. 2 .
REȚELE NEURONALE UNIDIRECȚIONALE CU FUNCȚII DE ACTIVARE TREAPTĂ. REZOLVAREA PROBLEMELOR DE CLASIFICARE

Obiectivele lucrării

- prezentarea reţelelor neuronale unidirecționale cu funcții de activare de tip treaptă (rețele perceptron) şi a problematicii aplicaţiilor de clasificare a unor date de intrare;
- rezolvarea unor probleme de clasificare utilizând reţele neurale de tip perceptron implementate în mediul Matlab și folosind pachetul Neural Network Toolbox.

Noțiuni teoretice

Reţelele neuronale unidirecționale cu funcții de activare de tip treaptă (rețele perceptron) realizează o clasificare liniară a unui set de date de intrare (vectori cheie, forme, eng. *patterns*) pe baza unor algoritmi de antrenare specifici, care exploatează forma particulară a funcţiilor de activare. Datele de intrare folosite pentru instruire sunt organizate sub forma unei liste de asociere, în care fiecare element corespunde unei perechi formate dintr-un vector cheie şi clasa de apartenenţă a acestuia. Dacă au fost memorate suficient de multe obiecte din fiecare clasă, se poate construi o reprezentare internă "prototipică" a fiecărei clase prin ponderile de conexiune ale reţelei. În faza de lucru, reţeaua va stabili pentru o formă de intrare oarecare, clasa prototip corespunzătoare (aferentă).

Modelul perceptronului

Modelul unei unități funcționale din cadrul unei rețele de tip perceptron este similar modelului neuronal de bază, prezentat în lucrarea anterioară, cu particularitatea că la perceptron funcția

de activare este de tip treaptă. Păstrând notațiile din descrierea modelului neuronal de bază, modelul perceptronului este descris de relațiile:

$$s = \sum_{i=1}^{n} w_i x_i + b \tag{2.1}$$

$$y = f(s) \tag{2.2}$$

Variantele uzuale ale funcției de activare sunt funcția treaptă asimetrică:

$$f(s) = \begin{cases} 0, & s \geq 0 \\ 1, & s < 0 \end{cases} \tag{2.3}$$

și funcția treaptă simetrică:

$$f(s) = \begin{cases} -1, & s \geq 0 \\ 1, & s < 0 \end{cases} \tag{2.4}$$

Probleme de clasificare liniară. Condiția de liniar-separabilitate.

Deoarece mulțimea valorilor semnalului de ieșire al unui perceptron simplu conține două valori, $y \in \{0,1\}$, respectiv $y \in \{-1,1\}$, se poate stabili că acesta realizează o clasificare a datelor sale de intrare în două categorii: clasa A care corespunde ieşirii $y = 0$, respectiv clasa B care corespunde ieşirii $y = 1$. Într-o rețea perceptron cu m unități funcționale (adică cu m semnale de ieșire), numărul de combinații posibile ale acestora (forme de ieșire) este 2^m, adică este de asemenea un număr finit. O astfel de rețea face o clasificare liniară a datelor de intrare în 2^m clase.

Pentru a putea face o clasificare liniară, datele de intrare trebuie să respecte *condiția de liniar-separabilitate*, care se exprimă după cum urmează. Fie P perechi de date de antrenare $(\mathbf{x}^k, d^k)$, $k = \overline{1,P}$, care sunt disponibile în faza de antrenare, unde d^k este o singură valoare, nu un vector, deci am considerat o rețea cu o unitate funcțională ($m = 1$). Ținând cont de clasele stabilite prin valorile d^k, vectorii de intrare sunt liniar-separabili în două clase dacă există cel puțin un hiperplan în spațiul n dimensional care separă datele de intrare conform claselor stabilite.

Condiția este cel mai clar descrisă pentru două mărimi de intrare, adică cu vectorul intrărilor în forma $\mathbf{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^{\mathrm{T}} \in \mathrm{P}^{2\times 1}$. Astfel, se reprezintă grafic toate punctele (x_1, x_2) din setul de date de antrenare (Figura 2.1) și dacă se poate trasa o dreaptă care să separe punctele conform claselor predefinite atunci mulțimea exemplelor de antrenare respectă condiția de liniar-separabilitate.

