

Maciej Krzywda,
Inżynieria Obliczeniowa, IMiP
Podstawy Sztucznej Inteligencji
nr albumu: 293102

Sprawozdanie 4
Tytuł projektu:
Uczenie sieci regułą Hebba.

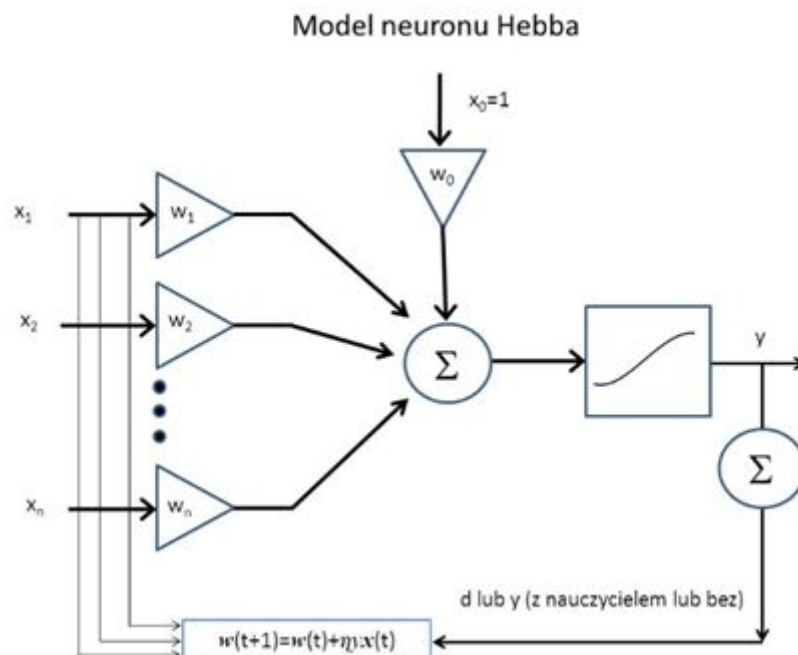
- **Cel projektu:**

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

- **Przebieg ćwiczenia**

1. Wygenerowanie danych uczących i testujących, zawierających 4 różne emotikony np. czarno-białe, wymiar 8x8 pikseli dla jednej emotikony.
2. Przygotowanie (implementacja lub wykorzystanie gotowych narzędzi) sieci oraz reguły Hebba w wersji z i bez współczynnika zapominania.
3. Uczenie sieci dla różnych współczynników uczenia i zapominania.
4. Testowanie sieci.

- **Część Teoretyczna**



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.

Proces samouczenia ma niestety wady. W porównaniu z procesem uczenia z nauczycielem samouczenie jest zwykle znacznie wolniejsze. Co więcej bez nauczyciela nie można z góry określić, który neuron wyspecjalizuje się w rozpoznawaniu 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. - Szacunkowo sieć powinna mieć co najmniej trzykrotnie więcej

elementów warstwy wyjściowej niż wynosi oczekiwana liczba różnych wzorów, które sieć ma rozpoznawać.

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, przeto od jakości tych początkowych, „wrodzonych” właściwości sieci silnie zależy, do czego sieć dojdzie na końcu procesu uczenia. Nie wiedząc z góry, jakiego zadania sieć powinna się uczyć, trudno wprowadzać jakikolwiek zdeterminowany mechanizm nadawania początkowych wartości wag, jednak pozostawienie wszystkiego wyłącznie mechanizmom losowym może powodować, że sieć (zwłaszcza mała) może nie zdołać wystarczająco zróżnicować swego działania w początkowym okresie procesu uczenia i wszelkie późniejsze wysiłki, by znaleźć w strukturze sieci reprezentację dla wszystkich występujących w wejściowych sygnałach klas, mogą okazać się daremne. Można jednak wprowadzić pewien mechanizm wstępnego „rozprowadzania” wartości wag w początkowej fazie procesu uczenia. Metoda ta, zwana *convex combination* modyfikuje początkowe wartości wag w taki sposób, by zwiększyć prawdopodobieństwo równomiernego pokrycia przez poszczególne neurony wszystkich typowych sytuacji pojawiających się w wejściowym zbiorze danych. Jeśli tylko dane pojawiające się w początkowej fazie uczenia nie będą różniły się istotnie od tych, jakie sieć będzie potem analizować i różnicować – metoda *convex combination* stworzy w sposób automatyczny dogodny punkt wyjścia do dalszego samouczenia i zapewni stosunkowo dobrą jakość nauczonej sieci w większości praktycznych zadań

$$w_{ij}[n+1] = w_{ij}[n] + \eta x_i[n] x_j[n]$$

- Schemat działania sieci w programie Matlab

• Przebieg ćwiczenia

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

4 emotikony, które powstały na macierzy wielkości 8x8. Dla białego pola 0, dla czarnego 1.

