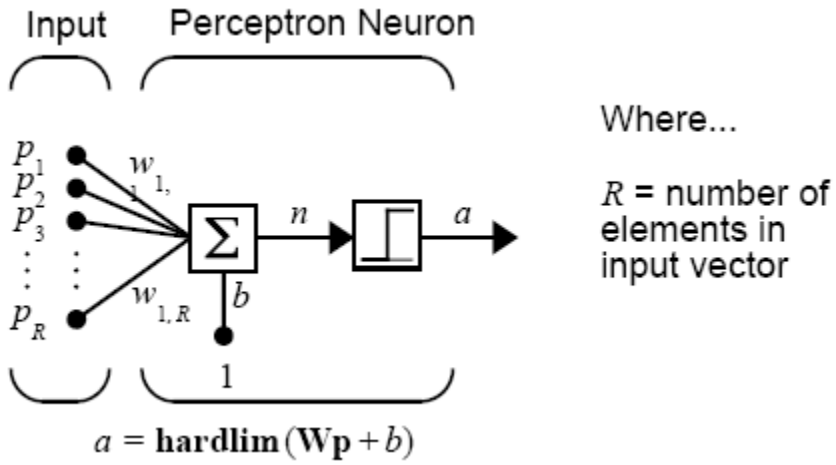
Laborator 5: Reţele de tip perceptron

1 Introducere

Reţelele de tip perceptron sunt utile în special în rezolvarea unor probleme simple de clasificare a formelor. Aceste reţele pot fi create în MATLAB cu funcţia **newp,** iniţializate, simulate şi antrenate cu funcţiile **init**, **sim,** respectiv **train** (sau **adapt**). În continuare, voi descrie modul în care lucrează aceste reţele şi voi introduce aceste funcţii.

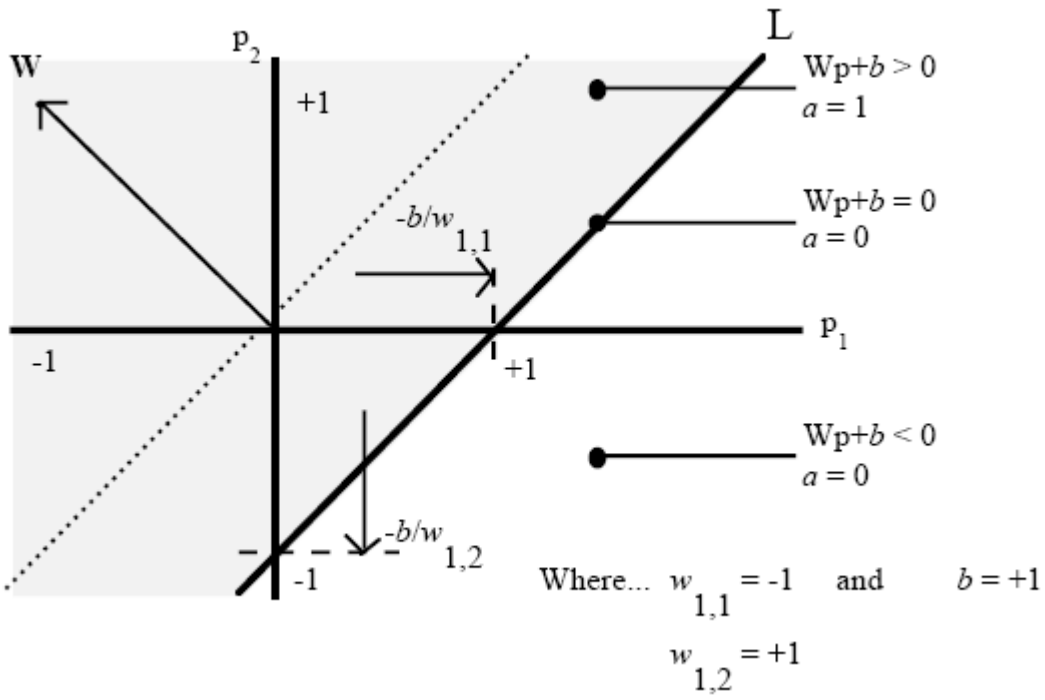
2 Modelul neuronului

Un perceptron care foloseşte funcţia de transfer hardlim este reprezentat mai jos:



Funcţia de transfer a perceptronului, hard-limit, determină clasificarea vectorilor de intrare în două regiuni distincte. Ieşirile reţelei vor fi 0 dacă starea neuronului, *n*, este strict mai mică decât 0 sau 1 dacă este pozitivă.

