Politechnika Łódzka **Instytut Informatyki Stosowanej**

Podstawy Sztucznej Inteligencji

Laboratorium

Ćwiczenie 2

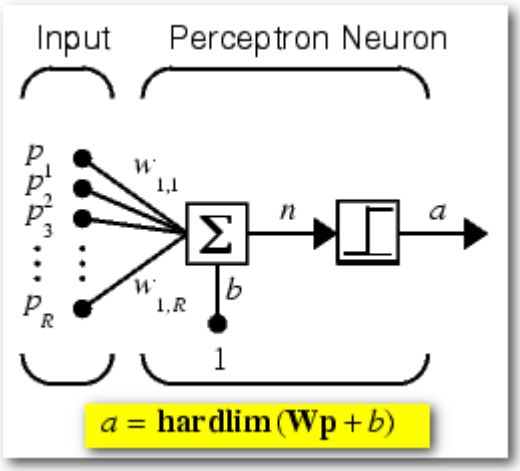
Wykorzystanie środowiska Matlab do modelowania sztucznych sieci neuronowych

Opracowali:
Dr hab. inż. Jacek Kucharski
Dr inż. Piotr Urbanek.

Część **1.** Zapoznanie się z podstawowymi funkcjami do tworzenia perceptronu, neuronu liniowego, sieci neuronów z warstwami ukrytymi.

1. **Tworzenie perceptronu za pomocą funkcji i metod środowiska MATLAB.**

   Modelowany perceptron przedstawiony na rys.1



Rys.1. Model perceptronu o R-wejściach oraz unipolarną funkcją aktywacji.

Do tworzenia pojedynczego perceptronu pokazanego na rys. 1 lub sieci perceptronów służy funkcja ***newp*** (w wersji 2007b) lub ***perceptron*** (we współczesnych wersjach Matlaba)
Wywołuje się ją następująco:

```
P = [0 2];
T = [0 1];
net = newp(P,T);
```

gdzie: P – wektor uczący,
T – wektor wzorcowy
Dla tak zdefiniowanych wektorów P i T perceptron będzie miał jedno wejście i jedno wyjście.

Po wywołaniu funkcji **newp** tworzony jest w przestrzeni roboczej obiekt net, którego własności podzielone na wyszczególnione klasy można odczytać wywołując jego nazwę (net).

net =

Neural Network object:

architecture:

- numInputs: 1
- numLayers: 1
- biasConnect: [1]
- inputConnect: [1]

- layerConnect: [0]
- outputConnect: [1]
- numOutputs: 1 (read-only)
- numInputDelays: 0 (read-only)
- numLayerDelays: 0 (read-only)

subobject structures:

- inputs: {1x1 cell} of inputs
- layers: {1x1 cell} of layers
- outputs: {1x1 cell} containing 1 output
- biases: {1x1 cell} containing 1 bias
- inputWeights: {1x1 cell} containing 1 input weight
- layerWeights: {1x1 cell} containing no layer weights

functions:

- adaptFcn: 'trains'
- divideFcn: (none)
- gradientFcn: 'calcgrad'
- initFcn: 'initlay'
- performFcn: 'mae'
- plotFcns: {'plotperform','plottrainstate'}
- trainFcn: 'trainc'

parameters:

- adaptParam: .passes
- divideParam: (none)
- gradientParam: (none)
- initParam: (none)
- performParam: (none)
- trainParam: .show, .showWindow, .showCommandLine, .epochs,
- goal, .time

weight and bias values:

- IW: {1x1 cell} containing 1 input weight matrix
- LW: {1x1 cell} containing no layer weight matrices
- b: {1x1 cell} containing 1 bias vector

other:

- name: ''
- userdata: (user information)

Jest to najprostszy sposób zdefiniowania sieci perceptronowej. Drugim jest wywołanie funkcji newp z następującymi parametrami:

newp([wart_min wart_maks;wart_min wart_maks;…] , liczba_neuronów)

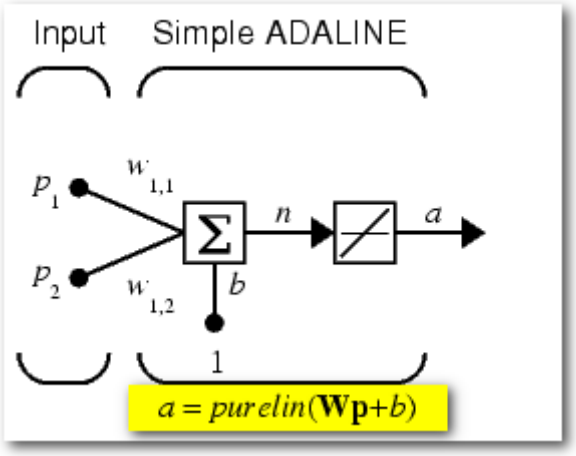
Gdzie: **wart_min, wart_maks**- spowiedziewane zakresy poszczególnych wejść perceptronu, **Liczba_neuronów** – liczba perceptronów w sieci.
Zatem definicja
net = newp([-1 1;-1 1],1) – oznacza jeden perceptron o dwóch wejściach, których wartości zmieniają się w granicach [-1 1].