Następnie powyższe macierze zapisane zostały w formie kolumnowej jako zmienną *input*. Dane testujące powstały z powyższych macierzy i zostały zapisane w formie wierszowej, gdzie każde przejście do nowego wiersza jest zaznaczone znakiem ; .

Powstała 64 elementowe wejście, które składa się z (0,1), 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 przeciwieństwem. Wykorzystałem z gotowych narzędzi pakietu Matlab, które zawierają już przygotowane funkcje tworzące sieć i wykorzystujące regułę Hebba.

newff

Create a feed-forward backpropagation network

Syntax

```
net = newff
```

```
net = newff(PR,[S1 S2...SN1},{TF1 TF2...TFN1},BTF,BLF,PF)
```

Description

`net = newff` creates a new network with a dialog box.

`newff(PR,[S1 S2...SN1},{TF1 TF2...TFN1},BTF,BLF,PF)` takes,

`PR` - $R \times 2$ matrix of min and max values for R input elements.

`Si` - Size of i th layer, for $N1$ layers.

`TFi` - Transfer function of i th layer, default = 'tansig'.

`BTF` - Backpropagation network training function, default = 'traingdx'.

`BLF` - Backpropagation weight/bias learning function, default = 'learngdm'.

`PF` - Performance function, default = 'mse'.

and returns an N layer feed-forward backprop network.

Syntax

```
[dW,LS] = learnh(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)
info = learnh(code)
```

Description

`learnh` is the Hebb weight learning function.

`learnh(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)` takes several inputs,

`W` - $S \times R$ weight matrix (or $S \times 1$ bias vector).

`P` - $R \times Q$ input vectors (or `ones(1,Q)`).

`Z` - $S \times Q$ weighted input vectors.

`N` - $S \times Q$ net input vectors.

`A` - $S \times Q$ output vectors.

`T` - $S \times Q$ layer target vectors.

`E` - $S \times Q$ layer error vectors.

`gW` - $S \times R$ gradient with respect to performance.

`gA` - $S \times Q$ output gradient with respect to performance.

`D` - $S \times S$ neuron distances.

`LP` - Learning parameters, none, `LP = []`.

`LS` - Learning state, initially should be = `[]`.

and returns,

`dW` - $S \times R$ weight (or bias) change matrix.

`LS` - New learning state.

Learning occurs according to `learnh`'s learning parameter, shown here with its default value.

`LP.lr = 0.01` - Learning rate.

`learnh(code)` returns useful information for each code string:

'pnames' - Names of learning parameters.

'pdefaults' - Default learning parameters.

'needg' - Returns 1 if this function uses `gW` or `gA`.

Reprezentacja Graficzna i macierzowa Emotikon.

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

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

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	0	0	0	0	1
0	1	0	0	0	0	1	0
0	0	1	1	1	1	0	0

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

- Listing programu

[illegible]

```

        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;
        0 0 0 0;
        0 0 0 0;
    ];

%1 success / 0 fail in emotikon.
output = [ 1 0 0 0    %smile
          0 1 0 0    %shock
          0 0 1 0    %confuse
          0 0 0 1]; %sad

%Hebb Learn Parameters
lp.dr = 0.01; %wsp. zapominania
lp.lr = 0.01; %wsp. uczenia

%Use Hebb Methods
hebb = learnh( [], input, [], [], output, [], [], [], [], [], lp, []);
heb=hebb';

net.trainParam.epochs = 1000;
net.trainParam.goal = 0.01;
net = train(net, input, heb);

```



```

%Training Data.
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];

%TEST.
test = sim(net, smile);
test1 = sim(net, shock);
test2 = sim(net, confuse);
test3 = sim(net, sad);

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

```

• Analiza Wyników

EMOTIKON SMILE			
Wartość współczynnika uczenia / wartość współczynnika zapominania			
Próba	0.01 / 0.01	0.1 / 0.1	0.5 / 0.5
1	-0.7096	1	-1
2	1	1	0.5801
3	0.4374	0.5436	-0.8171
4	-1	-0.8520	0.8112
5	-0.8830	1	0.6000

