Jakub Sowiński

Temat ćwiczenia: Budowa i działanie perceptronu

Cel ćwiczenia:

Celem ćwiczenia jest poznanie budowy i działanie perceptronu poprzez implementację oraz uczenie perceptronu realizującego wybraną funkcję logiczną dwóch zmiennych.

Zadania do wykonania:

a) Implementacja sztucznego neuronu wg algorytmu podanego na wykładzie lub dowolnego innego.
b) Wygenerowanie danych uczących i testujących wybranej funkcji logicznej dwóch zmiennych.
c) Uczenie perceptronu dla różnej liczby danych uczących, różnych współczynników uczenia.
d) Testowanie perceptronu.

Wstęp teoretyczny:

Sztuczny neuron - prosty system przetwarzający wartości sygnałów wprowadzanych na jego wejścia w pojedynczą wartość wyjściową, wysyłaną na jego jedynym wyjściu (dokładny sposób funkcjonowania określony jest przez przyjęty model neuronu). Jest to podstawowy element sieci neuronowych, jednej z metod sztucznej inteligencji, pierwowzorem zbudowania sztucznego neuronu był biologiczny neuron.

Perceptron – najprostsza sieć neuronowa, składająca się z jednego bądź wielu niezależnych neuronów, implementująca algorytm uczenia. Perceptron jest funkcją, która potrafi określić przynależność parametrów wejściowych do jednej z dwóch klas. Może być wykorzystywany tylko do klasyfikowania zbiorów.

Algorytmu uczenia perceptronu - tzn. automatycznego doboru wag na podstawie napływających przykładów. W uproszczonym przypadku dwuwymiarowym algorytm wygląda następująco:

- Inicjujemy wagi losowo.
- Dla każdego przykładu uczącego obliczamy odpowiedź perceptronu.
- Jeśli odpowiedź perceptronu jest nieprawidłowa, to modyfikujemy wagi:

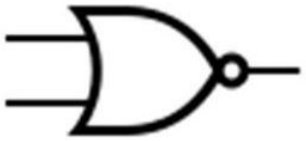
w_1 += n * (d-y) * x_1
w_2 += n * (d-y) * x_2
b += n * (d-y)

gdzie,

n jest niewielkim współczynnikiem uczenia (n > 0),
d - oczekiwana odpowiedź
y - odpowiedź neuronu.

Po wyczerpaniu przykładów zaczynamy proces uczenia od początku, dopóki następują jakiekolwiek zmiany wag połączeń. Warto zwrócić uwagę na graficzną interpretację powyższych wzorów. Każdy źle zaklasyfikowany przypadek powoduje przechylenie prostej oddzielającej pozytywne odpowiedzi perceptronu od negatywnych w kierunku zmierzającym do przerzucenia wadliwego przykładu na drugą stronę prostej.
Opisany schemat jest w miarę przejrzysty tylko dla pojedynczych perceptronów lub niewielkich sieci. Ciężko jest stosować reguły tego typu dla bardziej skomplikowanych modeli. Tymczasem np. do rozpoznawania wszystkich liter potrzeba by sieci złożonej z 26 takich perceptronów, a żeby wyniki rozpoznawania były satysfakcjonujące, powinniśmy użyć wielowarstwowej sieci neuronowej.

Użyte bramki logiczne do przetestowania perceptronu:

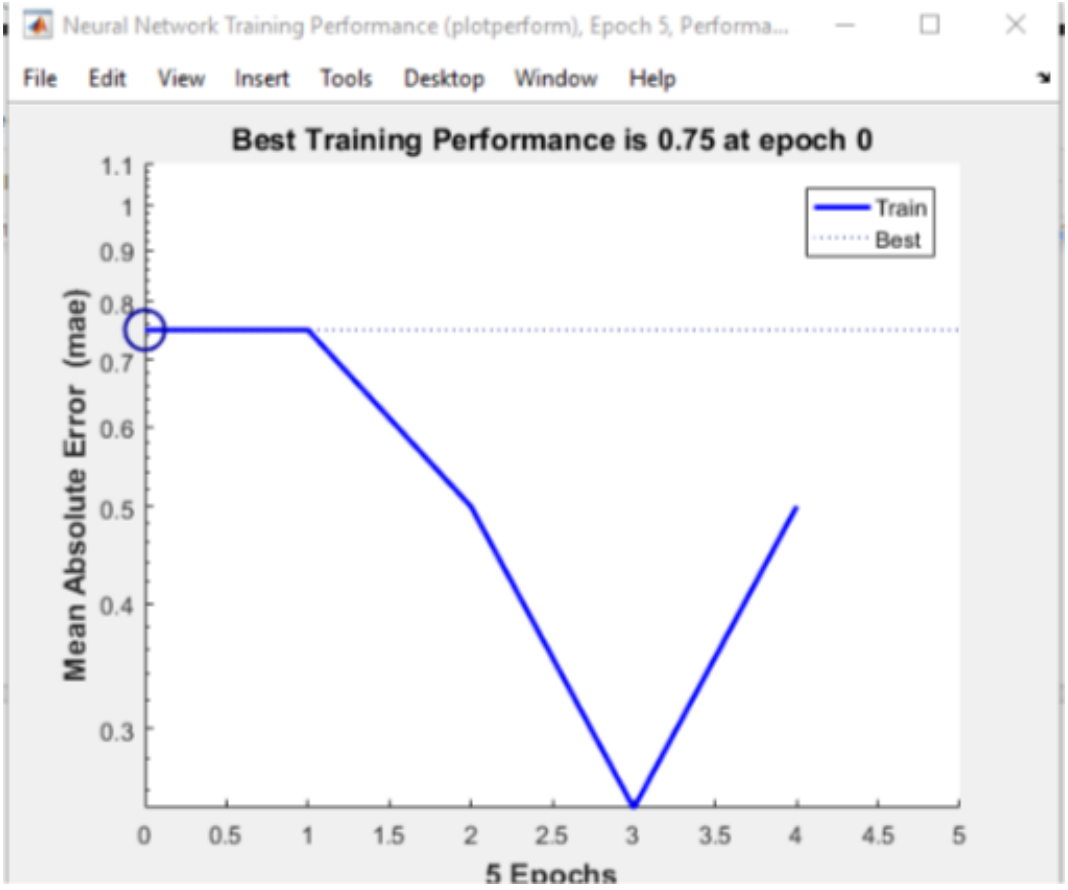
AND		$A \cdot B$	INPUT OUTPUT A B A AND B 0 0 0 0 1 0 1 0 0 1 1 1
OR		$A + B$	INPUT OUTPUT A B A OR B 0 0 0 0 1 1 1 0 1 1 1 1
NAND		$\overline{A \cdot B}$	INPUT OUTPUT A B A NAND B 0 0 1 0 1 1 1 0 1 1 1 0
NOR		$\overline{A + B}$	INPUT OUTPUT A B A NOR B 0 0 1 0 1 0 1 0 0 1 1 0
XOR		$A \oplus B$	INPUT OUTPUT A B A XOR B 0 0 0 0 1 1 1 0 1 1 1 0

Zaimplementowany model neuronu został użyty z pozycji 2) w Literaturze na końcu sprawozdania.

1. **Bramka AND**

```
%% AND
close all; clear all ;clc
net = newp([0 1;-2 2], 1);
P2 = [0 1 0 1;0 0 1 1];
T2 = [0 0 0 1 ];
net = init(net);
Y = sim(net,P2);
%net.trainParam.epochs = 10;
net = train(net,P2,T2);
Y = sim(net,P2)
e=Y-T2
plot(P2,Y-T2,'g',P2,T2,'b',P2,Y,'r')
```

Bramka AND zwraca 1 tylko w przypadku podania dwóch jedynek.
Po osiągnięciu celu, uruchomiona SSN zakończyła naukę. Potrzebne było 5 iteracji w celu skutecznej nauki.



Właściwie to już po 3 epokach(iteracjach) sieć by mogła zakończyć swoje działanie. W tym przypadku nastąpił efekt tzw. przeuczenia sieci, gdy danych uczących jest za dużo w stosunku do danych walidujących i testowych.