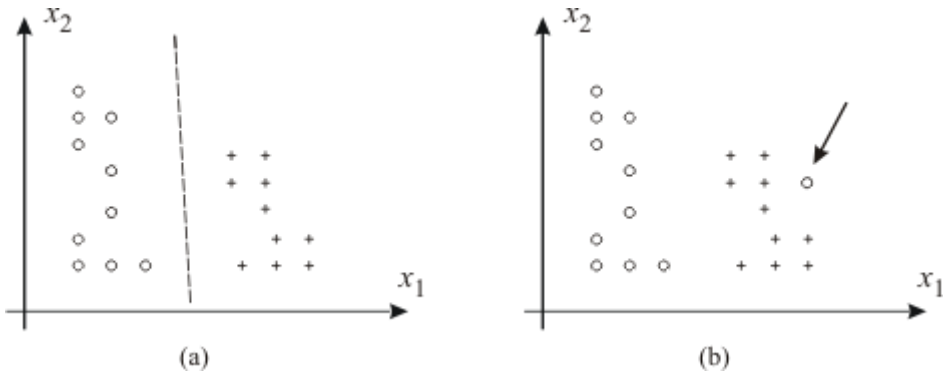

Figura 2.1. a) Date de antrenare liniar separabile; b) date de antrenare care nu sunt liniar separabile – punctul marcat nu permite trasarea unei drepte de separație.

Desfășurarea lucrării. Exemple

Implementarea Matlab a rețelelor neuronale de tip perceptron

În Matlab, reţeaua de tip perceptron este alcătuită dintr-un singur strat de unităţi având funcţii de transfer de tip **heaviside** sau **signum**. Acest tip de reţea se antrenează cu algoritmul specific, care are proprietatea că pentru o problemă de clasificare a unui set de date de antrenare liniar separabile conduce la obținerea hiperplanelor de separație după un număr finit de iteraţii. În varianta de implementare din Matlab, se poate stabili un număr maxim de parcurgeri ale datelor de antrenare, ceea ce permite utilizarea funcției de antrenare și pe date care nu respectă condiția de liniar separabilitate. Rețeaua antrenată obținută cu această limitare este de cele mai multe ori nesatisfăcătoare, în sensul că nu clasifică corect toate datele de intrare, însă își dovedește utilitatea în cazul unui volum mare de date de antrenare, despre care nu știm dinainte dacă sunt liniar separabile sau nu.

- **Inițializarea unei rețele de tip perceptron**

Pentru crearea unui perceptron cu n semnale de intrare şi m unităţi funcţionale (semnale de ieșire) se foloseşte funcţia **newp**:

```
perceptron = newp([min_1 max_1; ...; min_n max_n], m, fct_transfer, alg_invatare)
```

unde:

- min_i și max_i sunt valorile minimă și maximă a semnalului de intrare x_i; numărul de linii ale acestei matrice stabilește numărul de semnale de intrare;
- m este numărul de neuroni (numărul de semnale de ieșire),

- fct_transfer reprezintă funcţia de transfer şi poate lua una dintre valorile 'hardlim' (valoarea implicită) şi 'hardlims',
- alg_invatare reprezintă varianta de algoritm de învăţare şi poate lua una dintre valorile 'learnp' (algoritmul standard, valoarea implicită) şi 'learnpn' (varianta normalizată a algoritmului).

Valorile limită ale semnalelor de intrare sunt fie adoptate pe baza cunoștințelor despre aplicația concretă (vezi mai departe exemplele cu porți logice), fie determinate din datele de antrenare disponibile folosind în mod corespunzător funcțiile **min** și **max**.

Numărul unităților de ieșire se alege cunoscând faptul că m unităţi funcţionale pot clasifica datele de intrare în $q = 2^m$ clase.

O secvență de program pentru inițializarea unui perceptron ar arăta astfel:

```
inputs_minmax = [0 1; 0 1];
output_units = 1;
perceptron = newp(inputs_minmax, output_units);
```

Funcţia **newp** returnează o structură de date neomogenă ce conţine toate informaţiile privind reţeaua.

- **Antrenarea reţelei.**

Setul de date de antrenare se prezintă sub forma a două matrice, conținând valorile semnalelor de intrare, respectiv valorile asociate semnalelor de ieșire care definesc clasele. Fiecare semnal are valorile din datele de antrenare enumerate pe linie. Evident, cele două matrice trebuie să aibă același număr de coloane, iar numărul de linii trebuie să corespundă cu numărul de semnale de intrare și ieșire de la inițializarea rețelei.