EMOTIKON SHOCK

Wartość współczynnika uczenia / wartość współczynnika zapominania			
Próba	0.01 / 0.01	0.1 / 0.1	0.5 / 0.5
1	-0.1986	1	-1
2	1	-0.4893	-0.4557
3	0.1491	1	0.2064
4	-1	0.9852	0.9192
5	-0.2091	1	-0.1354

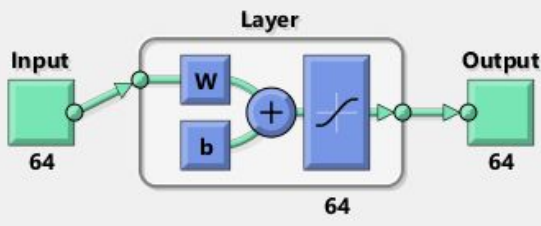
EMOTIKON CONFUSE			
Wartość współczynnika uczenia / wartość współczynnika zapominania			
Próba	0.01 / 0.01	0.1 / 0.1	0.5 / 0.5
1	-0.5272	1	-0.9834
2	1	1	0.4274
3	0.6064	1	-0.3183
4	-1	0.7723	0.9770
5	-0.6268	1	-0.0335

EMOTIKON SAD			
Wartość współczynnika uczenia / wartość współczynnika zapominania			
Próba	0.01 / 0.01	0.1 / 0.1	0.5 / 0.5
1	0.5638	1	-1
2	1	0.3350	-0.5065
3	0.5542	1	0.6986
4	-1	-0.9076	0.6985
5	0.5705	1	-0.5098

