Grupa lab. ***1***	*Przedmiot* ***Podstawy Sztucznej Inteligencji***	*Data wykonania.* ***26.10.2018***
Nr ćwicz. ***1***	*Temat ćwiczenia.* ***Budowa i działanie perceptronu***	
Imię i nazwisko. ***Katarzyna Giądła***		*Ocena i uwagi*

Część teoretyczna

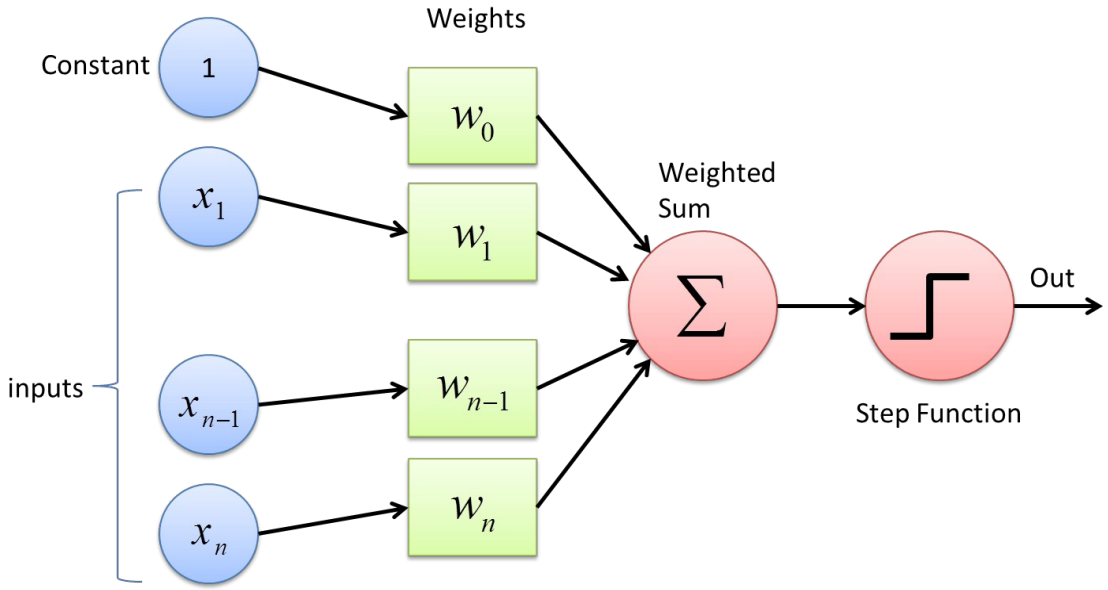
Celem tego projektu było poznanie budowy i działania perceptronu poprzez implementację oraz uczenie perceptronu realizującego wybraną funkcję logiczną dwóch zmiennych.

Perceptron jest prostym modelem naturalnego neuronu. Składa się jedynie z warstwy wejściowej (*n* zmiennych wejściowych $x_1, x_2, \ldots, x_n$ przyjmujące wartości rzeczywiste lub binarne) i warstwy wyjściowej (dla omawianego w naszym przypadku perceptronu prostego stanowi pojedynczą wartość *0* lub *1* – taka funkcja nazywana jest funkcją unipolarną). Parametrami wewnętrznymi perceptronu to *n* wag połączeń $w_1, w_2, \ldots, w_n$ (ze zbioru liczb rzeczywistych) oraz wartość odchylenia *b* odpowiadająca za nieliniowe przekształcenie wejść w wyjście.

$$f(z)=\begin{cases}1 \Leftrightarrow z \geq 0\\ 0 \Leftrightarrow z < 0\end{cases}$$

$$e=\sum_{i=1}^{n} w_i x_i + w_0 = \sum_{i=0}^{n} w_i x_i \quad \Leftarrow x_0 := 1$$

Model, jaki przyjęliśmy do budowy naszego sztucznego neuronu to **model McCullocha-Pittsa.** Polega on na obliczaniu sum ważonych sygnałów wejściowych, a następnie sumowaniu progu aktywacji w_0. W tym modelu występuje też pewna funkcja aktywacji *f* zależna od wyliczonej sumy ważonej.