Spaţiul de intrare al unui neuron cu două intrări, având ponderile $w_{1,1} = -1$, $w_{1,2} = 1$, bias-ul $b = 1$ şi funcţia de transfer hard-limit este reprezentat în imaginea de mai jos:



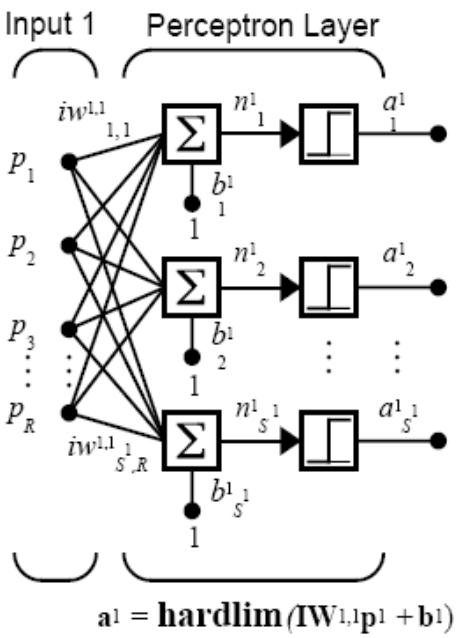
Dreapta de separare L dată de ecuaţia **W*p** + *b* = 0 împarte spaţiul intrărilor în două regiuni (semiplane). Această dreaptă este perpendiculară pe vectorul ponderilor ($w_{1,1}$, $w_{1,2}$) şi are o deplasare dată de bias-ul *b*. Punctele aflate în semiplanul din stânga determinat de linia de separare corespund stării neuronului, **net = W*p** + *b* , mai mari decât 0 şi deci li se asociază ieşirea 1 sau clasa 1, iar punctele, din celălalt semiplan determină **W*p** + *b* < 0 şi prin urmare corespund ieşirii 0 sau clasei 0. Direcţia dreptei de separare şi deplasarea *b* pot fi modificate astfel încât o mulţime de date să fie bine clasificată.

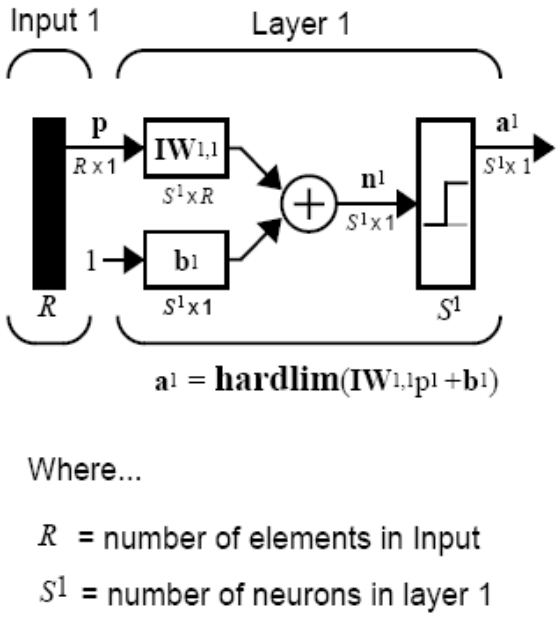
Neuronii având deplasarea *b* = 0 şi funcţia de transfer hard-limit au ca linie de separare o dreaptă ce trece prin origine. Adăugând un bias *b*, neuronul va permite rezolvarea unor probleme de clasificare în care datele de intrare nu sunt

situate în cadrane diferite. Bias-ul permite ca linia de separare să poată fi translatată fată de origine, cum se poate observa din figura de mai sus. Aceste observaţii pot fi vizualizate prin rularea programului demonstrativ **nnd4db**. Observaţi cum se modifică ponderile, respectiv bias-ul, precum şi culoarea marginilor punctelor în momentul în care modificaţi poziţia dreptei de separare.

