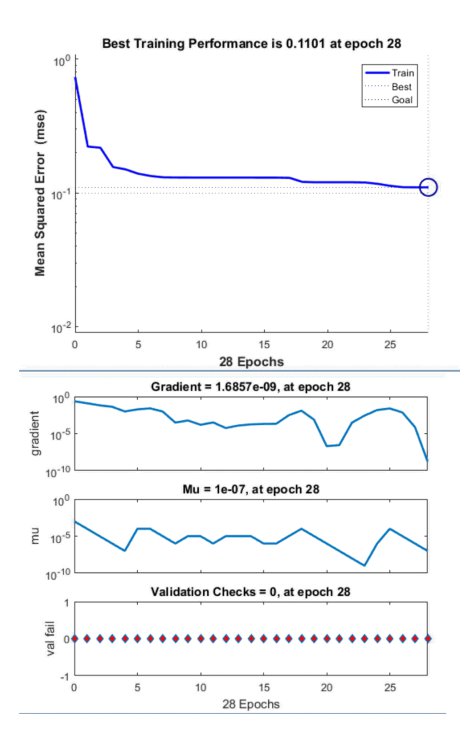
**Sprawozdanie nr 4**

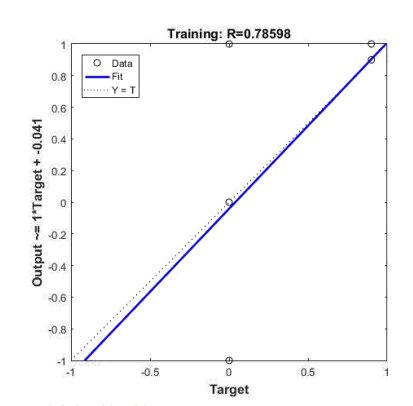
Cel ćwiczenia:Poznanie działania reguły Hebba na przykładzie rozpoznawania emotikon (użyte emotikony : :O ☹

**Wstęp teoretyczny:**- *Sieć neuronowa (sztuczna sieć neuronowa)* – ogólna nazwa struktur matematycznych i ich  
programowych lub sprzętowych modeli, realizujących obliczenia lub przetwarzanie  
sygnałów poprzez rzędy elementów, zwanych sztucznymi neuronami, wykonujących pewną  
podstawową operację na swoim wejściu. Oryginalną inspiracją takiej struktury była budowa  
naturalnych neuronów, łączących je synaps, oraz układów nerwowych, w szczególności  
mózgu.  
- *Reguła Hebba* - Jest to jedna 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, 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.  
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.  
Dokładniejsza analiza procesu samouczenia metodą Hebba pozwala stwierdzić, że w wyniku  
konsekwentnego stosowania opisanego algorytmu początkowe, najczęściej przypadkowe  
„preferencje” neuronów ulegają systematycznemu wzmacnianiu i dokładnej polaryzacji. Jeśli  
jakiś neuron miał „wrodzoną skłonność” do akceptowania sygnałów pewnego rodzaju - to w  
miarę kolejnych pokazów nauczy się te sygnały rozpoznawać coraz dokładniej i coraz bardziej  
precyzyjnie. Po dłuższym czasie takiego samouczenia w sieci powstaną zatem wzorce  
poszczególnych typów występujących na wejściu sieci sygnałów. W wyniku tego procesu sygnały  
podobne do siebie będą w miarę postępu uczenia coraz skuteczniej grupowane i rozpoznawane  
przez pewne neurony, zaś inne typy sygnałów staną się „obiektem zainteresowania” innych  
neuronów W wyniku tego procesu samouczenia sieć nauczy się, ile klas podobnych do siebie  
sygnałów pojawia się na jej wejściach oraz sama przyporządkuje tym klasom sygnałów neurony,  
które nauczą się je rozróżniać, rozpoznawać i sygnalizować.

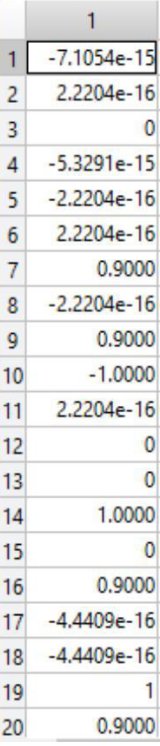
• **Przykładowe wyniki po uruchomieniu programu:**- parametry algorytmu Hebba : współczynnik zapomnienia = 0.0, współczynnik uczenia = 0.9;  
parametry treningu sieci : maksymalna ilość epok = 1000, cel wydajności sieci = 0.1,  
wskaźnik uczenia sieci = 0.2











**Listing kodu w Matlab:**

close all; clear all; clc;

%wejscia do sieci oraz minimalne oraz maksymalne wartosci wejsc

%(25 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;];

%ilosc wyjsc z sieci (jedna warstwa - 25 neutronow na wyjsciu)

wyjscia\_s = 25;

%uzycie funkcji newff

net = newff(start, wyjscia\_s, {'tansig'}, 'trainlm', 'learnh');

%kolumnowa reprezentacja binarna 4 emotikonow dla tablicy 8x4

% :):O:(:|

WEJSCIE = [ 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

0 0 0 0

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 0 0 0

0 1 1 1

0 1 1 1

0 1 1 1

1 0 0 0

0 0 1 0

1 1 0 0

1 1 0 0

1 1 0 0

0 0 1 0

];

%zmienna, ktora reprezentuje, czy uzytkownik "trafil" w wybrana przez

%siebie emotikone - 1 oznacza trafienie, 0 - chybienie

WYJSCIE = [1 0 0 0 ; %:)

0 1 0 0 ; %:O

0 0 1 0 ; %:(

0 0 0 1 ]; %:|

%PARAMETRY ALGORYTMU HEBBA

% \* wspolczynnik zapominania

lp.dr = 0.0;

% \* wspolczynnik uczenia

lp.lr = 0.99;

%dostosowanie parametr?w 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;0;0;0;0;

0;1;0;1;0;

0;0;0;0;0;

1;0;0;0;1;

0;1;1;1;0]; %:)

b\_testowe=[0;0;0;0;0;

0;1;0;1;0;

0;0;0;0;0;

0;1;1;1;0;

0;1;1;1;0;]; %:O

c\_testowe=[0;0;0;0;0;

0;1;0;1;0;

0;0;0;0;0;

0;1;1;1;0;

1;0;0;0;1;]; %:(

d\_testowe=[0;0;0;0;0;

0;1;0;1;0;

0;0;0;0;0;

0;1;1;1;0;

0;0;0;0;0;]; %:|

efekt = wagiHebba;

%symulacja sieci net

efekt\_1 = sim(net, a\_testowe);

%wypisywanie wartosci reguly Hebba, wypisywanie kolejnych wierszy

disp('Jednokrotne wykorzystanie reguly Hebba: ')

disp(':) = '), disp(sum(efekt(1, ':')));

disp(':O = '), disp(sum(efekt(2, ':')));

disp(':( = '), disp(sum(efekt(3, ':')));

disp(':| = '), disp(sum(efekt(4, ':')));

%wypisywanie wartosci

disp('Dzialanie algorytmu z wykorzystaniem r. Hebba dla emotikon: ')

disp(':) = '), disp(efekt\_1(1));

disp(':O = '), disp(efekt\_1(2));

disp(':( = '), disp(efekt\_1(3));

disp(':| = '), disp(efekt\_1(4));

**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.