Perceptron posiadający *n* wejść dzieli *n*-wymiarową przestrzeń wektorów wejściowych *x* na 2 półprzestrzenie, które są podzielone *(n-1)*-wymiarową hiperpłaszczyzną, zwn. **Granicą decyzyjną**.

Algorytm uczenia perceptronu:

1. W sposób losowy wybieramy wagi początkowe perceptronu w_i
2. Na wejściu perceptronu podajemy kolejny wektor uczący *X*
3. Obliczamy wartość wyjściową perceptronu *y(x)*.
4. Porównujemy uzyskaną wartość wyjściową *y(x)* z wartością wzorcową *d* dla wektora *X*.
5. Dokonujemy modyfikacji wag według zależności:
   Jeżeli $y(x) \neq d(x)$, to $w_0 = w_0 + d(x)$ oraz $w_i = w_i + d(x) * x_i$.
   W przeciwnym przypadku waga się nie zmienia.
6. Obliczamy średni błąd dla wszystkich wzorców uczących.
7. Jeżeli błąd jest mniejszy od założonego lub osiągnięto maksymalną ilość powtórzeń zbioru uczącego przerwij algorytm.
   W przeciwnym razie przejdź do kroku 2.

Część praktyczna

Do wykonania tego projektu zostało użyte środowisko MATLAB, ponieważ posiada ono narzędzie *Neural Networking Training Tool*. Dzięki temu pakietowi możemy tworzyć proste sieci neuronowe oraz dostosowywać niektóre parametry algorytmów uczenia.

Poniżej opiszę kilka z funkcji użytych w kodzie źródłowym:

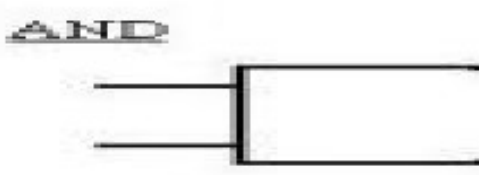
- *net = newp(p, t, tf)* → funkcja tworząca prosty perceptron, gdzie *p* – macierz składająca się z wektorów wejściowych, *t* – macierz reprezentująca ilość neuronów, a *tf* – funkcja uczenia – domyślnie jest to *'hardlim'* (przyjmujemy, że posługujemy się funkcją unipolarną)
- *plotpv(P, T)* → funkcja rysująca wykresu z wynikami
- *plotpc(W, B)* → wykres granicy decyzyjnej
- *train(net, X, T)* → funkcja uczenia, gdzie struktura *net* zawiera informacje o parametrach treningów, *X* jest macierzą danych wejściowych, a *T* jest macierzą danych wynikowych
- *sim(net, X)* → przeprowadza symulację opartą na parametrach w obiekcie *net* i operując na danych wejściowych.
- *randi([imin imax], sizmin, sizmax)* → tablica losowych wartości całkowitych o rozmiarze *sizmin x sizmax*

Dzięki obiektowi *net* możemy dowolnie sterować opcjami treningu, testowania i symulacji. Szczególnie istotne w naszym przypadku okazały się następujące parametry:

- *net.trainParam.epochs* – maksymalna liczba cykli uczenia
- *net.trainParam.goal* – próg, w którym uznajemy zadanie za wykonane (założony błąd)
- *net.trainParam.mu* – parametr regulujący błąd średniokwadratowy

Można również dostosowywać m.in. co ile cykli uczenia powinny ukazywać się wyniki, czy wyświetlanie treningu w GUI. Dla naszych potrzeb te ustawienia pozostały na poziomie wartości domyślnych.