3 Arhitectura perceptronului

O reţea de perceptroni constă dintr-un singur nivel de S perceptroni, conectaţi cu R input-uri cărora le corespund ponderi de forma $w_{i,j}$ (pondere ce conectează intrarea p_j şi neuronul i) având reprezentarea de mai jos:





Regula de învăţare a perceptronului, pe care o vom descrie este capabilă să antreneze doar o reţea având un singur nivel.

4 Crearea unui perceptron (newp)

Pentru crearea unei reţele de perceptroni se foloseşte funcţia **newp**:

net = newp(PR, S, TF, LF)

unde:

PR este o matrice de dimensiune R × 2 în care sunt precizate minimul şi maximul valorilor datelor de intrare

S reprezintă numărul perceptronilor din reţea

TF (Transfer Function) reprezintă funcţia de transfer din reţea; poate lua două valori '*hardlim'* (funcţia implicită) sau '*hardlims*'

LF (Learning Function) reprezintă funcţia (tipul) de învăţare; poate lua două valori: '*learnp'* (funcţia implicită) sau '*learnpn'*

sau

net = newp(P, T, TF, LF)

unde:

P este o matrice de dimensiune R × Q, care conţine Q vectori de intrare de dimensiune R

T este o matrice de dimensiune S × Q care conţine Q vectori de ieşire de dimensiune S asociaţi vectorilor de intrare din matricea P

Exemplu: Codul următor creează o reţea de tip perceptron cu un singur neuron şi o singură intrare ce ia valori în intervalul [0, 2].

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

Funcţia **newp** generează o structură ce conţine toate informaţiile legate de reţea, cum ar fi: matricea *ponderilor* şi vectorul *bias-urilor.*

Pentru a afla aceşti parametri ai reţelei vom da comenzile:

weights = net.IW{1,1}

bias = net.b{1}

Pentru a vizualiza parametrii rezultaţi în urma creării reţelei se execută următorul cod:

inputweights = net.inputweights{1,1} care va returna

inputweights =

delays: 0
initFcn: 'initzero'
learn: 1
learnFcn: 'learnp'
learnParam: []
size: [1 1]
userdata: [1x1 struct]
weightFcn: 'dotprod'

Trebuie să observăm că funcţia implicită de învăţare este *learnp* pe care o vom discuta puţin mai târziu, iar intrarea reţelei către funcţia de transfer *hardlim* este *dotprod*, funcţie care generează produsul scalar dintre matricea ponderilor şi vectorul de intrare la care adaugă bias-ul pentru a calcula intrarea reţelei. De asemenea, funcţia de iniţializare implicită este *initzero*, funcţie folosită pentru a iniţializa valorile ponderilor.

În mod similar, avem:
biases = net.biases{1} care returnează

biases =

initFcn: 'initzero'
learn: 1
learnFcn: 'learnp'
learnParam: []
size: 1
userdata: [1x1 struct]

Observăm de asemenea că *bias*-ul este iniţializat cu 0.

Exerciţiu: Modificaţi valorile ponderilor şi ale bias-ului (atribuindu-le valori aleatoare în intervalul [0,1]) şi afişaţi valorile acestora pentru reţeaua creată în modul următor:

net = newp([-1 1;-1 1],1);

Verificaţi ce fac comenzile:

net = newp([-1 1;-1 1],1);

net.inputweights{1,1}.initFcn = 'rands';

net.biases{1}.initFcn = 'rands';

5 Simularea unei reţele (sim)

Pentru a arăta modul în care lucrează funcţia *sim*, vom considera următorul exemplu.

Considerăm un perceptron cu un singur vector de intrare bidimensional, având setate ponderile şi bias-ul ca perceptronul prezentat în figura liniei de separare.

