**Politechnika Łódzka** **Instytut Informatyki Stosowanej**

# Podstawy Sztucznej Inteligencji

## *Laboratorium*

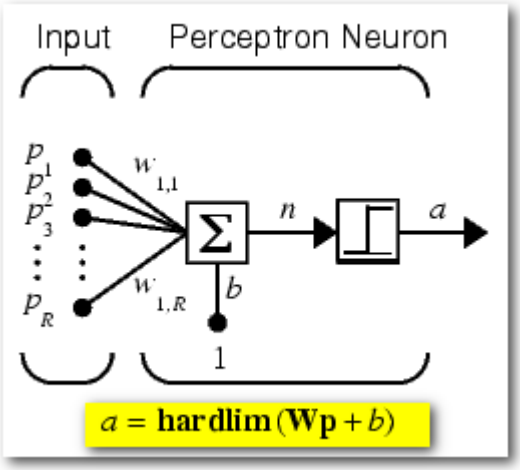
### Ćwiczenie 2

Wykorzystanie środowiska Matlab do modelowania sztucznych sieci neuronowych

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

## *Rozwiązanie zagadnienia klasyfikacji za pomocą pojedynczego perceptronu.*

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].

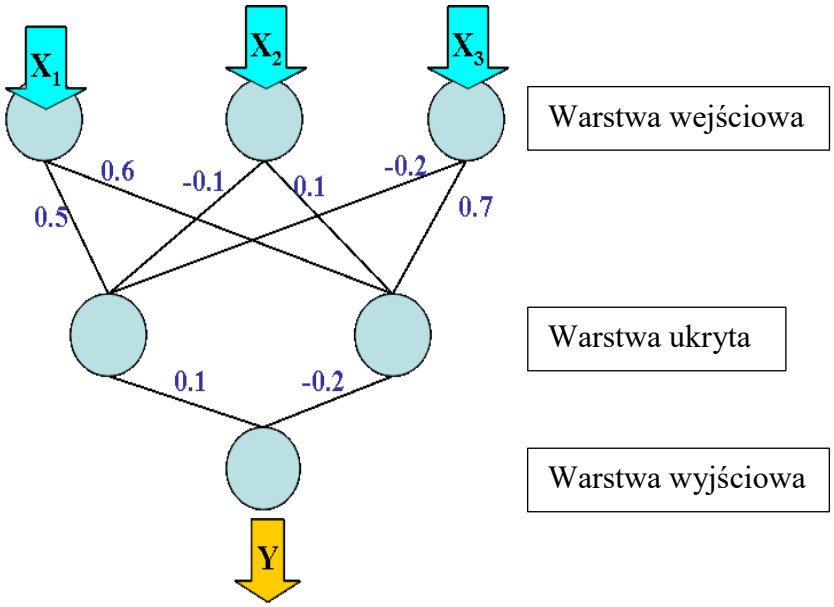
Zatem program wykorzystujący sieć perceptronową do klasyfikacji dwóch zbiorów mógłby mieć następującą postać:

| | |
|---|---|
| P = [ -0.5 -0.5 +0.3 -0.1; ...<br>-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 |

## *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. 2. Schemat SSN zawierającej 3 wejścia, warstwę ukrytą (2 neurony) oraz warstwę wyjściową.

Przykładem wykorzystania sieci wielowarstwowej jest generalizacja dowolnie skomplikowanego przebiegu. Skuteczność odwzorowania uczonego przebiegu zależy od:

1. Struktury SSN
    a. Liczby neuronów w warstwie ukrytej
    b. Liczby wejść SSN (zależy od rodzaju rozwiązywanego zagadnienia)
    c. Liczby wyjść (zależy od rodzaju rozwiązywanego zagadnienia)
2. Metody uczenia SSN, czyli algorytmu doboru wag neuronów we wszystkich warstwach.

```
clear;
clc;

% Definiowanie wektorów 1 elementowych, kolumnowych

[P T]=humps([0:0.05:2]);
% Definiowanie sieci wielowarstwowej uczonej metodą wstecznej propagacji
% błędów

%Uczenie metodą największego spadku, 3 neurony w warstwie ukrytej
LN=10; %Liczba neuronów
net = newff(P,T,LN,{},'trainbr');

% 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 = 1000;
net.trainParam.goal = 1e-5;

y = sim(net,P);

blad=mse(T-y)


% Po procesie uczenia

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

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

blad1=mse(y1-T)
```

**Zadania do wykonania:**

1. Przebadać wpływ liczby neuronów w warstwie ukrytej na sprawność uczenia sieci obliczając błąd średniokwadratowy (**mse**) pomiędzy 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’) oraz regularyzacji bayesowej (f. ‘trainbr’)

3. Wyniki badań przedstawić w następujących tabelach:

Algorytm metody największego spadku (traingd)

| Liczba neuronów w warstwie ukrytej | 3 | 5 | 10 | 20 | 30 | 50 |
|---|---|---|---|---|---|---|
| Średniokwadratowy błąd odwzorowania | | | | | | |