Antrenarea propriu-zisă se face cu una dintre funcțiile **adapt** (antrenare incrementală) sau **train** (antrenare pe lot), ambele descrise în lucrarea precedentă. Pentru **adapt**, se stabilește numărul de parcurgeri ale setului de date de antrenare inițializând un parametru al structurii de date neomogenă obținută anterior și anume perceptron.adaptParam.passes. Liniile de cod ar arăta astfel:

```
perceptron.adaptParam.passes = 10;
perceptron_antrenat = adapt(perceptron, train_data_inputs, train_data_output);
```

În cazul antrenării cu **train** parametrul care trebuie inițializat este perceptron.trainParam.goall prin care se stabilește eroarea de antrenare admisă. Dacă datele de antrenare sunt liniar separabile, atunci valoarea acestui parametru poate fi zero.

```
perceptron.trainParam.goal = 0;
```

```
perceptron_antrenat = train(perceptron, train_data_inputs, train_data_output);
```

- **Vizualizarea rezultatelor antrenării.**

  În urma antrenării, valorile ponderilor semnalelor de intrare și cele ale pragurilor de activare pentru fiecare unitate funcțională sunt disponibile în vectorii de celule

```
perceptron_antrenat.IW{1};
perceptron_antrenat.b{1};
```

  iar structura nivelului de intrare a rețelei este

```
perceptron_antrenat.inputs{1};
```

- **Testarea rețelei antrenate.**

  Pentru simularea reţelei, se utilizează funcţia Matlab **sim**:

```
output = sim(perceptron_antrenat, test_data_inputs);
```

  unde test_data_inputs este un vector coloană conținând valori de test ale semnalelor de intrare. (Vezi lucrarea precedentă.) Aceste valori pot fi din datele de antrenare, caz în care valorile ieșirilor rețelei output trebuie să fie identice cu valorile impuse în datele de antrenare train_data_output, sau pot fi alte valori de testare.

- **Vizualizarea rezultatelor antrenării**

  În cazul în care datele de intrare pot fi reprezentate grafic în Matlab (adică sunt cel mult trei variabile de intrare), atunci rezultatul antrenării se poate reprezenta grafic. Pentru afișarea datelor de intrare sub forma unor puncte se folosește funcția **plotpv**:

```
plotpv(train_data_inputs, train_data_output);
```

  iar pentru reprezentarea hiperplanurilor de separație ale claselor se folosește funcția **plotpc**:

```
plotpc(perceptron_antrenat.IW{1}, perceptron_antrenat.b{1});
```

Exemple

- **Exemplul 2.1.**

  Implementarea unor rețele de tip perceptron pentru realizarea operațiilor logice AND și OR.

- *Rezolvare*

Datele pentru antrenarea unor rețele de tip perceptron care realizează operații logice sunt cele care descriu modul lor de operare pentru două variabile (Tabelul 2.1).

Tabelul 2.1. Datele de antrenare corespunzătoare operațiilor logice AND, OR și XOR.

x_1	x_2	d AND	d OR	d XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Mărimile de intrare sunt binare, așadar valorile minimă și maximă ale fiecăreia (necesare pentru funcția **newp**) sunt 0, respectiv 1. Deoarece mulțimea rezultatelor posibile are doar două elemente, este necesară o rețea cu o singură unitate funcțională.

Păstrând notațiile din lucrarea anterioară, datele de antrenare pentru operația AND sunt perechile:

$$\mathscr{P} = \{([x_1, x_2], d)^k\}\Big|_{k=\overline{1,4}} = \{([0,0],0);([0,1],0);([1,0],0);([1,1],0)\}$$

care în Matlab sunt exprimate prin inițializările:

```
train_data_inputs = [0 0 1 1; 0 1 0 1];
train_data_output = [0 1 1 1];
```

Fiecare linie a matricii *train_data_inputs* corespunde unei mărimi de intrare.