lp.dr = 0.01; %wsp. zapominania

```
lp.lr = 0.01; %wsp. uczenia
```

Neural Network



Algorithms

Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Calculations: MEX

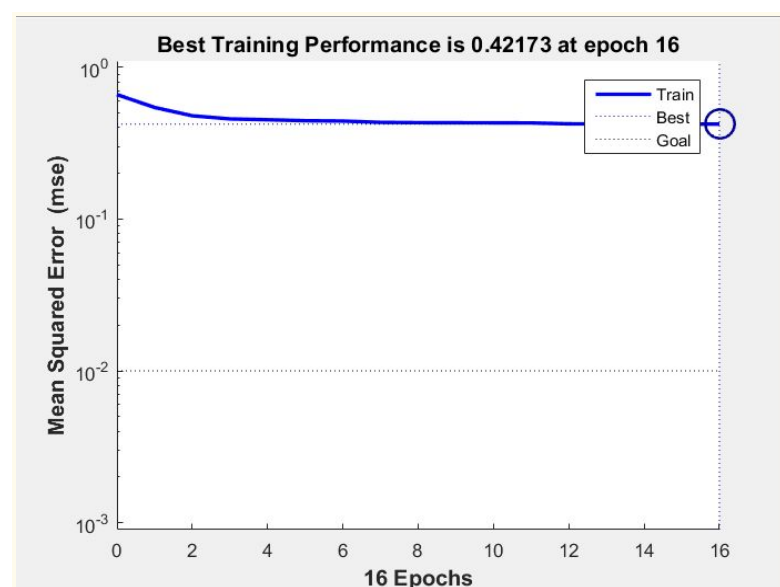
Progress

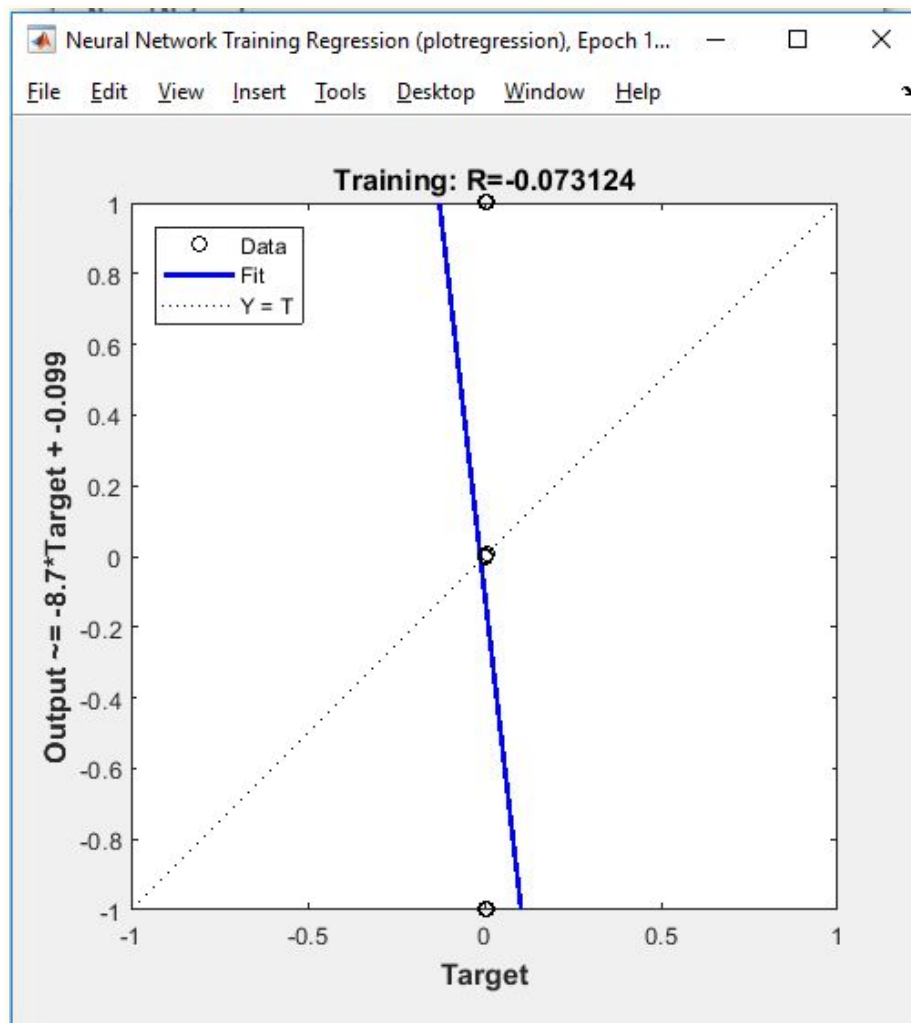
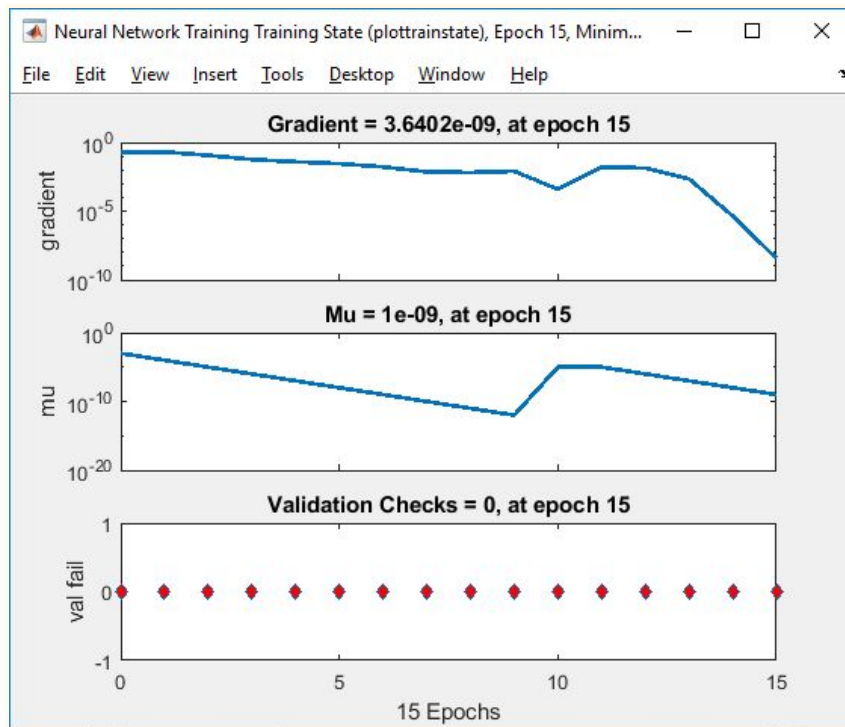
Epoch:	0	16 iterations	1000
Time:		0:00:28	
Performance:	0.660	0.422	0.0100
Gradient:	0.214	1.56e-14	1.00e-07
Mu:	0.00100	1.00e-16	1.00e+10
Validation Checks:	0	0	6

