Modalități de instruire a rețelelor neuronale

Procesul de atribuire de valori ponderilor sinapselor unei rețele neuronale se numeşte *proces de învățare* sau *proces de instruire* sau *proces de antrenare*.

Există două tipuri fundamentale de învățare în rețelele neuronale: *învățarea supervizată* şi *învățarea nesupervizată*.

În cazul învățării supervizate, se prezintă rețelei o mulțime de exemple de învățare, reprezentând perechi vectoriale intrare-ieşire caracteristice aplicației. Vectorii de intrare sunt introduşi în calcule succesiv, împreună cu ponderile curente ale rețelei, ținând seamă de structura acesteia şi de funcțiile de răspuns ale neuronilor ei, rezultând, corespunzător, un set de vectori de ieşire.

Se compară, respectiv, vectorii de ieşire rezultați cu vectorii de ieşire indicați în cadrul exemplelor de învățare şi, în funcție de abateri, se ajustează ponderile astfel încât cei doi termeni ai comparației să devină identici sau cât mai apropiați. Tot acest proces se repetă de un număr de ori, iar la sfârşitul lui, se poate considera că rețeaua este *instruită*. Evident, se mizează pe faptul că, impunând rețelei un anumit comportament într-un număr de situații cunoscute, ea se va comporta corespunzător –altfel spus: ”*în acelaşi spirit*”- şi în situații noi.

În cazul învățării nesupervizate, mulțimea de învățare constă doar dintr-un număr de vectori de intrare. Aceştia sunt introduşi în calcule ca şi în cazul învățării supervizate. De această dată, însă, finalitatea calculelor este ajustarea ponderilor astfel încât pentru vectori de intrare apropiați să se obțină unul şi acelaşi vector de ieşire sau, eventual, vectori de ieşire apropiați.

Formula generală de ajustare a ponderilor unui neuron oarecare, "*j*", în cadrul unui pas, "*k*", al procesului de învățare, este [Amari 1990]:

$$\tilde{\mathbf{w}}^{\mathbf{j}}(k) = \tilde{\mathbf{w}}^{\mathbf{j}}(k-1) + cr(\tilde{\mathbf{w}}^{\mathbf{j}}(k-1), \tilde{\mathbf{i}}^{\mathbf{jh}}, d^{jh})\tilde{\mathbf{i}}^{\mathbf{jh}}$$

în cazul învățării supervizate, respectiv:

$$\tilde{\mathbf{w}}^{\mathbf{j}}(k) = \tilde{\mathbf{w}}^{\mathbf{j}}(k-1) + cr(\tilde{\mathbf{w}}^{\mathbf{j}}(k-1), \tilde{\mathbf{i}}^{\mathbf{jh}})\tilde{\mathbf{i}}^{\mathbf{jh}}$$

în cazul învățării nesupervizate, unde:

- r reprezintă aşa-numitul "*semnal de învățare*"
- c este "*rata de învățare*"
- $\tilde{\mathbf{w}}^{\mathbf{j}}(k)$, $\tilde{\mathbf{w}}^{\mathbf{j}}(k-1)$ sunt vectorii ponderilor extinşi ai neuronului "*j*", la paşii "*k*", respectiv "*k–1*"
- $\tilde{\mathbf{i}}^{\mathbf{jh}}$ este vectorul de intrare extins, aplicat curent la bornele neuronului "*j*"
- d^{jh} este valoarea dorită la ieşirea neuronului "*j*", când la intrarea sa este prezent vectorul $\tilde{\mathbf{i}}^{\mathbf{jh}}$

Se poate remarca faptul că, atât în cazul învățării supervizate, cât şi în cazul învățării nesupervizate, cantitatea cu care se ajustează vectorul ponderilor în cadrul unui pas al învățării este proporțională cu produsul dintre semnalul de învățare şi vectorul de intrare extins.

După cum se va vedea pe parcursul lucrării, expresia semnalului de învățare se instanțiază într-un fel sau altul, corespunzător metodei de învățare sau, cum se obişnuieşte mai degrabă a se spune, "regulii de învățare" avută în vedere la un moment dat.