Definim reţeaua folosind comanda:

net = newp([-2 2;-2 2],1);

Cum am spus şi mai sus, această comandă setează ponderile şi bias-ul 0; dacă vrem să le modificăm trebuie să le atribuim noi valori. Considerăm ponderile şi bias-ul setate ca la perceptronul prezentat în figura liniei de separare:

net.IW{1,1}= [-1 1];

net.b{1} = [1];

Pentru a verifica corectitudinea setărilor folosim comenzile:

net.IW{1,1}

ans =

-1 1

net.b{1}

ans =

1

Simulăm comportamentul reţelei pentru două date de intrare p1 = [1;1] şi p2 = [1;-1] (puncte aflate în semiplane diferite); ne propunem să aflăm din ce clase fac parte cele două puncte.

```
p1 = [1;1];
a1 = sim(net,p1)

a1 =
     1
```

şi

```
p2 = [1;-1];
a2 = sim(net,p2)
a2 =
     0
```

Se observă că reţeaua clasifică corect cele două date de intrare (ieşirile sunt 0 şi 1 corespunzătoare funcţiei hardlim).

Exerciţiu: Reprezentaţi grafic linia de separare determinată de parametrii din figură şi adăugaţi 4 puncte figurii (pe care să le reţineţi într-o matrice pe coloane) atribuindu-le simboluri potrivite în funcţie de clasa din care fac parte şi returnaţi ieşirile reţelei corespunzătoare punctelor care indică această clasă.

Putem introduce datele de intrare şi printr-o matrice de celule, iar rezultatul va fi returnat tot într-o matrice de celule:

```
p3 = {[1;1] [1;-1]};
a3 = sim(net,p3)
a3 =
     [1]  [0]
```

6 Iniţializarea unui perceptron (init)

Folosim funcţia *init* pentru a reseta ponderile şi bias-urile asociate reţelei la valorile iniţiale (valorile cu care s-a iniţializat reţeaua). Presupunem că am definit reţeaua:

net = newp([-2 2;-2 2],1);

Ponderile reţelei le putem vizualiza folosind comenzile:

wts = net.IW{1,1}

care returnează valorile aşteptate

wts =

0 0

În acelaşi mod, se poate observa că bias-ul este iniţial 0, utilizând comanda:

bias = net.b{1}

bias =

0

Setăm ponderilor valorile 3 şi respectiv 4 şi bias-ului valoarea 5.

net.IW{1,1} = [3,4];

net.b{1} = 5;

Pentru a reiniţializa ponderile (atribuim ponderilor valorile iniţiale) folosim funcţia *init* cu sintaxa:

net = init(net);

Putem modifica modul în care un perceptron este iniţializat folosind funcţia *init*. De exemplu, putem redefini ponderile şi bias-ul reţelei astfel:

net.inputweights{1,1}.initFcn = 'rands';

net.biases{1}.initFcn = 'rands';

net = init(net);

7 Regula de învăţare a perceptronului

Definim o regulă de învăţare ca fiind o metodă de modificare a ponderilor şi a bias-ului reţelei cu scopul de a antrena reţeaua în vederea realizării unei sarcini.

Considerăm ca fiind dată o mulţime de antrenare $\{(p_1,t_1),(p_2,t_2),...,(p_m,t_m)\}$, unde p_i reprezintă intrarea reţelei, iar t_i valoarea dorită corespunzătoare intrării p_i. Obiectivul învăţării constă în minimizarea erorii date prin ***e = t – a,*** unde ***t*** este valoarea target corespunzătoare unei intrări ***p***, iar ***a*** reprezintă ieşirea reţelei.

Regula implicită de învăţare a perceptronului - **learnp** – este o regulă de învăţare supervizată ce calculează noile ponderi, respectiv bias-ul pentru un vector de intrare ***p*** dat şi pentru eroarea calculată ***e***. Regula de învăţare a perceptronului constă de fapt în găsirea unei drepte de separare care să clasifice corect mulţimea vectorilor de intrare.

La fiecare nouă aplicare a regulii **learnp**, probabilitatea ca perceptronul să returneze o ieşire corectă creşte. S-a demonstrat că regula perceptronului converge la o soluţie într-un număr finit de iteraţii, dacă există o soluţie (adică dacă mulţimea este liniar separabilă).