Plots

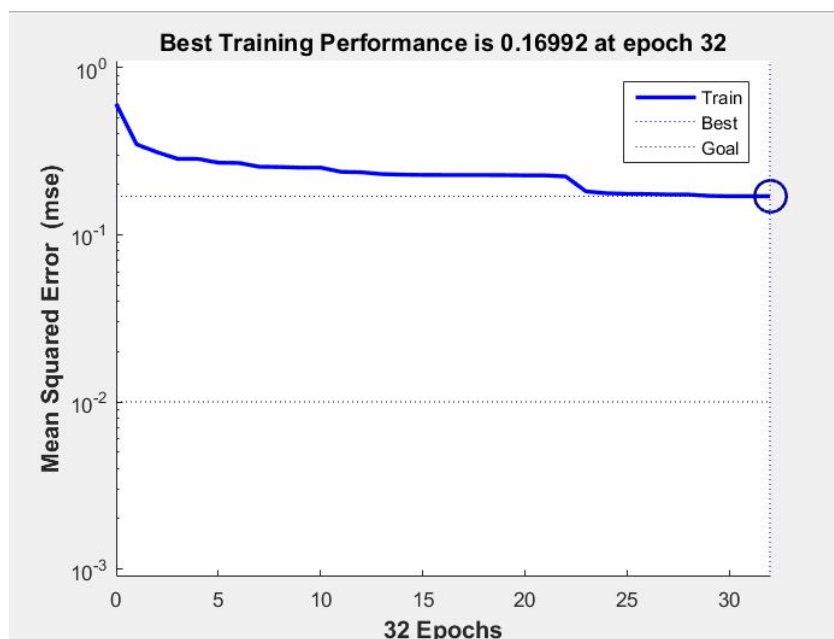
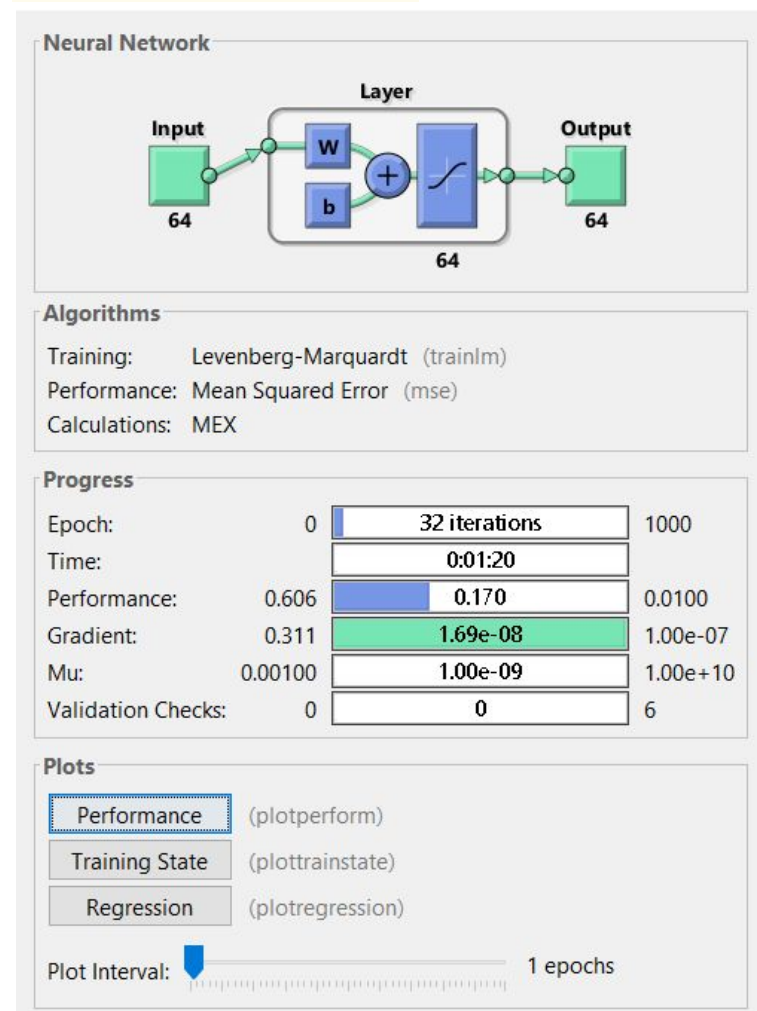
☒ Performance (plotperform)
☐ Training State (plottrainstate)
☐ Regression (plotregression)