Rezolvarea în MATLAB a unor probleme de clasificare cu ajutorul perceptronului

În *Figura 1* este reprezentat un perceptron cu:

- *R* intrări - $p_{1,} p_{2,...,} p_R$
- 1 ieşire – *a*
- $w_{1,1},...w_{1,R}$ ponderile asociate intrărilor
- *b* – bias-ul (translatarea sau pragul)
- *n*- excitația neuronului

Perceptronul considerat are ca funcție de răspuns la excitație *funcția prag unipolară discretă,* implementată în *MATHLAB* printr-o funcție numită *hardlim (Hard Limit)*:

$$hardlim(n) = \begin{cases} 0, daca\ n < 0 \\ 1,\ daca\ n >= 0 \end{cases}$$

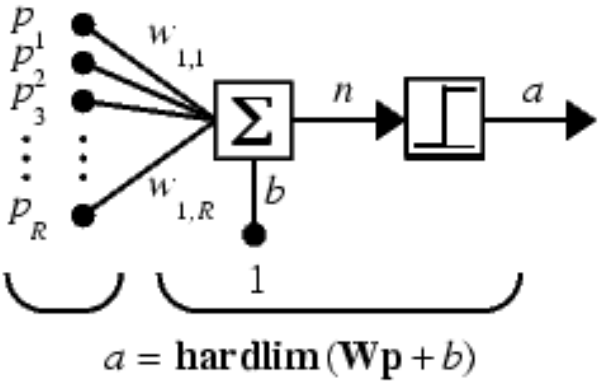

Figura 1. Reprezentarea unui perceptron cu *R* intrări, având ca funcție de răspuns la excitație *funcția prag unipolară discretă (hardlim)*

În *Figura 1*, $a = hardlim(n) = hardlim(W \cdot p + b)$, unde *W* reprezintă vectorul ponderilor transpus: $W = w^T = [w_{1,1}\ w_{1,2}\ ...w_{1,R}]$, iar *p* reprezintă vectorul intrărilor $p = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_R \end{bmatrix}$.

Funcția *hardlim* oferă perceptronului abilitatea de a clasifica vectorii de intrare în două regiuni complementare, separabile printr-un hiperplan de decizie, reprezentat aici prin dreapta $W \cdot p + b = 0$.

Se poate, atunci, spune că vectorii de intrare aflați în partea pozitivă a hiperplanului de decizie corespund unei clase –să o notăm cu *A1*-, iar vectorii de intrare aflați în partea negativă a hiperplanului de decizie corespund altei clase –să o notăm cu *A2*-. Valoarea +1 la un moment dat a ieşirii *a* a perceptronului va fi un indiciu că vectorul de intrare din acel moment aparține clasei *A1*, în timp ce valoarea 0 a ieşirii *a* corespunde faptului că vectorul de intrare curent aparține clasei *A2*.

Crearea unui perceptron în MATLAB

Un perceptron se creează cu ajutorul funcției *perceptron*:

$$net = perceptron(P, T, TF, LF)$$

- *TF* – funcția de transfer (funcția de răspuns la excitație, de activare). Implicit, această funcție este *hardlim*
- *LF* – funcția care realizează instruirea rețelei. Implicit această funcție este *learnp*, care implementează algoritmul perceptronului.

Funcția *perceptron* returnează un nou perceptron.
Funcția de transfer *TF* poate fi HARDLIM sau HARDLIMS.
Funcția de învățare *LF* poate fi LEARNP sau LEARNPN.

Configurarea unei retele (implicit şi a unui perceptron) în MATLAB

```
net = configure(net,x,t)
net = configure(net,x)
net = configure(net,'inputs',x,i)
net = configure(net,'outputs',t,i)
```

Configurarea unei retele neuronale este procesul de setare a dimensiunilor si plajelor de variatie ale intrarilor si iesirilor retelei si de initializare a ponderilor, astfel incat sa se asigure o potrivire cat mai buna intre setul de date de intrare si setul target de date de iesire.