Regula de învăţare a perceptronului (de actualizare a ponderilor şi a bias-ului) este dată prin relaţiile:

$$\mathrm{W}^{\mathrm{new}} = \mathrm{W}^{\mathrm{old}} + e\mathrm{p}^{\mathrm{T}}$$

$$\mathrm{b}^{\mathrm{new}} = \mathrm{b}^{\mathrm{old}} + e$$

Regula de învăţare **learnpn** a perceptronului calculează ponderile actualizate în acelaşi mod ca funcţia **learnp**, dar pentru vectorii de intrare normaţi, adică vectorii de intrare vor fi de forma:

$$p \mapsto \frac{p}{\|p\|}.$$

Folosirea acestei reguli este indicată în cazul în care mulţimea de antrenare conţine date având valori discordante în raport cu celelalte exemple, în sensul că norma acestora are o valoare extremă mare/mică faţă de celelalte date. Această

discordanţă înre date poate determina un proces de antrenare de o complexitate mult mai mare, întrucât în momentul introducerii vectorului având o valoare discordantă ponderile vor fi actualizate prin adăugarea acestui vector în cazul în care este misclasificat, ceea ce va implica un număr mai mare de parcurgeri ale mulţimii de antrenare astfel încât să se manifeste şi influenţa celorlalte date în modificarea ponderilor.

Exemplu:

Fie o reţea cu un singur perceptron, având ca intrare un vector cu două elemente. Considerăm reţeaua creată astfel:

net = newp([-2 2;-2 2],1);

Considerăm parametrii reţelei cu următoarele setări:

net.b{1} = [0];

w = [1 -0.8];

net.IW{1,1} = w;

Perechea input-target (p, t) este dată de:

p = [1; 2];

t = [1];

Putem calcula ieşirea reţelei şi eroarea folosind comenzile:

a = sim(net,p)

a =

0

e = t-a

e =

1

şi în final aplicăm funcţia **learnp** pentru a actualiza ponderile (se returnează diferenţa dintre ponderea modificată şi ponderea iniţială, notată prin dw)

dw = learnp(w, p, [], [], [], [], e, [], [], [])

dw =

1 2

Noile ponderi sunt obţinute astfel:

w = w + dw

w =

2.0000 1.2000

Procesul de determinare a ponderilor (a bias-urilor) poate fi repetat până când eroarea devine 0, deoarece se ştie că algoritmul de determinare a ponderilor este convergent într-un număr finit de paşi pentru toate problemele care pot fi rezolvate folosind un perceptron, adică pentru problemele în care mulţimea datelor de intrare este liniar separabilă. Datele pe care vrem să le clasificăm vor fi în acest caz separate de o dreaptă.

Exerciţiu: Studiaţi programul demonstrativ **nnd4pr**, care vă permite să adăugaţi noi date de intrare şi să observaţi cum se modifică ponderile în momentul în care aplicăm regula de învăţare cu scopul de a clasifica corect datele.

8 Antrenarea perceptronului (train)

În cazul în care problema este liniar separabilă, aplicând de un număr finit de ori în mod repetat funcţiile **learnp** şi **sim** perceptronul poate determina dreapta de separare. Fiecare parcurgere a mulţimii de antrenare se numeşte *epocă* sau *ciclu*. Funcţia **train** efectuează un astfel de ciclu. La fiecare pas, pentru o secvenţă de input-uri, funcţia **train** calculează ieşirea şi eroarea şi modifică ponderile şi bias-ul dacă e nevoie pentru fiecare dintre datele din secvenţa de intrare.

Trebuie reţinut faptul că aplicând funcţia **train** pentru o singură epocă nu se garantează faptul că reţeaua rezultată îndeplineşte funcţia ţintă. Valorile **W** şi **b** pot fi verificate prin calcularea ieşirilor reţelei pentru fiecare vector de intrare pentru a vedea dacă valorile target sunt atinse.

Exemplu: Fie mulţimea de antrenare având următoarele date:

$$\left\{p_1=\begin{bmatrix}2\\2\end{bmatrix},t_1=0\right\},\ \left\{p_2=\begin{bmatrix}1\\-2\end{bmatrix},t_2=1\right\},\ \left\{p_3=\begin{bmatrix}-2\\2\end{bmatrix},t_3=0\right\},\ \left\{p_4=\begin{bmatrix}-1\\1\end{bmatrix},t_4=1\right\}$$

Dorim să clasificăm corect aceste date folosind o reţea de tip perceptron cu un neuron şi având ca date de intrare vectori cu două componente. Iniţializăm ponderile şi bias-ul cu 0. Astfel, vom avea W(0) = [0 0], b(0) = 0 (indicăm pasul algoritmului prin indicii din paranteze asociaţi ponderilor şi bias-ului).