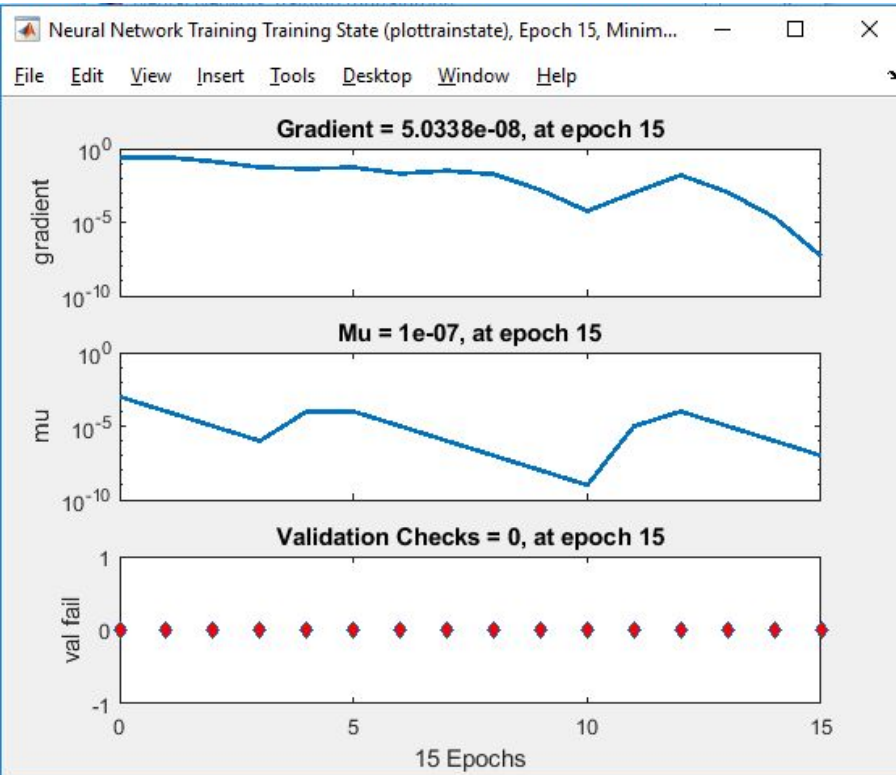
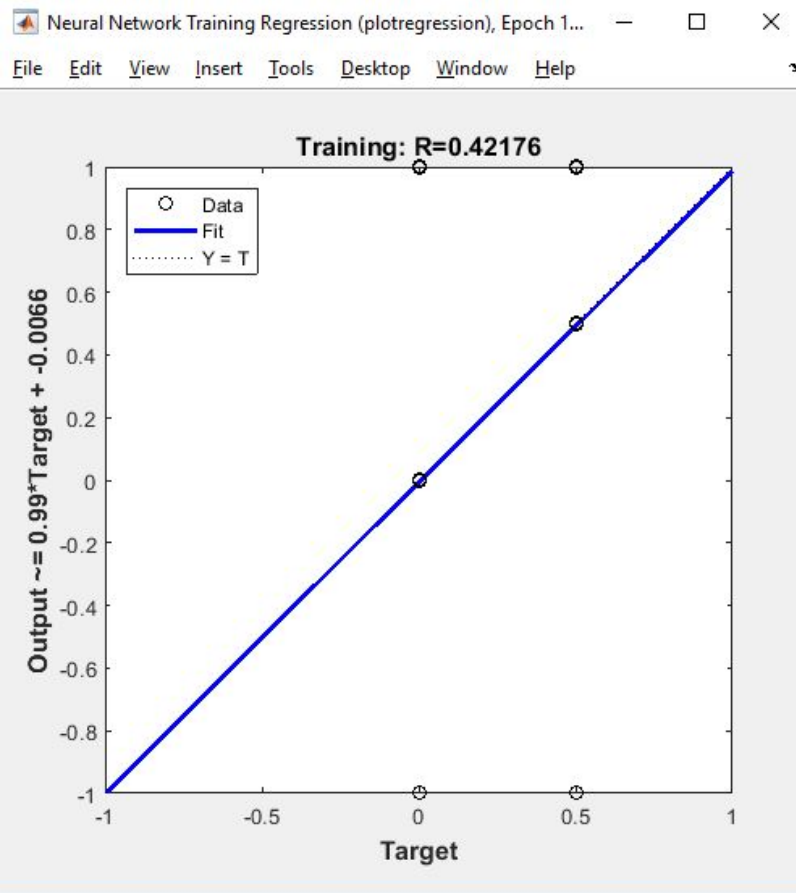
Plot Interval: 1 epochs