Pentru operația AND, programul complet este următorul:

```
% valorile extreme ale marimilor de intrare
% numarul de linii determina numarul de intrari ale perceptronului
inputs_minmax = [0 1; 0 1];

% numarul de unitati functionale
output_units = 1;

% initializarea perceptronului
perceptron = newp(inputs_minmax, output_units);
```

```
% setul de date de antrenare,
% format din valorile celor doua marimi de intrare si valorile impuse marimilor de iesire
train_data_inputs = [0 0 1 1;
                     0 1 0 1];
train_data_output = [0 0 0 1];

% numarul de parcurgeri ale setului de date de antrenare (ales prin incercari)
perceptron.adaptParam.passes = 10;

% antrenarea retelei (incrementala, cu functia adapt)
perceptron = adapt(perceptron, train_data_inputs, train_data_output);

% testarea reteaua pe datele de antrenare
results = sim(perceptron, train_data_inputs);
% results este egal cu train_data_output

% reprezentarea grafica a datelor de antrenare ...
plotpv(train_data_inputs, train_data_output);
hold on;
% si dreapta de separatie a celor doua clase obtinuta prin antrenare
plotpc(perceptron.IW{1}, perceptron.b{1});
hold off;
```

În urma execuției programului se obțin ponderile $w_1 = 1$, $w_2 = 1$ și valoarea de prag $b = -2$, adică ecuația dreptei de separare a celor două clase va fi $x_1 + x_2 - 2 = 0$ (Figura 2.2).

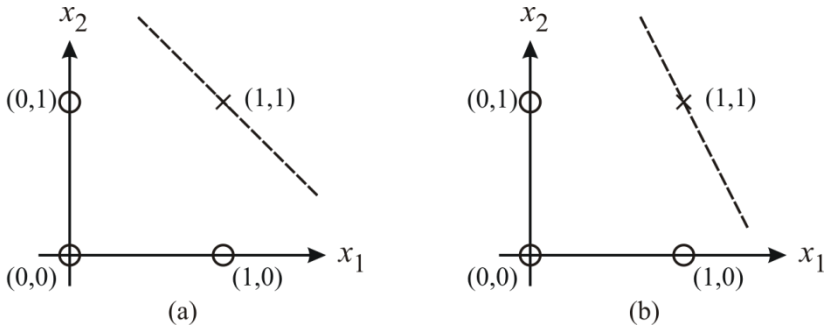

Figura 2.2. Datele de antrenare corespunzătoare operației logice AND și dreptele de separare ale acestora obținute prin antrenarea unui perceptron cu funcțiile: a) „adapt” și b) „train”.

În cazul antrenării folosind funcția **train**, deoarece datele de antrenare sunt liniar separabile, putem stabili eroarea maximă admisă zero

```
perceptron.trainParam.goal = 0;
```

Linia de cod pentru antrenarea perceptronului devine

```
perceptron = train(perceptron, train_data_inputs, train_data_output);
```

În această variantă se obțin ponderile $w_1 = 2$, $w_2 = 1$, valoarea de prag $p = -3$ și ecuația dreptei de separare a celor două clase $2x_1 + x_2 - 3 = 0$ (Figura 2.2). Antrenarea se finalizează după 6 parcurgeri ale datelor de antrenare (epoci – "epochs" în Matlab). Trebuie notat că dacă revenim la antrenarea cu funcția **adapt** și stabilim numărul de parcurgeri ale datelor de antrenare la 6 nu obținem același rezultat ca la antrenarea cu funcția **train**.

Pentru realizarea operației OR, se modifică datele de antrenare astfel:

```
train_data_inputs = [0 0 1 1; 0 1 0 1];
train_data_output = [0 1 1 1];
```

În cazul antrenării incrementale (cu **adapt**), se obține $w_1 = 2$, $w_2 = 2$, $p = -1$ și ecuația dreptei de separare $2x_1 + 2x_2 - 1 = 0$, iar în cazul antrenării pe lot (cu **train**) se obține $w_1 = 1$, $w_2 = 1$, $p = -1$ și ecuația dreptei de separare $x_1 + x_2 - 1 = 0$ (Figura 2.3).

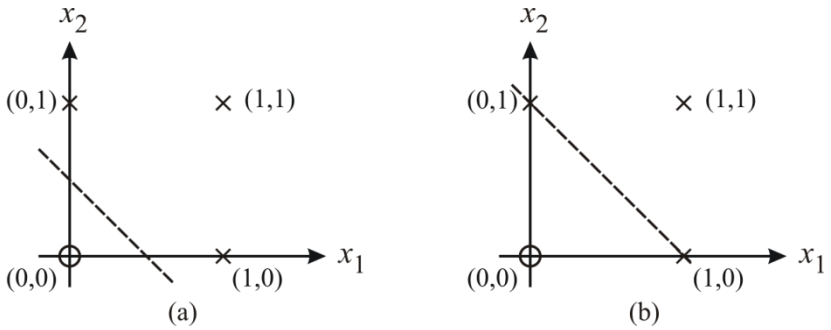

Figura 2.3. Datele de antrenare corespunzătoare operației logice OR și dreptele de separare ale acestora obținute prin antrenarea unui perceptron cu funcțiile: a) „adapt" și b) „train".