Configurarea retelei trebuie sa aiba loc inainte ca ponderile si bias-urile retelei sa fie initializate. Retelele neconfigurate sunt automat configurate si initializate prima data cand se apeleaza functia *train*. Ca si alternativa, o retea poate fi configurata manual, fie prin apelul acestei functii, fie prin setarea manuala a dimensiunilor si a plajelor de variatie ale intrarilor si iesirilor, precum si a informatiilor de procesare si de initializare.

net = configure(net,x,t) preia setul de date de intrare *x* si setul target de date de iesire, *t,* si configureaza intrarile si iesirile retelei pentru a asigura potrivirea.

net = configure(net,x) configureaza doar intrarile.

net = configure(net,'inputs',x,i) configureaza intrarile specificate prin vectorul index *i*. Daca *i* nu este specificat, toate intrarile vor fi configurate.

net = configure(net,'outputs',t,i) configureaza iesirile specificate prin vectorul index *i*. Daca *i* nu este specificat, toate iesirile target vor fi configurate.

Antrenarea unei rețele neuronale (implicit şi a unui perceptron) în MATLAB

Pentru antrenarea unei rețele în Matlab putem folosi funcția *train*.
Exemplu:

```
net.trainParam.epochs = 50;
net.trainParam.goal = 0.01;
net = train(net,p,t);
```

În mod obişnuit, o epocă de antrenare este definită ca fiind procesarea de către rețea a tuturor exemplelor din setul de antrenament. În cazul metodei *train*, valorile bias-ului si ale ponderilor sunt modificate după procesarea tuturor exemplelor de învățare.

Antrenarea rețelei se consideră finalizată, fie cand s-a atins numărul maxim de epoci (setat prin `net.trainParam.epochs`), fie când s-a ajuns la o valoare a erorii permisă (setată prin `net.trainParam.goal`), fie când se îndeplinesc diferite condiții de oprire a antrenării specificate în funcția NET.trainFcn.

Simularea unei rețele neuronale (implicit şi a unui perceptron) în MATLAB

Exemplu:

```
a = net(p1);
a = net(p);
```

Funcția *net* calculează ieşirea rețelei *net*, pentru vectorul de intrare p_1, sau ieşirile rețelei pentru toți vectorii de intrare din setul de antrenament, *p*.

Setarea ponderilor şi a bias-ului

Funcția *init* resetează ponderile şi bias-ul rețelei la valorile implicite.

Exemplu:

```
net = init(net);
```

Pentru setarea valorilor ponderilor si bias-lui, putem folosi urmatoarele linii de cod:

```
net.IW{1} = [3,4];
net.b{1} = 5;
```

Pentru vizualizarea valorilor ponderilor si bias-lui, putem folosi urmatoarele linii de cod:

```
net.IW{1}
net.b{1}
```

Probleme de clasificare

Exemplul 1

Se consideră un perceptron cu 2 intrări, având ca funcție de activare funcția prag unipolară discretă (*hardlim*). Ne propunem să antrenăm acest perceptron astfel încât să clasifice 4 vectori de intrare în două categorii.

Fie:

```
% matricea cu intrari P
P = [-0.5 -0.5 +0.3 -0.1;   -0.5 +0.5 -0.5 +1.0];

% iesirile target
T = [1 1 0 0];
```

P – conține cei 4 vectori de intrare cu 2 elemente fiecare.
T – reprezintă vectorul *target* de ieşire a rețelei, care are valori care definesc cele două categorii.

Funcția *plotv* trasează punctele aferente vectorilor de intrare ce urmează a fi clasificați precum si vectorul target care reprezintă categoriile din care aceştia fac parte. Vectorii de intrare care au asociata ieşirea dorită 1 sunt reprezentați prin simbolul “+”, iar vectorii de intrare care au asociata ieşirea dorită 0 sunt reprezentați prin simbolul “o”.

```
plotpv(P,T);
```

Rezultatul rulării acestei funcții este prezentat în Figura 2.

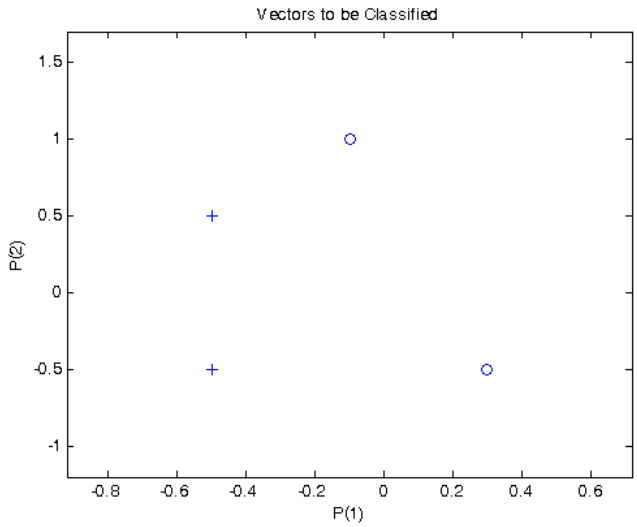

Figura 2. Reprezentarea vectorilor de intrare ce urmează a fi clasificați
şi a vectorului *target* de ieşire