La primul pas al învăţării calculăm ieşirea asociată intrării p_1 folosind ponderile iniţiale astfel:

$$a=\text{hardlim}(\text{W}(0)p_1+b(0))$$
$$=\text{hardlim}([0\ 0]\begin{bmatrix}2\\2\end{bmatrix}+0)=\text{hardlim}(0)=1$$

Ieşirea *a* nu este egală cu ieşirea target 1, deci vom aplica regula de învăţare a perceptronului pentru a modifica în manieră incrementală ponderile şi bias-ul bazându-ne pe eroarea *e.* Obţinem astfel:

$$e=t_1-a=0-1=-1$$
$$\Delta\text{W}=ep_1^T=(-1)[2\ 2]=[-2\ -2]$$
$$\Delta b=e=(-1)=-1$$

Putem calcula noile ponderi şi bias-ul folosind regula perceptronului în modul următor:

$$\text{W}^{\text{new}}=\text{W}^{\text{old}}+ep^T=[0\ 0]+[-2\ -2]=[-2\ -2]=\text{W}(1)$$
$$b^{new}=b^{old}+e=0+(-1)=-1=b(1)$$

Introducem acum cel de al doilea vector de intrare p_2 şi obţinem ieşirea:

$$a=\text{hardlim}(\text{W}(1)p_2+b(1))$$
$$=\text{hardlim}([-2\ -2]\begin{bmatrix}1\\-2\end{bmatrix}-1)=\text{hardlim}(1)=1$$
$$e=t_2-a=1-1=0$$
$$\text{W}(2)=\text{W}(1)=[-2\ -2]$$
$$b(2)=b(1)=-1$$

$$e = t_2 - a = 1 - 1 = 0$$

Deoarece atât ieşirea target, cât şi ieşirea reţelei sunt 1 rezultă că eroarea este zero, deci ponderile nu se modifică şi avem $W(2) = W(1) = [-2 \ -2]$ şi $b(2) = b(1) = -1$

După primii 4 paşi, adică după prima epocă, obţinem $W(4) = [-3 \ -1]$ şi $b(4) = 0$. Pentru a determina dacă am obţinut o soluţie trebuie să parcurgem mulţimea de antrenare şi să verificăm dacă ieşirea target a fiecărui punct corespunde ieşirii reţelei determinate de ponderile şi bias-ul obţinute după o epocă. Întrucât nu toate punctele sunt bine clasificate (punctul p_2 din mulţimea de antrenare este misclasat) algoritmul continuă până când obţinem $W(6) = [-2 \ -3]$ şi $b(6) = 1$.

Putem descrie algoritmul prezentat mai sus şi folosind funcţia **train**.

Fie reţeaua de tip perceptron definită ca mai jos. Vrem să antrenăm reţeaua folosind mulţimea de antrenare de mai sus într-o singură epocă şi să observăm că se obţin aceleaşi ponderi ca în algoritmul prezentat mai devreme.

net = newp([-2 2;-2 +2],1);

net.trainParam.epochs = 1;

Vectorii de intrare şi valorile target corespunzătoare sunt:

p = [[2;2] [1;-2] [-2;2] [-1;1]]

t = [0 1 0 1]

Acum antrenăm reţeaua utilizând funcţia **train**:

net = train(net,p,t);

După această primă epocă vom obţine W = [-3 -1] şi *b* = 0.

Pentru a verifica dacă ponderile şi bias-ul rezolvă problema de clasificare simulăm reţeaua astfel:

a = sim(net,p)

a =

[0] [0] [1] [1]

e = t – a

Întrucât vectorul de eroare nu are toate componentele nule, rezultă că nu am obţinut parametrii care definesc linia de separare, deci trebuie să continuăm

antrenarea reţelei pentru un număr mai mare de epoci. Pentru un număr de epoci egal cu 2 vom obţine vectorul de eroare *e* având toate componentele nule, deci obţinem parametrii corecţi care clasifică corect datele din mulţimea de antrenare.