Pentru a exemplifica dificultatea cu privire la condiția ca datele de antrenare să fie liniar separabile, să studiem funcționarea algoritmilor de antrenare pentru cazul operației XOR. Datele de antrenare sunt descrise în Tabelul 2.1 și se observă că acestea nu respectă condiția (Figura 2.4). În acest caz, datele de antrenare se inițializează cu

```
train_data_inputs = [0 0 1 1; 0 1 0 1];
train_data_output = [0 1 1 0];
```

Vom păstra limita de 10 epoci pentru algoritmul de antrenare incremental și eroarea zero admisă pentru antrenarea pe lot. Antrenarea cu funcția **adapt** nu produce nici un rezultat, în sensul că rețeaua rămâne în forma inițială ($w_1 = 0$, $w_2 = 0$, $p = 0$), iar antrenarea pe lot

conduce la o rețea care clasifică eronat 2 perechi de date din cele 4 (cu $w_1 = -1$, $w_2 = 0$, $p = 0$).

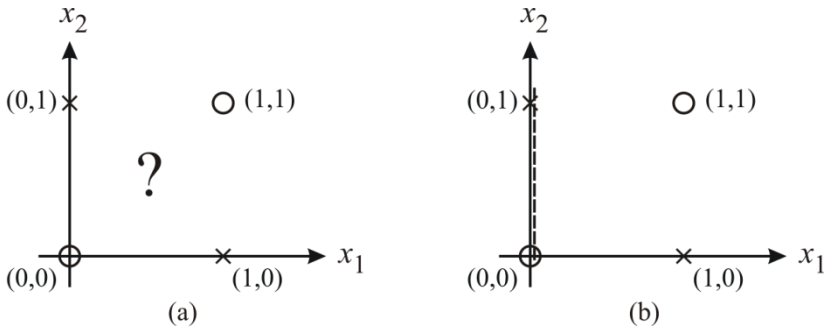

Figura 2.4. Datele de antrenare corespunzătoare operației logice XOR și dreptele de separare ale acestora obținute prin antrenarea unui perceptron cu funcțiile: a) „adapt" și b) „train".

- **Exemplul 2.2.**

Fie setul de date de intrare din tabelul următor, grupate în două clase notate A și B. Se cere antrenarea unei rețele de tip perceptron pe baza acestor date.

Tabelul 2.2. Date de antrenare liniar separabile în două clase.

x_1	1	1	2	2	3	2	1	1	2	1
x_2	1	2	1	3	1	6	7	8	8	9
Clasa	A	A	A	A	A	A	A	A	A	A

x_1	7	8	9	8	9	6	6	5	5	6
x_2	1	1	1	2	2	6	7	7	8	8
Clasa	B	B	B	B	B	B	B	B	B	B

- *Rezolvare*

Vom asocia clasei A valoarea 0 a mărimii de ieșire, iar clasei B valoarea 1. Astfel, datele de antrenare vor fi:

```
train_data_inputs = [1 1 2 2 3 2 1 1 2 1 7 8 9 8 9 6 6 5 5 6;
                     1 2 1 3 1 6 7 8 8 9 1 1 1 2 2 6 7 7 8 8];
train_data_output = [0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1];
```

De această dată, limita de 10 epoci pentru antrenarea incrementală nu este suficientă și vom stabili această limită la 100. (De fapt, se testează mai multe valori pentru numărul de

parcurgeri, păstrând o valoare suficient de mică pentru obținerea unei rețele corect antrenată.) Suplimentar, vom completa programul cu câteva linii care să afișeze câte perechi de date de antrenare ar fi incorect clasificate de către perceptronul antrenat.

```
% calculeaza erorile absolute pentru fiecare pereche de date
errors = abs(train_data_output - output);
% insumeaza valoarea 1 pentru oricare valoare a erorii absolute mai mare de 0 (countif)
sum( errors(:) > 0 )
```

În urma execuției programului, se obțin $w_1 = 36$, $w_2 = -5$, $p = -112$ și toate perechile de date sunt corect clasificate de perceptronul antrenat (Figura 2.5.a).