Algorytm metody Levenberga – Marquardt'a (trainlm)

| Liczba neuronów w warstwie ukrytej | 3 | 5 | 10 | 20 | 30 | 50 |
|---|---|---|---|---|---|---|
| Średniokwadratowy błąd odwzorowania | | | | | | |

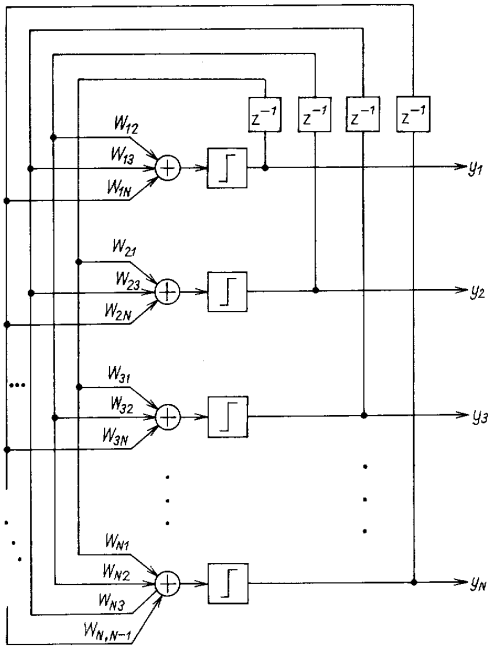
Algorytm regularyzacji [1]Bayesa (trainbr)

| Liczba neuronów w warstwie ukrytej | 3 | 5 | 10 | 20 | 30 | 50 |
|---|---|---|---|---|---|---|
| Średniokwadratowy błąd odwzorowania | | | | | | |

4. Opisać wnioski z doświadczenia.

## *Wykorzystanie sieci Hopfielda do rozwiązania zagadnienia skojarzenia (asocjacji).*

Schemat sieci Hopfielda przedstawia rysunek 3.

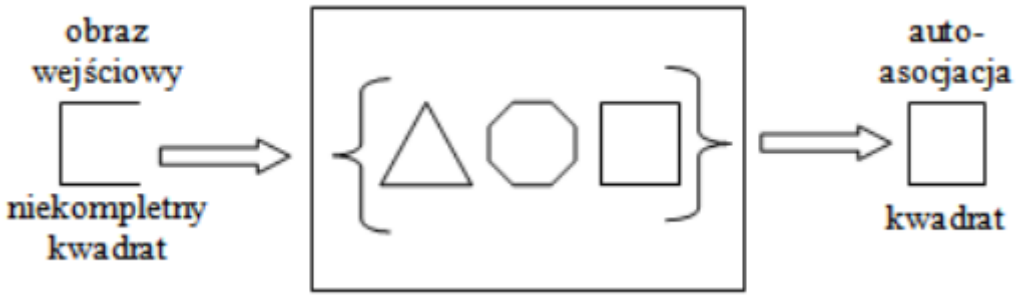



Rys. 3. Sieć Hopfielda ze sprzężeniem zwrotnym.

[1] Regularyzacja – wprowadzenie dodatkowej informacji do rozwiązywanego zagadnienia źle postawionego w celu polepszenia jakości rozwiązania. Regularyzacja jest często wykorzystywana przy rozwiązywaniu problemów odwrotnych.

Sieci tego typu potrafią rozwiązywać zagadnienia asocjacji:


Rys. 4. Przykład klasyfikacji




Rys. 5. Przykład autoasocjacji.

**Przykład 1.**
Rozpoznawanie punktu w przestrzeni.

```
clear
clc
close all

T = [-1 -1 1; 1 -1 1]';
net = newhop(T);
Ai = T;
[Y,Pf,Af] = sim(net,2,[],Ai);
Y

Ai = {[-0.1; -0.8; 0.7]};

[Y,Pf,Af] = sim(net,{1 5},{},Ai);
```

**Przykład 2.**
Rozpoznawanie pojedynczej litery.

```
clear
clc
close all

Lit=[1 1 1 1 1
     1 0 0 0 1
     1 1 1 1 1
     1 0 0 0 1
     1 0 0 0 1];

 T=reshape(Lit,[25 1]);

net = newhop(T);
view(net)
W= net.LW{1,1};
b = net.b{1,1};

Lit1=[1 1 1 1 1
      1 0 0 0 1
      1 0 0 1 1
      1 0 1 0 1
      1 0 0 0 1]

  T1=reshape(Lit1,[25,1]);

  [odpY] = sim(net,1,[],T1);

  odpNN=reshape(odpY,[5 5])

  for i=1:5
     for j=1:5
        if (odpNN(i,j)<1 & odpNN(i,j) >=0.5)
           odpNN(i,j) = 1
        elseif (odpNN(i,j) < 0.5)
           odpNN(i,j) = 0;
        end
     end
  end

  odpNN
```

**Zadanie do wykonania.**
Uzupełnić powyższy kod o możliwość rozpoznania litery B i C.