Zatem program wykorzystujący sieć perceptronową do klasyfikacji dwóch zbiorów mógłby mieć następującą postać:

P = [-0.5 -0.5 +0.3 -0.1; ... -0.5 +0.5 -0.5 +1.0];	Wektor uczący
T = [1 1 0 0];	Wektor wzorcowy
plotpv(P,T);	Rysowanie punktów na płaszczyźnie kartezjańskiej
net = newp([-1 1;-1 1],1);	Def. perceptronu
plotpv(P,T);	
net.adaptParam.passes = 3;	3 pętle doboru wag i przesunięcia
net = adapt(net,P,T);	Dobór wag i przesunięcia
wagi=net.IW{1,1}	
przesuniecie=net.b{1}	Wyświetlenie wartości wag i przesunięcia
plotpc(net.IW{1},net.b{1});	Rysowanie wag perceptronu
p = [0.7; 1.2];	Zdefiniowanie punktu sprawdzającego
a = sim(net,p);	Sprawdzenie działania nauczonego perceptronu

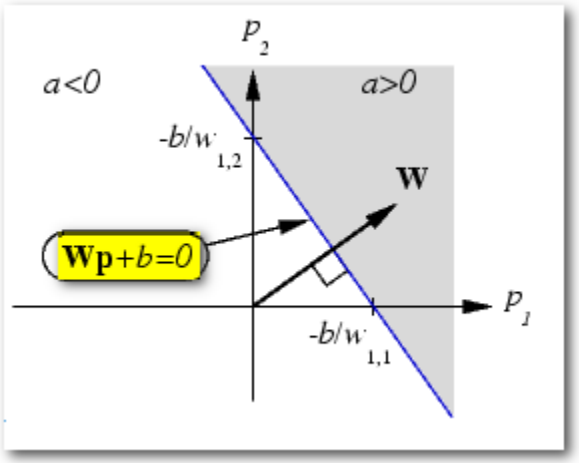
2. Modelowanie neuronu z liniową funkcją aktywacji (ADALINE).

Przykładowy schemat neuronu pokazuje rys.2



Rys. 2. Przykładowy neuron dwuwejściowy z liniową funkcję aktywacji.

Neuron taki może oprócz (podobnie jak perceptron) być wykorzystany do klasyfikacji zbiorów na płaszczyźnie kartezjańskiej przedstawionej na rys. 3.

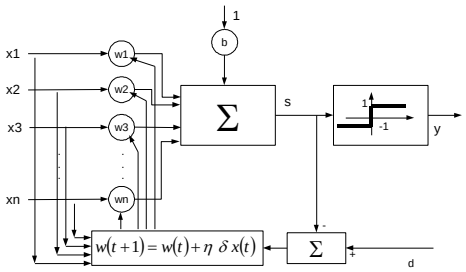


Rys.3. Klasyfikacja punktów na płaszczyźnie.

Do utworzenia neuronu z liniową funkcją aktywacji służy polecenie **newlin**. Przykładowy skrypt realizujący tworzenie neuronu liniowego, odczyt oraz ręczną zmianę wag oraz symulację jego działania.

```
net = newlin([-1 1; -1 1],1);              %Tworzenie neuronu z liniową funkcją aktywacji
W = net.IW{1,1}       %Odczyt wartości początkowych wag
b = net.b{1}    %Odczyt wartości początkowych przesunięcia
net.IW{1,1} = [2 3];  %Przykład ręcznej zmiany wartości wag
net.b{1} = -4;             % Przykład ręcznej zmiany wartości przesunięcia
p = [5; 6];       %Wartość współrzędnych wektora wejściowego
a = sim(net,p) %Symulacja działania pojedynczego neuronu z liniową funkcją aktywacji
```

Neuron ADALINE może służyć do linearyzacji funkcji nieliniowych.
Algorytm uczenia jest następujący:



$$Q(w) = \frac{1}{2}\varepsilon^2 = \frac{1}{2}\left[d - \left(\sum_{i=0}^{n} w_i x_i\right)\right]^2$$

$$w_i(t+1) = w_i(t) + \eta\,\delta\,x_i$$

Gdzie η – współczynnik uczenia neuronu.
δ - błąd między wzorcem a wyjściem sieci. δ=d-s

Przykładowy skrypt realizujący uczenie neuronu funkcji liniowej:

```
clear;
clc;

% Definiowanie wektorów 1 elementowych, kolumnowych

P = [ 1.0   2.0   3.0; ...
```