Din reprezentarea grafică a datelor de antrenare se poate observa că acestea sunt liniar separabile. Așadar, am putea folosi funcția **train** cu eroarea admisă zero. În acest caz se obține $w_1 = 3$, $w_2 = 1$, $p = -19$ și toate perechile de date corect clasificate (Figura 2.5.b).

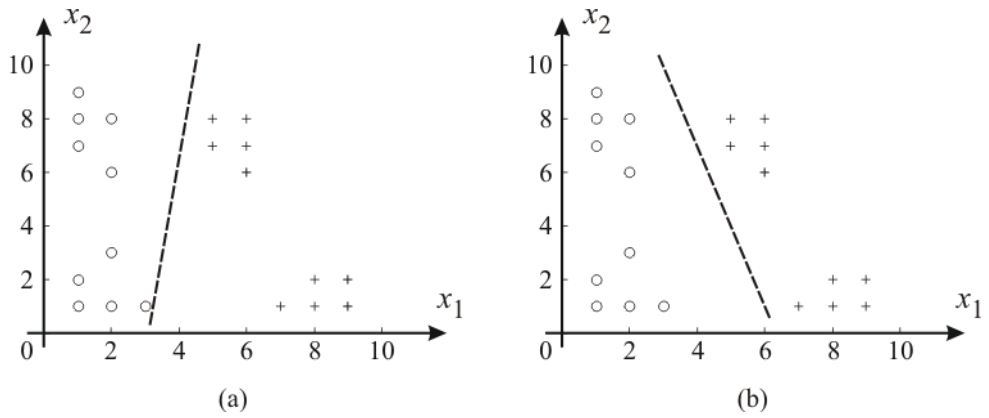

Figura 2.5. Clasificarea datelor de antrenare din tabelul 2.2 obținută prin antrenarea unui perceptron cu: a) „adapt” și b) „train”.

- **Exemplul 2.3.**

Să considerăm aceleași date de intrare de la Exemplul 1.2, clasificate de această dată în 4 grupe, notate A, B, C și D (Tabelul 2.3). Se cere antrenarea unei rețele de tip perceptron pe baza acestor date de antrenare.

Tabelul 2.3. Date de antrenare liniar separabile în patru clase.

x_1	1	1	2	2	3	2	1	1	2	1
x_2	1	2	1	3	1	6	7	8	8	9
Clasa	A	A	A	A	A	B	B	B	B	B

x_1	7	8	9	8	9	6	6	5	5	6

x_2	1	1	1	2	2	6	7	7	8	8
Clasa	C	C	C	C	C	D	D	D	D	D

- *Rezolvare*

Pentru a face o grupare în patru clase, avem nevoie de o rețea de tip perceptron cu două unități de ieșire (Figura 2.6), adică vom inițializa:

```
output_units = 2;
```

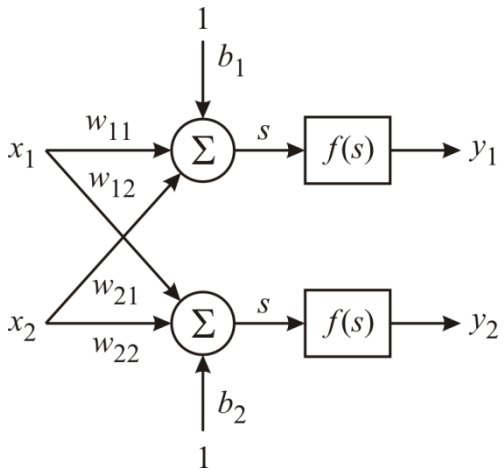

Figura 2.6. Rețea neuronală cu două unități de ieșire.

În urma antrenării, ponderile mărimilor de intrare vor defini două drepte de separație

$$w_{11}x_1 + w_{21}x_2 + p_1 = 0$$

$$w_{12}x_1 + w_{22}x_2 + p_2 = 0$$

în care w_{ij} este ponderea intrării *i* la neuronul *j*.