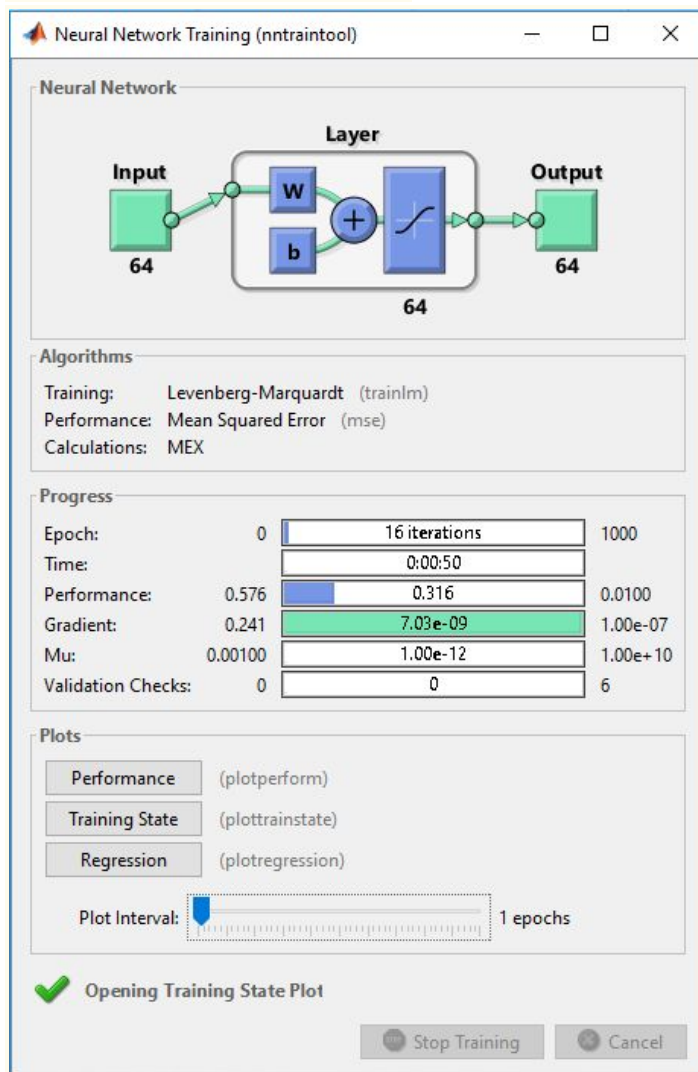


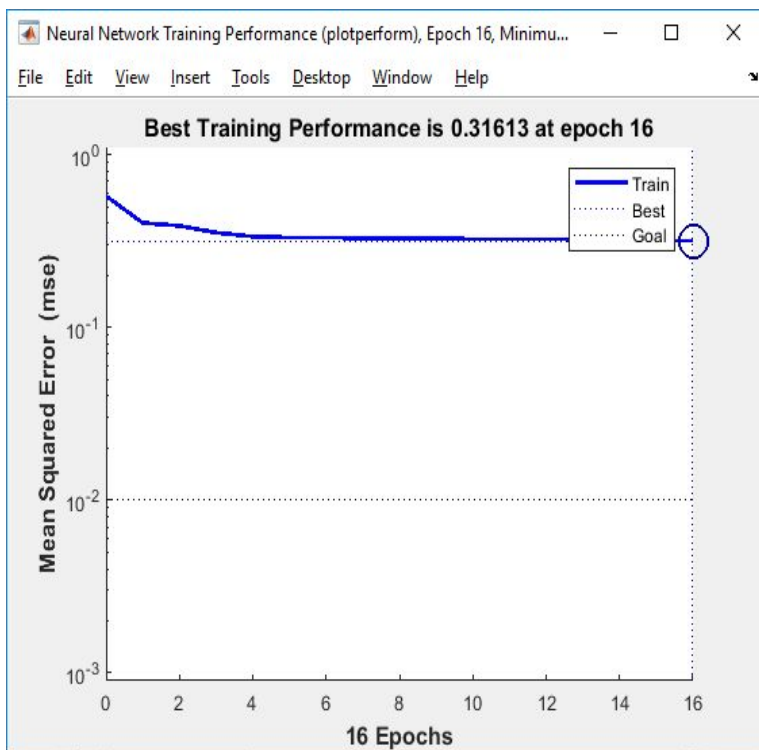
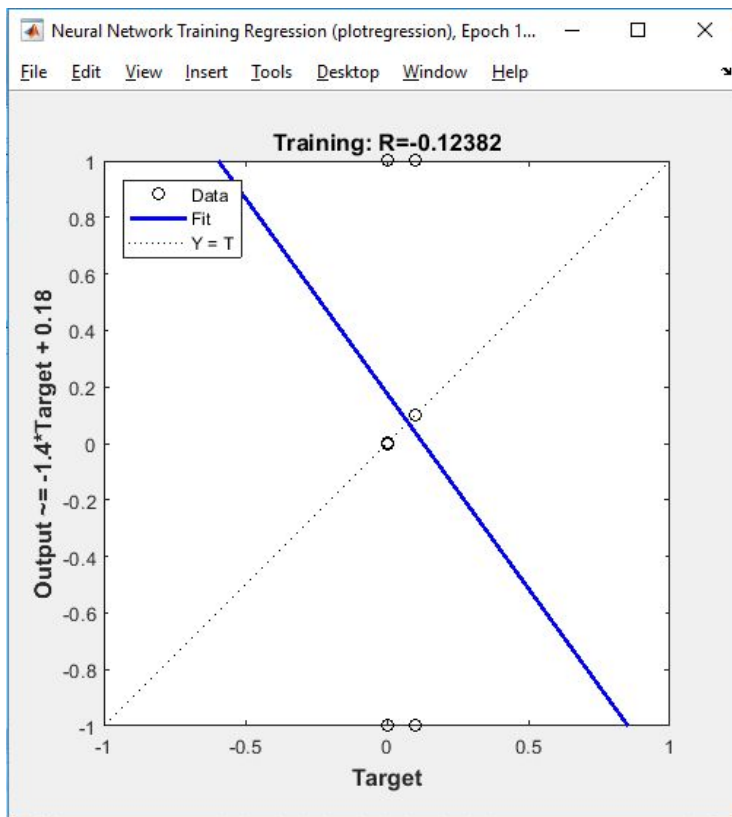
```
lp.dr = 0.5; %wsp. zapominania  
lp.lr = 0.5; %wsp. uczenia
```