Perceptronul considerat va trebui să clasifice cei 4 vectori de intrare prezentați în *P* în două clase complementare, definite de vectorul *target T*. Perceptronul are ca şi funcție de ieşire funcția *hardlim*, astfel e capabil să separe un spațiu de intrare printr-o dreaptă în două categorii (0 şi 1).

```
%creare retea neuronala de tip perceptron
net = perceptron;
net = configure(net,P,T);
```

Ponderile inițiale sunt setate implicit la 0.

```
%afisarea ponderilor si a bias-ului inainte de antrenare
net.IW{1}
net.b{1}
```

În continuare, ne propunem să antrenăm perceptronul şi ulterior, să reprezentăm grafic (Figura 3) cei 4 vectori de intrare (`plotpv(P,T)`) şi dreapta care realizează împarțirea acestora în două clase complementare (`plotpc(net.IW{1},net.b{1}))`.

```
%stabilirea numarului de iteratii
net.trainParam.epochs = 50;
%stabilirea erorii permise
net.trainParam.goal = 0.01;
%antrenarea retelei
net=train(net,P,T);
%afisarea ponderilor si a bias-ului dupa antrenare
net.IW{1}
net.b{1}
%reprezentarea grafica a celor 4 puncte
plotpv(P,T);
hold on;
%reprezentarea dreptei de decizie care separa regiunile de clasificare
plotpc(net.IW{1}, net.b{1});
hold off;
```

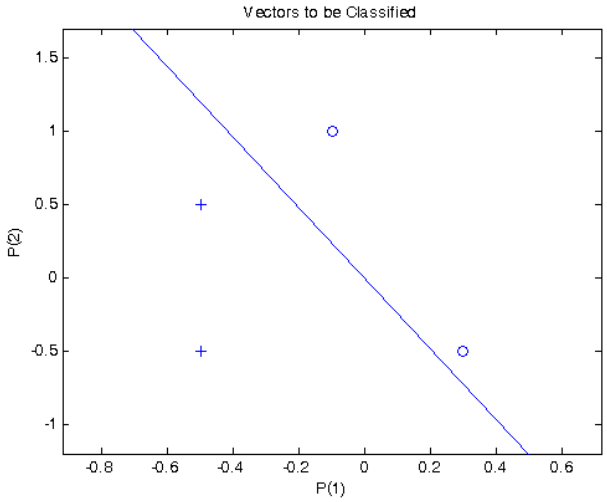

Figura 3. Reprezentarea vectorilor de intrare ce au fost clasificați,
a vectorului *target* de ieşire şi a dreptei care realizează împarțirea spațiului
de intrare în două clase complementare

Se consideră un nou vector de intrare pe care dorim să îl clasificăm, p . Pentru a-l distinge de vectorii de intrare prezentați în setul de antrenament, îl vom colora cu roşu (Figura 4).

Cu ajutorul funcției *net* vom calcula ieşirea rețelei pentru noua intrare considerată.

```
%testarea retelei
a=net(P)
%testarea retelei pentru alt set de intrari
x = [0.7; 1.2];
a = net(x)
plotpv(x,a);
point = findobj(gca,'type','line');
set(point,'Color','red');
```