Zmienna e pozwala nam sprawdzić czy faktycznie SSN się nauczyła. W naszym przypadku porównujemy dane nauczone z pożądanym celem, czyli wyjściami bramki.
Zmienna e wyświetla w konsoli same 0: e=[0 0 0 0], czyli możemy potwierdzić, że sieć się nauczyła danych i już w pełni może funkcjonować jako bramka AND.

2. Bramka NAND

```
%% NAND
close all; clear all ;clc
net = newp([0 1;-2 2], 1);
P2 = [0 1 0 1;0 0 1 1];
T2 = [1 1 1 0 ];
net = init(net);
Y = sim(net,P2)
%net.trainParam.epochs = 20;
net = train(net,P2,T2);
Y = sim(net,P2)
plot(P2,Y-T2,'g',P2,T2,'b',P2,Y,'r')
```

Działanie podobne jak bramka AND, jednak zaprzeczamy jej wyjścia, czyli tam gdzie 0 teraz jest 1 i na odwrót.
Tak samo jak w przypadku AND, SSN nauczyła się w przeciągu 5 iteracji.
Dokładniejsze stwierdzenia jednak nam pokaże performance:

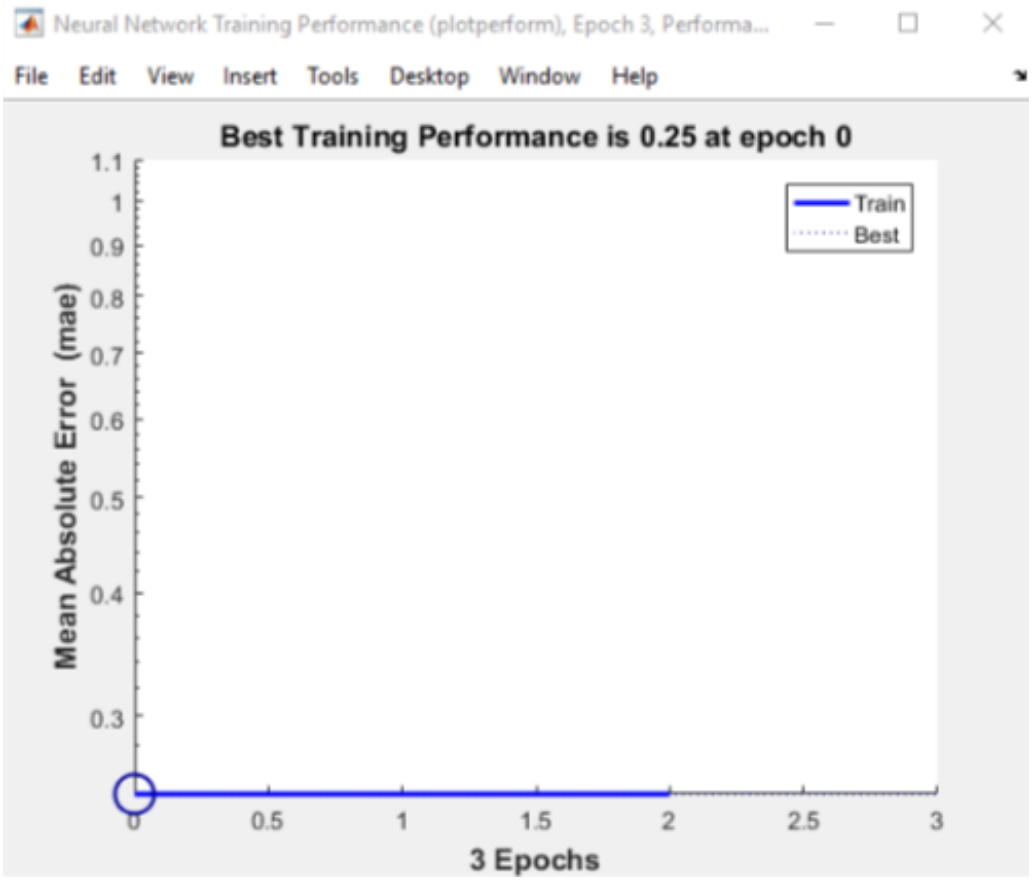
Właściwie to najniższy możliwy średni błąd był na samym początku i w 3 epoce. Tak samo jak w przypadku bramki AND nastąpiło przeuczenie danymi i średni błąd zamiast zmierzać do 0 rośnie.
Zmienna e wyświetla w konsoli same 0: e=[0 0 0 0], czyli możemy potwierdzić, że sieć się nauczyła danych i już w pełni może funkcjonować jako bramka NAND.