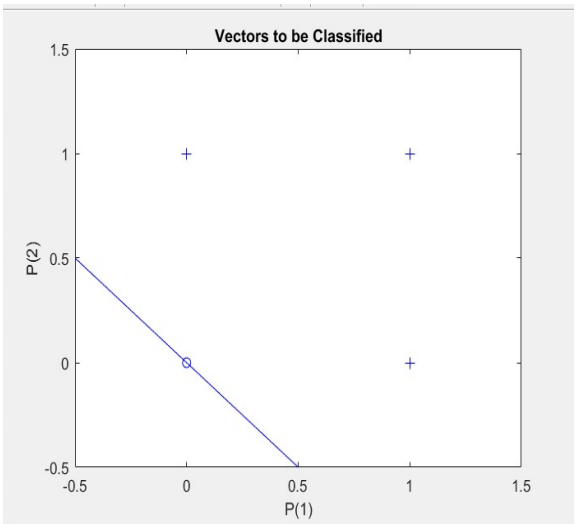
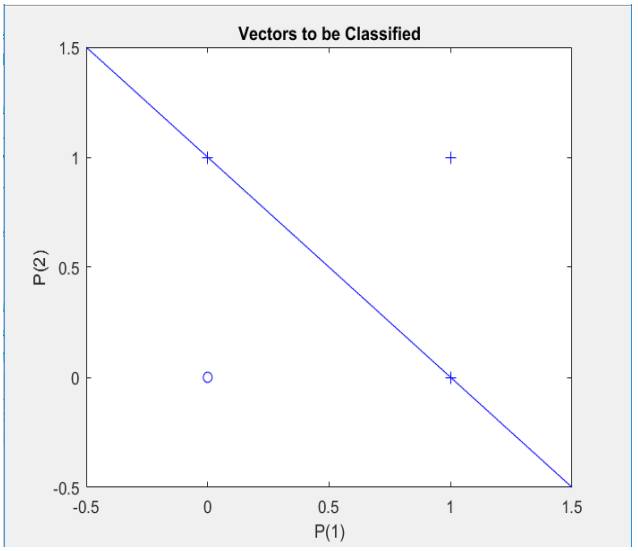
Wartość logiczną, którą postanowiłam nauczyć perceptron była funkcja AND (zwn. koniunkcją).