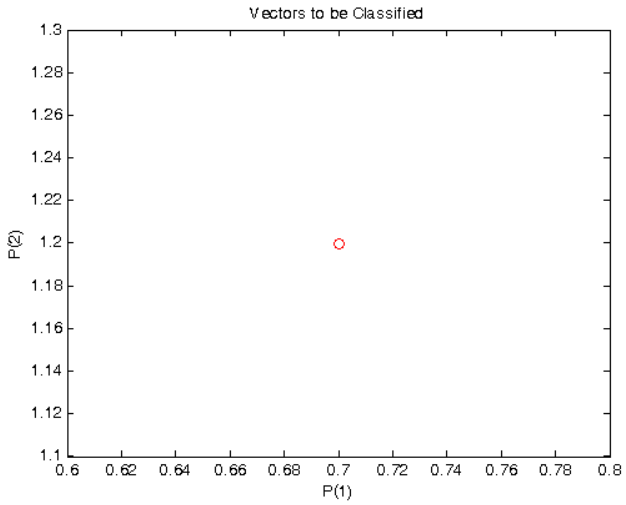

Figura 4. Reprezentarea noului vector de intrare clasificat cu rețeaua antrenată

Perceptronul a clasificat corect noul punct, prin simbolul „o”, ca facând parte din categoria *zero*, şi nu din categoria *unu* reprezentată prin simbolul “+”.

În Figura 5, am reprezentat dreapta care realizează clasificarea şi vectorii de intrare clasificați (atât cei din setul de antrenament, cât şi vectorul nou considerat).

```
hold on;
plotpv(P,T);
plotpc(net.IW{1},net.b{1});
hold off;
```

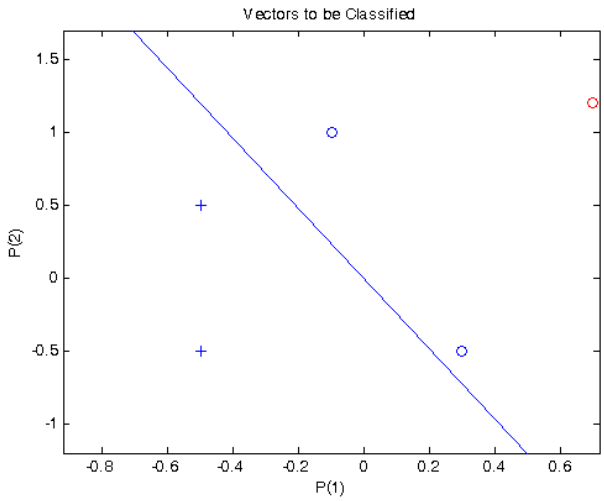

Figura 5. Reprezentarea vectorilor de intrare clasificați şi a dreptei de clasificare

Exemplul 2

Se consideră un perceptron cu 1 intrare, având ca funcție de activare funcția prag unipolară discretă (*hardlim*). Ne propunem să antrenăm acest perceptron astfel încât el să poată clasifica numerele ce-i vor fi prezentate la intrare în numere pozitive, respectiv în numere negative.

```
P=[1 -0.5 3 -0.1 0.1 -2]
T=[1  0   1   0   1   0]
```

Cu ajutorul funcției *plotpv* vom reprezenta cele 6 valori ale intrării perceptronului din setul de antrenament, precum şi valorile target asociate. Observăm că numerele negative sunt reprezentate de simbolul “o”, iar numerele pozitive de simbolul “+”.

```
plotpv(P,T);
```

Rezultatul rulării acestei funcții este prezentat în Figura 6.

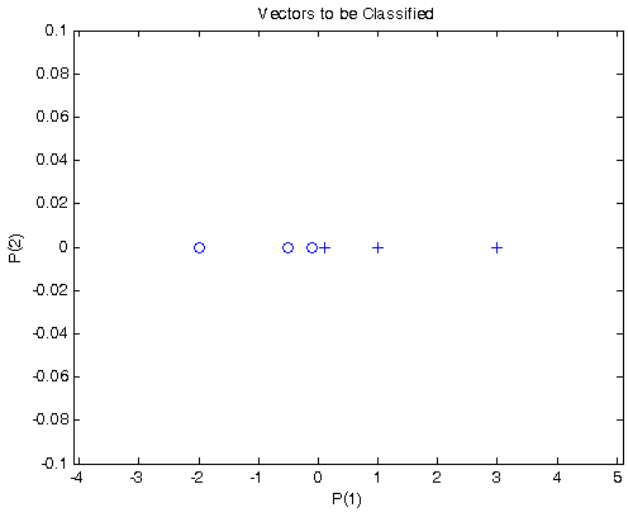

Figura 6. Reprezentarea vectorilor de intrare ce urmează a fi clasificaţi şi a vectorului *target* de ieşire