3. Bramka OR

```
%% or
close all; clear all ;clc
net = newp([0 1;-2 2], 1);
P2 = [0 1 0 1;0 0 1 1];
T2 = [0 1 1 1 ];
net = init(net);
Y = sim(net,P2);
%net.trainParam.epochs = 20;
net = train(net,P2,T2);
Y = sim(net,P2);
e=Y-T2
plot(P2,Y-T2,'g',P2,T2,'b',P2,Y,'r')
```

Bramka zwraca 1 w przypadku wprowadzenia przynajmniej jednej jedynki.

Z powyższego okna dowiadujemy się, że sieć nauczyła się w przeciągu 3 iteracji. Po sprawdzeniu jednak okna Performance:



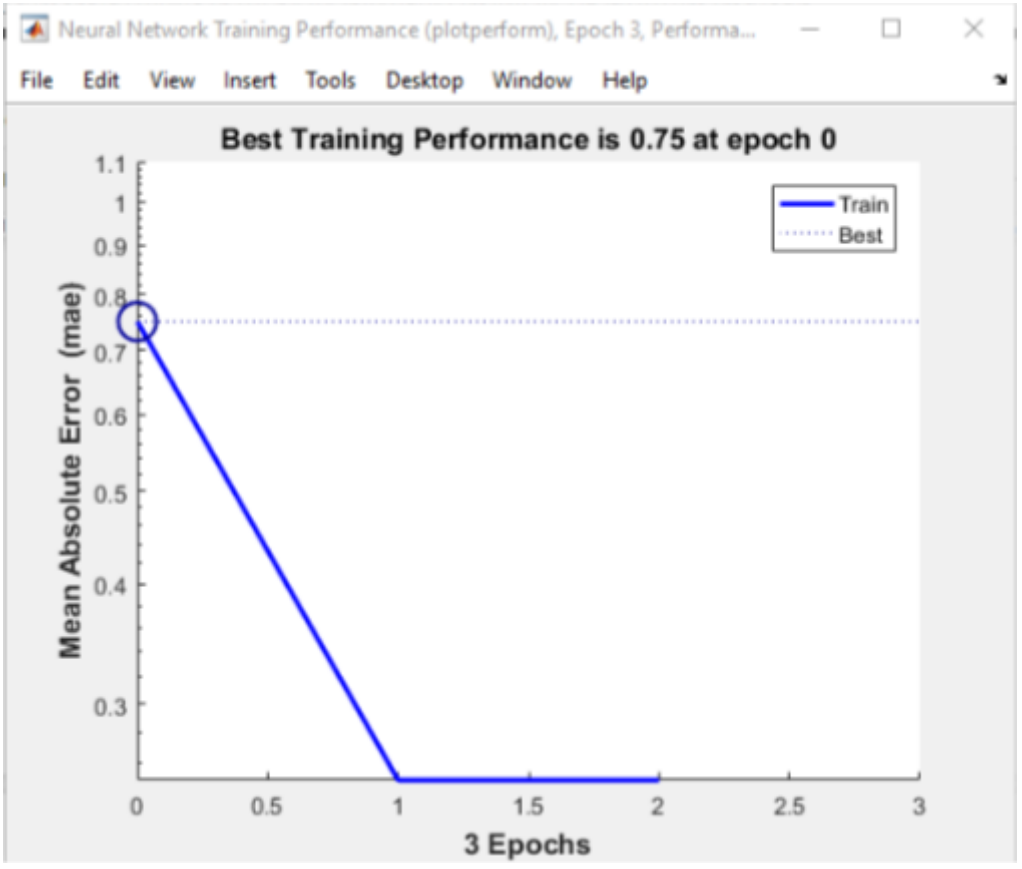
Widzimy tu, że średni błąd był stały i wynosił 0.25 i w zasadzie nauka zakończyła się po 2 epokach. Jak narazie SSN dla bramki OR potrzebuje najmniej iteracji na naukę.