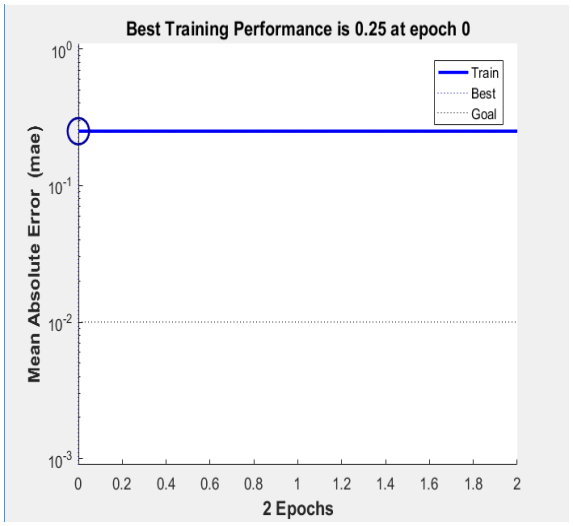


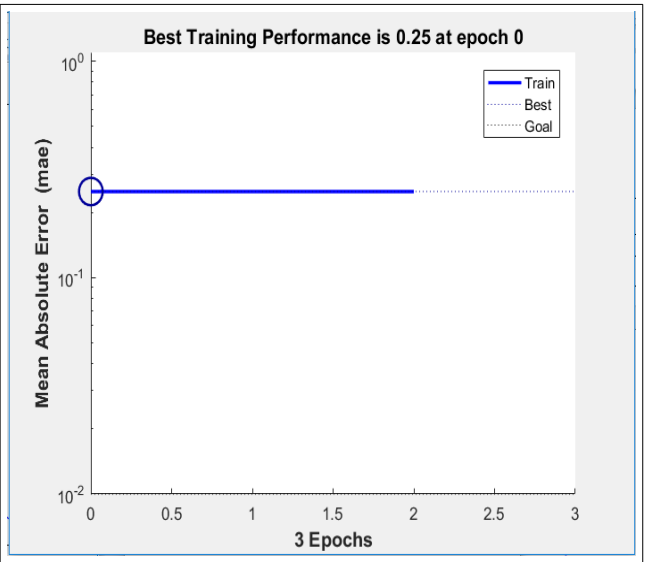
Wyniki

Testy zostały przeprowadzone po 2, 4, 6, …, 12 epok uczenia. Każdy eksperyment dla każdej epoki był przeprowadzany 2 razy, wraz ze zmianą progu osiągnięcia celu z *0.01* do *0.005*. Na czerwono oznaczono błędne wyniki.

Liczba epok treningowych	2	4	6	8	10	12
Wynik próby	test = 0 0 1 1 efekt = 1 1	test = 0 1 1 1 0 0 efekt = 1 1 1	test = 1 0 0 1 1 1 efekt = 1 1 1	test = 0 0 1 0 0 0 efekt = 0 0 1	test = 1 0 0 1 1 0 1 0 efekt = 1 0 1 1	test = 0 1 0 0 efekt = 0 1
Wynik próby ze zmianą progu osiągnięcia celu	test = 0 0 0 0 1 1 efekt = 1 1 1	test = 1 1 1 0 0 0 efekt = 1 1 1	test = 0 0 1 0 efekt = 1 0	test = 0 1 1 0 0 0 efekt = 0 1 1	ttest = 1 1 0 0 efekt = 1 1	test = 0 1 1 1 0 1 efekt = 1 1 1

Błędne wyniki przy niedoborze nauki – 2 epoki	***Najbardziej prawidłowe wyniki – powyżej 3 epok***
	





Wnioski

- Po osiągnięciu epoki 3 sieć jest przeuczona (próg sukcesu został osiągnięty) – jednak mimo to perceptron myli się. Jest to jedna z cech sieci neuronowych. Uczą się, analizują, ale mimo to popełniają błędy.
- Zmniejszenie progu osiągnięcia sukcesu nie wpłynęło korzystnie na wyniki testów – można postawić tezę, że im mniejsze stawiamy wymagania perceptronowi, tym gorzej wypada proces uczenia.
- Należy uczyć perceptrony na podstawie całej tablicy prawdy, ponieważ maszyna uczy się tylko tyle, ile zada jej programista.
- MATLAB daje wiele narzędzi do przeprowadzania analiz związanych z budową sieci neuronowych. Dzięki obiektom możemy dowolnie edytować ustawienia treningu oraz sterować parametrami za pomocą GUI.

Listing kodu źródłowego

```
close all; clear all; clc;

%%bramka AND
net = newp([0 1; 0 1], 1, 'hardlim'); %generowanie nowego neutronu
dane_in = [0 0 1 1;
           0 1 0 1];
dane_out = [0 1 1 1]; %dane_in, dane_out - wzorce uczenia
plotpv(dane_in, dane_out) %wyświetlanie wykresu danych wejściowych i wyjściowych

net.trainParam.epochs = 12; %ilość cykli (epok) uczenia
net.trainParam.goal = 0.005; %próg osiągnięcia celu
net.trainParam.mu = 0.01; %błąd średniokwaratowy
%net.trainParam.showWindow = false; - wyswietlanie interfejsu GUI

net = train(net, dane_in, dane_out); %proces uczenia
plotpc(net.iw{1, 1}, net.b{1}) %wykres stworzony z komórek pamięci, w kórych były
przechowywane wagi wejściowe oraz odchylenie b

Y = sim(net, dane_in); %test - czy perceptron nauczył się
test = randi([0 1], 2, randi([2, 5])); %generowane losowe wart. logiczne z 2 wierszami i
losową liczbą kolumn (<=5)
efekt = sim(net, test); %wyświetlenie wyników tego, co nauczył się perceptron

test
efekt
```