În cele ce urmează vom crea un perceptron şi il vom antrena cu ajutorul metodei *TRAIN*.

```
%creare retea neuronala de tip perceptron
net = perceptron;
net = configure(net,P,T);
%afisarea ponderilor si a bias-ului inainte de antrenare
net.IW{1}
net.b{1}

%stabilirea numarului de iteratii
net.trainParam.epochs = 50;
%stabilirea erorii permise
net.trainParam.goal = 0.01;
%antrenarea retelei
net=train(net,P,T);
%afisarea ponderilor si a bias-ului dupa antrenare
net.IW{1}
net.b{1}
%reprezentarea grafica a celor 4 puncte
plotpv(P,T);
hold on;
%reprezentarea dreptei de decizie care separa regiunile de clasificare
plotpc(net.IW{1}, net.b{1});
hold off;
```

În Figura 7, după ce antrenarea s-a finalizat, am reprezentat grafic cei 6 vectori de intrare (`plotpv(P,T)`) şi dreapta care realizează separarea acestora în două clase complementare (`plotpc(net.IW{1},net.b{1})`).

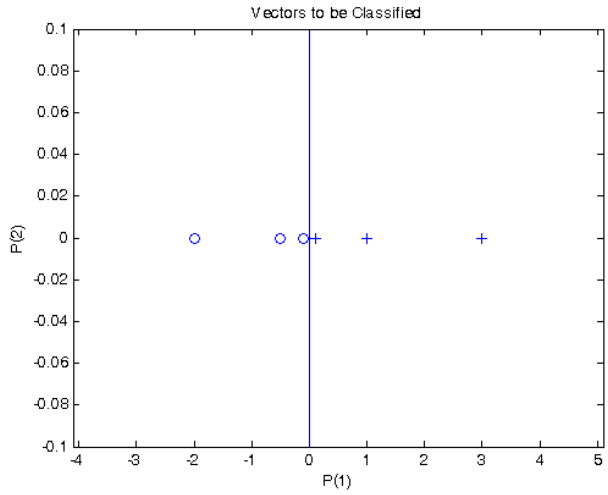

Figura 7. Reprezentarea vectorilor de intrare clasificați,
a vectorului *target* de ieşire şi a dreptei care realizează împarțirea spațiului
de intrare în două clase complementare

Se consideră un nou vector de intrare pe care dorim să îl clasificăm, p . Pentru a-l distinge de vectorii de intrare prezentați în setul de antrenament, îl vom colora cu roşu (Figura 8).

Cu ajutorul funcției *net* vom calcula ieşirea rețelei pentru noua intrare considerată.

```
%testarea retelei
a=net(P)
%testarea retelei pentru alt set de intrari
x=[-4];
a = net(x)
plotpv(x,a);
point = findobj(gca,'type','line');
set(point,'Color','red');
```

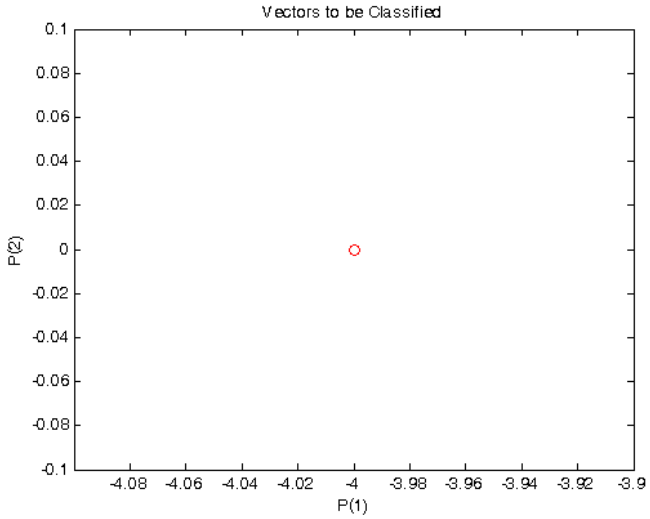

Figura 8. Reprezentarea noului vector de intrare clasificat cu rețeaua antrenată

Perceptronul a clasificat corect noua valoare de intrare (-4), fiind reprezentată prin simbolul „o” , care caracterizează numerele negative.