Zmienna e wyświetla w konsoli same 0: e=[0 0 0 0], czyli możemy potwierdzić, że sieć się nauczyła danych i już w pełni może funkcjonować jako bramka OR.

4. Bramka NOR

```
%% NOR
close all; clear all ;clc
net = newp([0 1;-2 2], 1);
P2 = [0 1 0 1;0 0 1 1];
T2 = [1 0 0 0 ];
net = init(net);
Y = sim(net,P2);
%net.trainParam.epochs = 20;
net = train(net,P2,T2);
Y = sim(net,P2);
plot(P2,Y-T2,'g',P2,T2,'b',P2,Y,'r')
e=Y-T2
```

Bramka NOR ma podobne działanie jak OR tylko jest jej zaprzeczeniem i 1 jest wyjściem dla dwóch zer.
Sztuczna sieć neuronowa potrzebowała 3 iteracji na naukę bramki NOR.
Przypuszczam, że wykres Performance wygląda podobnie jak dla OR. Jednak upewnijmy się.

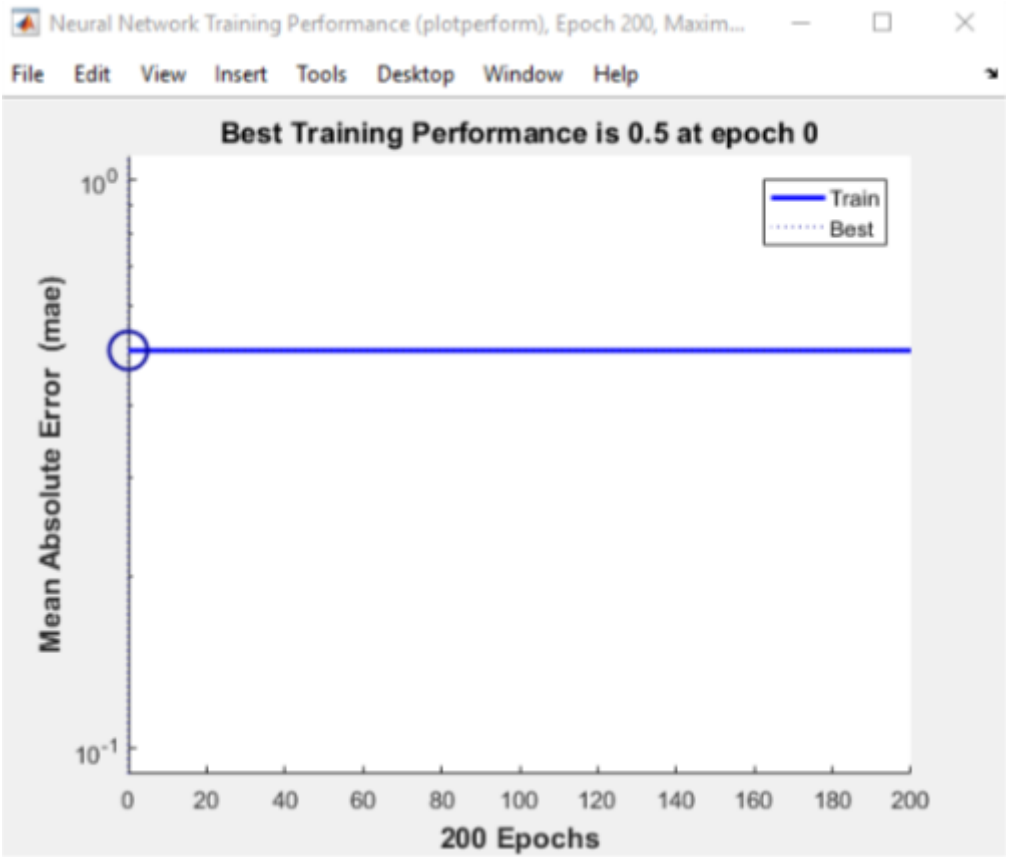


Jednak nie, najwyższy błąd wynosił 0.75 na samym początku. po uczeniu w pierwszej iteracji osiągnął błąd 0.25 i już był stały do końca.
Zmienna e wyświetla w konsoli same 0: e=[0 0 0 0], czyli możemy potwierdzić, że sieć się nauczyła danych i już w pełni może funkcjonować jako bramka NOR.