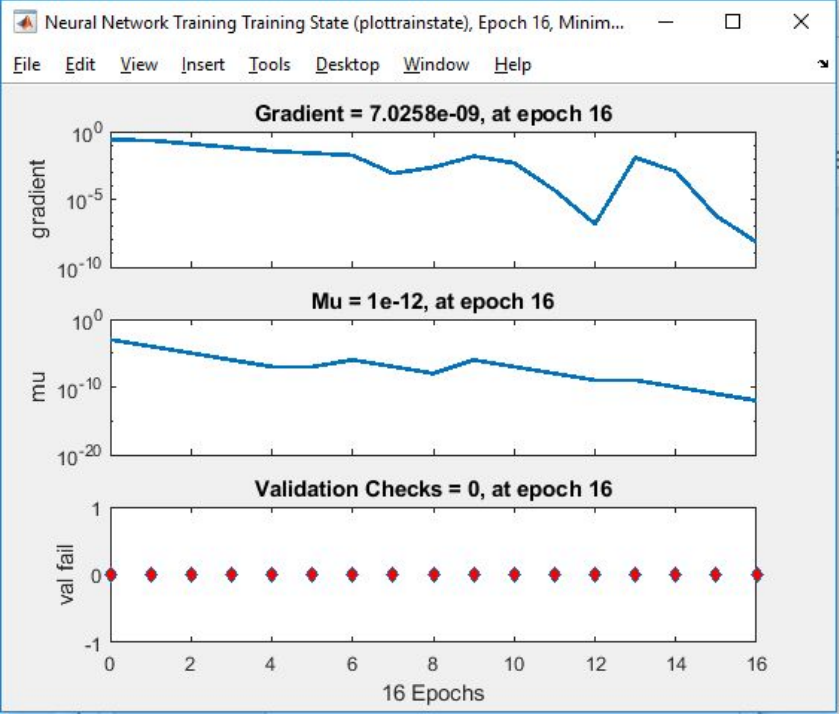




```
lp.dr = 0.1; %wsp. zapominania  
lp.lr = 0.1; %wsp. uczenia
```







• WNIOSKI

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

W porównaniu z procesem uczenia z nauczycielem samo uczenie jest zwykle znacznie wolniejsze.

Wartości < 0 mogą oznaczać że neurony nie traktują wprowadzonych do nich wartości jako własne i wyliczają je na minusie.

-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.

.Wartości wag które ustalimy na początku 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.

Długi czas uczenia się który zajmował ILOŚĆ EPOK - wpływał na lepszą dokładność wyników uczenia. Inne wprowadzonych wartościach sieć uczyła się szybciej, ale osiąga słabsze wyniki.

Podczas analizowania wydajności można zauważyć również miejsce, w którym jakość treningu przestaje znacząco wzrastać - może to świadczyć o tym, że jest to punkt, w którym sieć przestaje się uczyć bez większego progresu oraz wzrasta prawdopodobieństwo na zjawisko przeuczenia sieci.