În Figura 9, am reprezentat dreapta care realizează clasificarea şi vectorii de intrare clasificaţi (atât cei din setul de antrenament, cât şi vectorul nou considerat).

```
hold on;
plotpv(P,T);
plotpc(net.IW{1},net.b{1});
hold off;
```

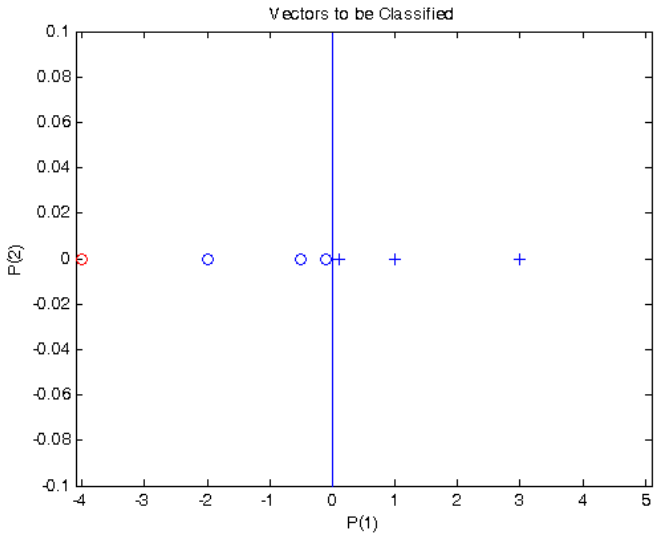

Figura 9. Reprezentarea vectorilor de intrare clasificaţi şi a dreptei de clasificare

Probleme de algebra booleana

Exemplul 1

Sa se creeze un perceptron pentru reprezentarea functiei logice AND si sa se reprezinte grafic datele de intrare si iesire si dreapta separatoare.

Se realizeaza tabela functiei booleene AND pentru doua propozitii logice p si q (p and q) (Figura 10).

p	q	p AND q
0	0	0
0	1	0
1	0	0
1	1	1

Figura 10.Functia booleana AND

Observatii: Ponderile vor fi initializate cu valori aleatoare cuprinse intre -0.5 si 1.5.

```
%Program Matlab - reprezentare functie AND
P=[0 0 1 1; 0 1 0 1];
T=[0 0 0 1];
%creare perceptron
net=perceptron;
net = configure(net,P,T);
%initializare aleatoare ponderi intre -0.5 si 1.5
```

```
net.IW{1}=-0.5 + (1-(-0.5))* rand(1,2)
%afisarea ponderilor si a bias-ului inainte de antrenare
net.IW{1}
net.b{1}
%stabilirea numarului de iteratii
net.trainParam.epochs = 50;
%stabilirea erorii permise
net.trainParam.goal = 0.01;
%antrenarea retelei
net=train(net,P,T);
%afisarea ponderilor si a bias-ului dupa antrenare
net.IW{1}
net.b{1}
%reprezentarea grafica a celor 4 puncte
plotpv(P,T);
hold on;
%reprezentarea dreptei de decizie care separa regiunile de clasificare
plotpc(net.IW{1}, net.b{1});
hold off;
%testarea retelei
y=net(P)
%testarea retelei pentru alt set de intrari
intrare=[1 0 1 0; 0 0 1 0];
y=net(intrare)
```

Perceptronul este antrenat sa reprezinte functia booleana AND. Dupa antrenare, se obtin cele doua regiuni de clasificare delimitate de dreapta de decizie (Figura 11).