5. Bramka XOR

```
%% XOR
close all; clear all ;clc
net = newp([0 1;-2 2], 1);
P2 = [0 0 1 1;0 1 0 1];
T2 = [0 1 1 0 ];
net = init(net);
Y = sim(net,P2);
net.trainParam.epochs = 200;
net = train(net,P2,T2);
Y = sim(net,P2);
e=Y-T2
plot(P2,Y-T2,'g',P2,T2,'b',P2,Y,'r')
```

Bramka XOR zwraca 1 w przypadku dwóch zer lub dwóch jedynek po uruchomieniu programu widzimy dziwną sytuację, sieć w ogóle się nie nauczyła tylko osiągnęła dany limit możliwych iteracji. To samo jest w Performance:



Również i tu się dowiadujemy, że z czasem kolejnych epok, sieć się w ogóle nie uczy tylko osiąga stały błąd równy 0.5.

Wynika to z faktu, że bramka XOR potrzebuje 2 neuronów w sieci a nie tak jak ustaliliśmy na samym początku 1 neuronu. XOR jest nieliniowy i dlatego ta bramka nie będzie działać w naszym przypadku.

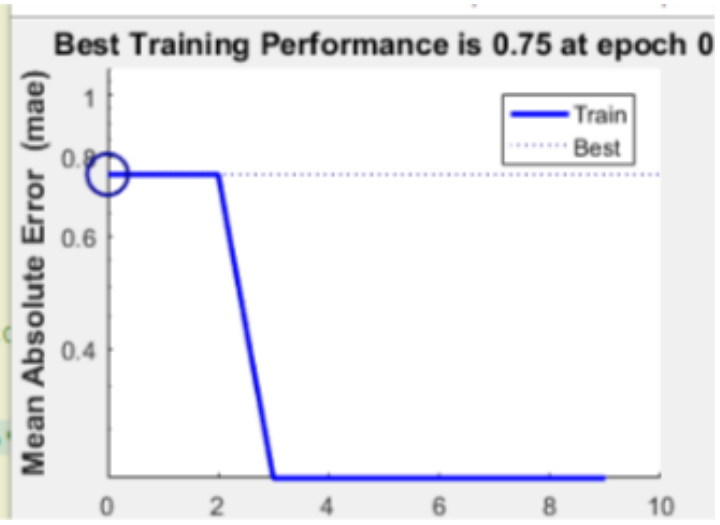
6. Uczenie perceptronu

Uczenie perceptronu przyjąłem jako zmianę wag **net.IW** oraz zmianę współczynnika b przy **net.b**

```
net.IW{1}=[2 3];
net.b{1}=[3]
```

Bramka NOR:

```
%% NOR
close all; clear all ;clc
net = newp([0 1;-2 2], 1);
P2 = [0 1 0 1;0 0 1 1];
T2 = [1 0 0 0 ];
%net = init(net);
net.IW{1}=[7 6];
net.b{1}=[3]
Y = sim(net,P2);
%net.trainParam.epochs = 2
net = train(net,P2,T2);
Y = sim(net,P2);
%plot(P2,Y-T2,'g',P2,T2,'b'
e=Y-T2
```

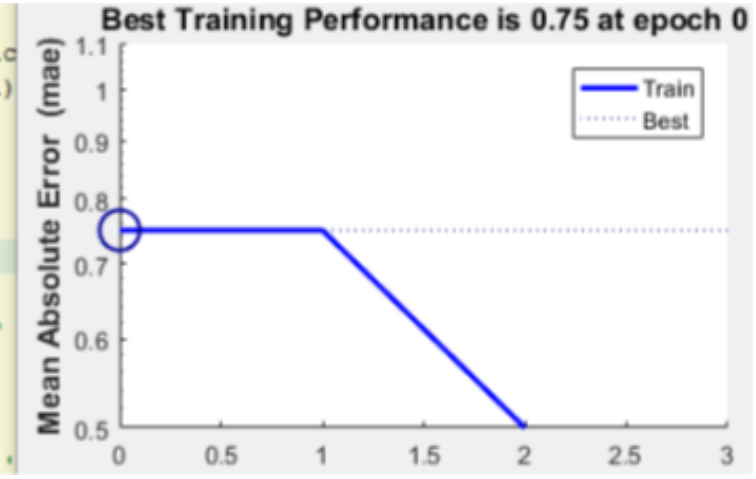



```
%% NOR
close all; clear all ;clc
net = newp([0 1;-2 2], 1);
P2 = [0 1 0 1;0 0 1 1];
T2 = [1 0 0 0 ];
%net = init(net);
net.IW{1}=[20 17];
net.b{1}=[2]
Y = sim(net,P2);
%net.trainParam.epochs = 20;
net = train(net,P2,T2);
Y = sim(net,P2);
%plot(P2,Y-T2,'g',P2,T2,'b',P
e=Y-T2
```