Vom stabili o asociere între cele patru combinații de valori ale mărimilor de ieșire (y_1, y_2) și cele patru clase ale datelor de antrenare de forma

$$(0,0) \rightarrow \text{A}\ ,\ (0,1) \rightarrow \text{B}\ ,\ (1,0) \rightarrow \text{C}\ ,\ (1,1) \rightarrow \text{D}$$

ceea ce se transpune în inițializarea datelor de antrenare de forma:

```
train_data_inputs = [1 1 2 2 3 2 1 1 2 1 7 8 9 8 9 6 6 5 5 6;
                     1 2 1 3 1 6 7 8 8 9 1 1 1 2 2 6 7 7 8 8];
train_data_output = [0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1;
                     0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1];
```

În urma antrenării, se obțin dreptele de separare $36x_1 - 5x_2 - 112 = 0$, $-13x_1 + 21x_2 - 40 = 0$, pentru antrenarea cu **adapt**, respectiv $3x_1 + x_2 - 19 = 0$, $-2x_1 + 6x_2 - 17 = 0$, pentru antrenarea cu **train**, iar perechile de date sunt corect clasificate în ambele variante (Figura 2.7).

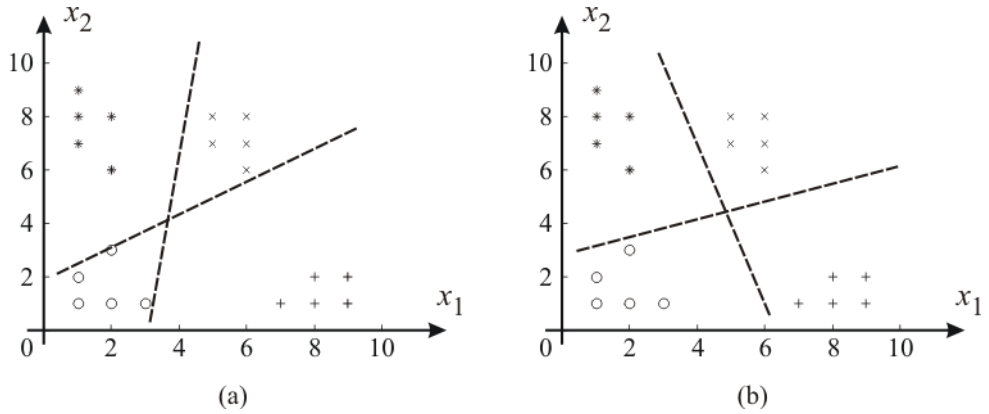

Figura 2.7. Clasificarea datelor de antrenare din tabelul 2.3 obținută prin antrenarea unei rețele de tip perceptron cu două unități de ieșire, folosind: a) „adapt” și b) „train”.

Probleme propuse

- **Problema 2.1**

  Să se implementeze în Matlab câte un perceptron pentru realizarea funcţiilor logice “și-nu” și “sau-nu”.

- **Problema 2.2**

  Fișierul “banknotes-train-data.txt” conține 1372 perechi de date de antrenare obținute prin fotografierea unor bancnote (despre care se știe inițial dacă sunt autentice sau contrafăcute) și extragerea a 4 caracteristici ale imaginilor obținute. Caracteristicile sunt: variance of Wavelet Transformed image, skewness of Wavelet Transformed image, curtosis of Wavelet Transformed image, entropy of image.

  Scrieți un program MATLAB care antrenează o rețea de tip perceptron pentru identificarea bancnotelor contrafăcute pe baza fotografierii lor, folosind datele din fișier.

  Indicații. Pentru antrenare se va folosi o parte dintre date, iar pentru testarea rețelei restul. Datele se încarcă din fișier folosind *Import Wizard* (meniul *File*, opțiunea *Import Data*).

- **Problema 2.3**

Fișierul “glass-train-data.txt” conține 214 perechi de date de antrenare obținute prin analiza în laborator a unor mostre din 7 tipuri de sticlă. Caracteristicile determinate pentru fiecare mostră sunt:

	Caracteristica (date de intrare)		Valoarea minimă	Valoarea maximă
1	indicele de refracție		1.5112	1.5339
2	Na	(procent de oxid al elementului chimic în cantitatea analizată)	10.73	17.38
3	Mg		0	4.49
4	Al		0.29	3.5
5	Si		69.81	75.41
6	K		0	6.21
7	Ca		5.43	16.19
8	Ba		0	3.15
9	Fe		0	0.51

Pentru fiecare exemplu din fișier, sunt date: numărul mostrei (indicele), cele 9 caractersitici și categoria.

Să se scrie un program MATLAB pentru implementarea unei rețele de tip perceptron care să realizeze clasificarea datelor din fișier în cele 7 clase considerate.

Referințe

[1] ***, https://archive.ics.uci.edu/ml/datasets/banknote+authentication

[2] ***, https://archive.ics.uci.edu/ml/datasets/Glass+Identification