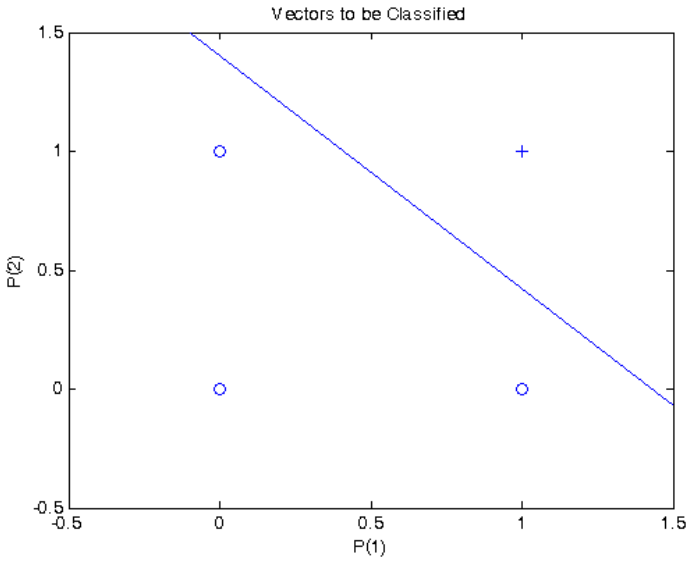

Figura 11. Dreapta de decizie pentru functia AND

Functia booleana AND, reprezentata de perceptron, functioneaza corect, intrucat la simularea retelei neuronale pentru alt set de intrari, se genereaza iesiri corecte.

Exemplul 2

Sa se creeze un perceptron pentru reprezentarea functiei logice OR si sa se reprezinte grafic datele de intrare si iesire si dreapta separatoare.

Se realizeaza tabela functiei booleene OR pentru doua propozitii logice p si q (p and q) (Figura 10).

p	q	p OR q
0	0	0
0	1	1
1	0	1
1	1	1

Figura 12.Functia booleana OR

Observatii: Ponderile vor fi initializate cu valori aleatoare cuprinse intre -0.5 si 1.5.

```
%Program Matlab - reprezentare functie AND
P=[0 0 1 1; 0 1 0 1];
T=[0 1 1 1];
%creare perceptron
net=perceptron;
net = configure(net,P,T);
%initializare aleatoare ponderi intre -0.5 si 1.5
net.IW{1}=-0.5 + (1-(-0.5))* rand(1,2)
%afisarea ponderilor si a bias-ului inainte de antrenare
net.IW{1}
net.b{1}
%stabilirea numarului de iteratii
net.trainParam.epochs = 50;
%stabilirea erorii permise
net.trainParam.goal = 0.01;
%antrenarea retelei
net=train(net,P,T);
%afisarea ponderilor si a bias-ului dupa antrenare
net.IW{1}
net.b{1}
%reprezentarea grafica a celor 4 puncte
plotpv(P,T);
hold on;
%reprezentarea dreptei de decizie care separa regiunile de clasificare
plotpc(net.IW{1}, net.b{1});
hold off;
%testarea retelei
y=net(P)
%testarea retelei pentru alt set de intrari
intrare=[1 0 1 0; 0 0 1 0];
y=net(intrare)
```

Perceptronul este antrenat sa reprezinte functia booleana OR. Dupa antrenare, se obtin cele doua regiuni de clasificare delimitate de dreapta de decizie (Figura 11).

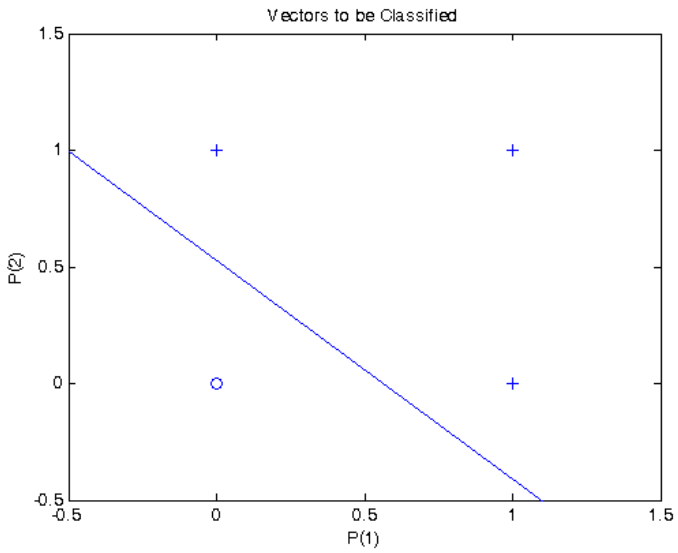

Figura 13. Dreapta de decizie pentru functia OR

Functia booleana OR reprezentata de perceptron, functioneaza corect, intrucat la simularea retelei neuronale pentru alt set de intrari, se genereaza iesiri corecte.

Problema propusa:

Sa se incerce sintetizarea functiei logice XOR cu ajutorul perceptronului simplu si sa se arate limitarea acestuia in rezolvarea problemelor de non linear separabilitate.