Bramka AND:

```
%% AND
close all; clear all ;clc
net = newp([0 1;-2 2], 1)
P2 = [0 1 0 1;0 0 1 1];
T2 = [0 0 0 1 ];
net = init(net);
net.IW{1}=[2 3];
net.b{1}=[1]
Y = sim(net,P2);
%net.trainParam.epochs =
net = train(net,P2,T2);
Y = sim(net,P2)
e=Y-T2
```


```
%% AND
close all; clear all ;clc
net = newp([0 1;-2 2], 1)
P2 = [0 1 0 1;0 0 1 1];
T2 = [0 0 0 1 ];
net = init(net);
net.IW{1}=[7 7];
net.b{1}=[3]
Y = sim(net,P2);
%net.trainParam.epochs =
net = train(net,P2,T2);
Y = sim(net,P2)
e=Y-T2
%plot(P2,Y-T2,'g',P2,T2,'
```

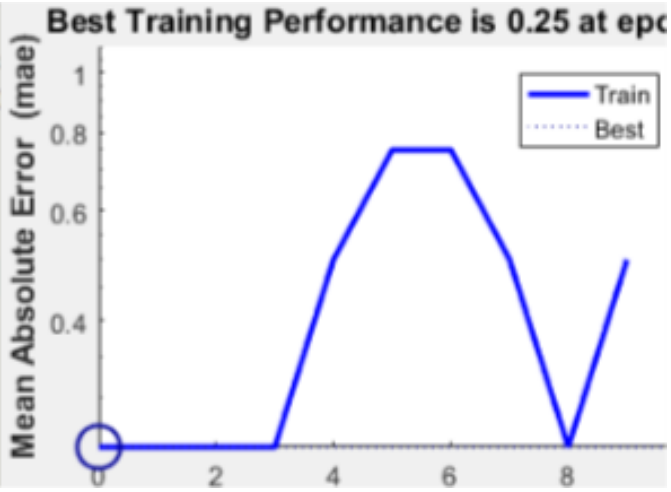



```
%% AND
close all; clear all ;cl
net = newp([0 1;-2 2], 1
P2 = [0 1 0 1;0 0 1 1];
T2 = [0 0 0 1 ];
net = init(net);
net.IW{1}=[9 13];
net.b{1}=[5]
Y = sim(net,P2);
%net.trainParam.epochs =
net = train(net,P2,T2);
Y = sim(net,P2)
e=Y-T2
```

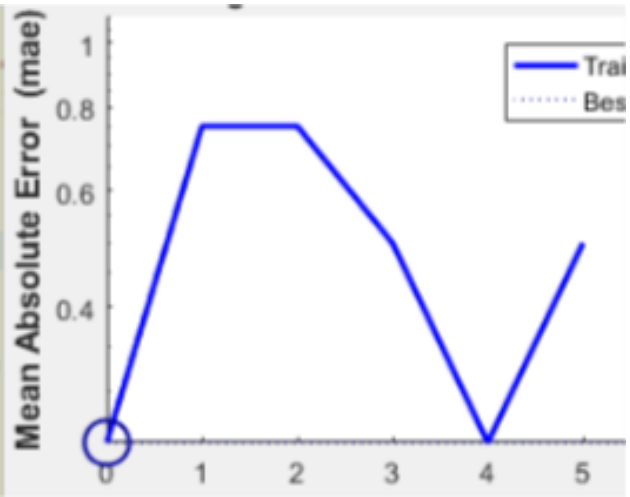
Bramka NAND:

```
%% NAND
close all; clear all ;c
net = newp([0 1;-2 2],
P2 = [0 1 0 1;0 0 1 1];
T2 = [1 1 1 0 ];
net = init(net);
net.IW{1}=[9 9];
net.b{1}=[4]
Y = sim(net,P2);
%net.trainParam.epochs
net = train(net,P2,T2);
Y = sim(net,P2);
e=Y-T2
```


```
%% NAND
close all; clear all ;cl
net = newp([0 1;-2 2], 1
P2 = [0 1 0 1;0 0 1 1];
T2 = [1 1 1 0 ];
net = init(net);
net.IW{1}=[5 5];
net.b{1}=[3]
Y = sim(net,P2);
%net.trainParam.epochs =
net = train(net,P2,T2);
Y = sim(net,P2);
e=Y-T2
```