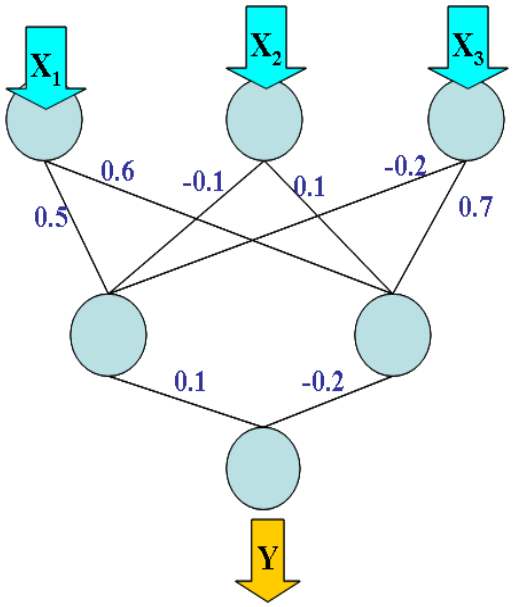
```
    4.0   5.0   6.0];

T = [0.5 1.0 -1.0];

% Tworzenie neuronu liniowego minimalizującego sumę kwadratów błędów
% pomiędzy odpowiedzią neuronu Y a wzorcem uczącym.

lr=0.1; %współczynnik uczenia – parametr dobierany.
net = newlin([0 10;0 10],1,[0],lr);

net.trainParam.show = 50;     % Częstotliwość odświeżania wyników uczenia na ekranie
net.trainParam.epochs = 1000;  % Maksymalna liczba epok
net.trainParam.goal = 0.001;  % Wartość błędu średniokwadratowego pomiędzy wzorcem a
wyjściem sieci.
[net,tr,Y] = train(net,P,T); %Trening sieci

Y1=sim(net,[1.5;2]);
```

Zadanie do wykonania:
W powyższym skrypcie zdefiniowano 3 różne punkty na płaszczyźnie odpowiadające trzem różnym klasom [0.5 1 -1]. Zaobserwować, czy neuron ADALINE poprawnie uczy się odwzorowania trzech powyższych klas? Opisać w sprawozdaniu wpływ wartości współczynnika uczenia lr na jakość uczenia. Opisać spostrzeżenia.

NAUKA SIECI NEURONOWEJ UCZONEJ ALGORYTMEM WSTECZNEJ PROPAGACJI BŁĘDÓW.

Uczenie sztucznej sieci neuronowej bardziej skomplikowanych przebiegów wymaga utworzenia sieci uwarstwionej z odpowiednią liczbą neuronów w warstwie ukrytej oraz odpowiednią liczbą wyjść. W poniższym skrypcie utworzono sztuczną sieć neuronową z 201 wejściami, 20 neuronami w warstwie ukrytej oraz jednym wyjściem. Sieć taka posiada zdolność generalizacji przebiegów nieliniowych, okresowych itp.

Schemat sieci wielowarstwowej przedstawia rysunek 5.



Rys. 5. Schemat SSN zawierającej 3 wejścia, warstwę ukrytą (2 neurony) oraz warstwę wyjściową.

Przykładem wykorzystania sieci wielowarstwowej jest generalizacja dowolnie skomplikowanego przebiegu.

```
clear;
clc;

% Definiowanie wektorów 1 elementowych, kolumnowych

P=[-1:0.01:1];
T=5+sin(2*pi*3*P);

% Definiowanie sieci wielowarstwowej uczonej metodą wstecznej propagacji
% błędów

%Uczenie metodą największego spadku, 3 neurony w warstwie ukrytej
LN=3; %Liczba neuronów
net = newff(P,T,LN,{},'traingd');

% Definiowanie parametrów procesu uczenia - 300 epok, dopuszczalny błąd
% 1e-5, wsp. uczenia 0.05

net.trainParam.show = 50;
net.trainParam.lr = 0.05;
net.trainParam.epochs = 300;
net.trainParam.goal = 1e-5;

y = sim(net,P);

blad=T-y;
plot(blad);
```

```
% Po procesie uczenia

net = train(net,P,T); % Trening sieci

y1=sim(net,P);
figure;
plot(T,'b');
hold on;
plot(y1,'r--');
```

Zadania do wykonania:

1. Sprawdzić sprawność uczenia sieci obliczając błąd średniokwadratowy (mse) dla różnej liczby neuronów (LN) w warstwie ukrytej. Przyjąć wartość początkową LN=5.
2. Dla danej liczby neuronów LN obliczyć mse dla dwóch algorytmów minimalizacji błędu wyjściowego sieci – dla metody największego spadku (f. ‘traingd’) z metodą Levenberga-Marquardt’a (f. trainlm’).
3. Opisać wnioski z doświadczenia.