Exerciţiu: Antrenaţi reţeaua de mai sus timp de o epocă după care reprezentaţi grafic punctele, respectiv dreapta de separare. Repetaţi antrenarea reţelei timp de patru epoci şi reprezentaţi grafic punctele, respectiv dreapta de separare obţinută. (Realizaţi ambele reprezentări grafice în aceeaşi fereastră). Interpretaţi rezultatele obţinute!

Exerciţiu: Rulaţi programul demonstrativ demop1 (program care clasifică o mulţime de antrenare şi ilustrează modul în care reţeaua este antrenată).

Funcţia **train** realizează o antrenare **serială** (de tip incremental) în sensul că după introducerea fiecărei date din mulţimea de antrenare se realizează ajustarea parametrilor, dar matricea ponderilor net.IW{1,1} şi vectorul bias net.b{1} va conţine valorile actualizate după parcurgerea întregii mulţimi de antrenare.

Voi introduce o nouă funcţie utilizată în antrenarea perceptronului, funcţia **adapt**, care însă realizează o antrenare de tip **batch** în care ponderile net.IW{1,1} şi bias-ul net.b{1} se modifică pe baza tuturor input-urilor. Ponderile şi bias-ul se modifică prin adăugarea la vechile valori suma punctelor misclasificate. În cazul funcţiei **adapt** trebuie să setăm numărul de parcurgeri ale mulţimii de antrenare, necesar antrenării reţelei. Funcţia **adapt** poate realiza şi o antrenare de tip incremental dacă este aplicată unei singure date de intrare. Astfel pentru antrenarea reţelei prezentate mai sus folosind o antrenare de tip **batch** vom utiliza următorul cod:

```
net = newp([-2 2; -2 2], 1);
p = [2 1 -1 -1; 2 -2 2 1]  % datele de intrare
t = [0 1 0 1]  % valorile target
net.adaptParam.passes = 100;
net = adapt(net, p, t);
plotpv(p,t); % reprezintă grafic punctele
hold on
```

plotpc(net.IW{1,1}, net.b{1}); % reprezintă grafic dreapta separatoare

Exerciţiu: Scrieţi un program MATLAB care să definească o reţea de tip perceptron utilizată pentru reprezentarea funcţiilor logice AND şi OR pe care să o antrenaţi pentru a recunoaşte aceste funcţii şi reprezentaţi grafic punctele, respectiv dreapta de separare.

9 Aplicaţie:

Se dă următoarea mulţime de antrenare formată din perechile de puncte, respectiv etichetele corespunzătoare:

$$\left\{\left(\begin{bmatrix}0\\0\end{bmatrix},1\right),\left(\begin{bmatrix}0\\0.5\end{bmatrix},1\right),\left(\begin{bmatrix}0\\1\end{bmatrix},1\right),\left(\begin{bmatrix}0.5\\0\end{bmatrix},1\right),\left(\begin{bmatrix}0.5\\0.5\end{bmatrix},-1\right),\left(\begin{bmatrix}0.5\\1\end{bmatrix},-1\right),\left(\begin{bmatrix}1\\0\end{bmatrix},-1\right),\left(\begin{bmatrix}1\\0.5\end{bmatrix},-1\right)\right\}$$

Implementaţi în MATLAB varianta **incrementală** a algoritmului de antrenare a perceptronului (fără a utiliza funcţiile MATLAB **adapt** sau **train**).

a. Repetaţi algoritmul de antrenare de un număr maxim de epoci de 500. În momentul determinării ponderilor care dau direcţia dreptei de separare, care clasifică corect punctele din mulţimea de antrenare, încheiaţi procesul de antrenare şi plotaţi punctele din mulţimea de antrenare şi dreapta de separare .

b. Afişaţi dacă mulţimea de antrenare este liniar separabilă sau neliniar separabilă folosind funcţia **disp** pe baza antrenării într-un număr maxim de epoci de 500. Dacă mulţimea de antrenare nu este liniar separabilă determinaţi dreapta care misclasifică cele mai puţine puncte şi afişaţi punctele misclasificate.

c. Studiaţi convergenţa algoritmului perceptronului, reprezentând grafic iteraţia versus valoarea cosinusului dintre doi vectori de ponderi obţinuţi la iteraţii consecutive.