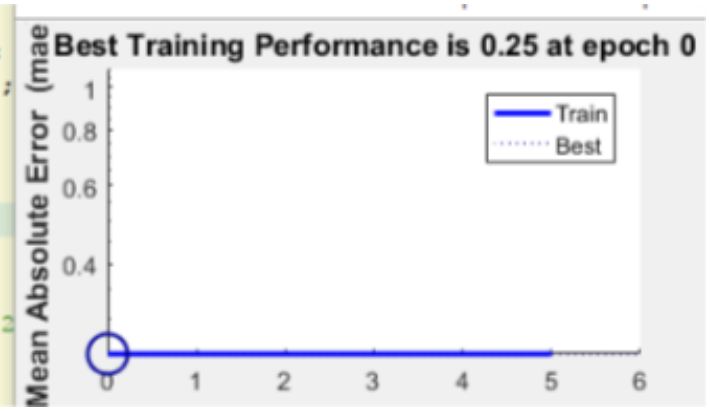



```
%% NAND
close all; clear all ;cl
net = newp([0 1;-2 2], 1
P2 = [0 1 0 1;0 0 1 1];
T2 = [1 1 1 0 ];
net = init(net);
net.IW{1}=[1 1];
net.b{1}=[0]
Y = sim(net,P2);
%net.trainParam.epochs =
net = train(net,P2,T2);
Y = sim(net,P2);
e=Y-T2
```



Bramka OR:
Dla wszystkich zmienionych wartości wykres wyglądał tak samo, jak ten poniżej:

```
%% or
close all; clear all ;clc
net = newp([0 1;-2 2], 1);
P2 = [0 1 0 1;0 0 1 1];
T2 = [0 1 1 1 ];
%net = init(net);
net.IW{1}=[7 8];
net.b{1}=[5]
Y = sim(net,P2);
%net.trainParam.epochs = 2
net = train(net,P2,T2);
Y = sim(net,P2);
```



6. Podsumowanie i wnioski

Wszystkie zamierzone cele zostały pomyślnie osiągnięte. Poznałem budowę perceptronu, jego implementację w programie MatLab, oraz sposób uczenia perceptronu. Całość została zrealizowana na podstawie bramek logicznych: AND,NAND, OR, NOR, XOR. Z tym, że ta ostatnia jest nieliniowa więc nie można jej zbudować na podstawie jednego neuronu.

Zazwyczaj uczenie zamyka się w przeciągu kilku iteracji. Bramki OR i NOR potrzebowały mniej epok, na naukę niż AND i NAND.

Zmiana wag również wpływa na długość procesu uczenia się. Gdy podamy wysokie wagi wejściowe neuronu, to nawet kilkukrotnie wydłuża się czas nauki. Również wagi wpływają na kształt wykresu postępu uczenia się. Wszystko to zawiera się w tym jak skomplikowany był proces uczenia się. Wysokie wagi w stosunku do danych wejściowych (0 i 1) bardzo wydłużały proces nauki perceptronu.

Literatura:

- http://edu.pjwstk.edu.pl/wyklady/nai/scb/wyklad3/w3.htm
- Podstawowe funkcje biblioteki narzędziowej„Neural Network Toolbox. Version 5” pakietu MATLAB v. 7.1, POLITECHNIKA BIAŁOSTOCKA
- wikipedia.org