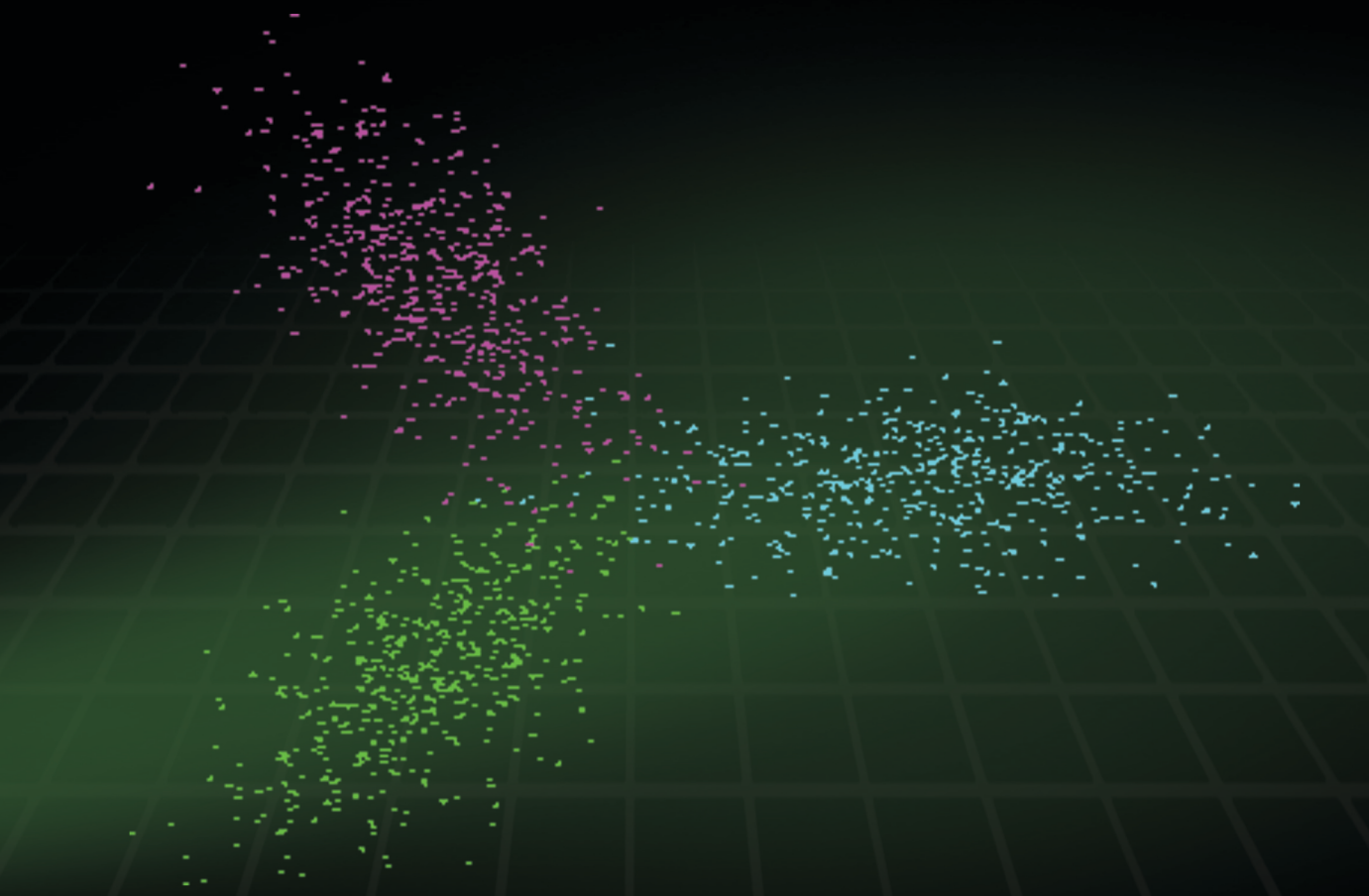
Cătălina Neghină Mihai Neghină

MATLAB

Metode de clasificare și segmentare



ISBN 978-973-0-31041-2

MATLAB

Metode de clasificare și segmentare

Cătălina NEGHINĂ

Mihai NEGHINĂ

Sibiu

2019

MATLAB: Metode de clasificare și segmentare
Ediție electronică, CD-ROM, 2019
Autori: Cătălina NEGHINĂ, Mihai NEGHINĂ
ISBN 978-973-0-31041-2

Introducere

Clasificarea și *segmentarea* sunt operații profund umane, folosite de-a lungul timpului de către oameni pentru a înțelege și a se orienta în lume. Impulsul de a împărți în clase (categorii) după caracteristici comune a dus la construirea regnurilor, ordinelor, speciilor și subspeciilor de animale (plante, fungi, etc.) în biologie, la tabelul lui Mendeleev și clasificarea substanțelor (acizi și baze, organice și anorganice, etc.) în chimie, la împărțirea substanțelor în gaze, lichide, solide (și plasmă) și la modelul standard al particulelor subatomice în fizică, la definirea unghiurilor ascuțite, drepte și obtuze, la împărțirea numerelor în întregi, fracționare și iraționale (reale/complexe) în matematică, tipurile de planete, stele sau galaxii în astronomie și așa mai departe. Tendința oamenilor de a împărți pe categorii este atât de puternică încât sunt definite clase inclusiv pentru proprietăți continue și subiective precum înălțimea (scund/înalt), vârsta (tânăr/bătrân) sau culoarea - într-un curcubeu, unde se oprește zona roșie și începe cea portocalie?

Clasificarea poate fi *supervizată* sau *nesupervizată* (*clustering*). Având un set de date despre care se cunoaște împărțirea în clase și un caz de test (reprezentat de obicei ca un vector de proprietăți), *clasificarea supervizată* răspunde problemei de atribuire a etichetei uneia dintre clase pentru cazul de test. Spre exemplu, o planetă nou descoperită într-un alt sistem planetar este telurică precum Pământul, gigant gazos ca Jupiter, sau un bulgăre de gheață precum corpurile din centura Kuiper? Clasificarea supervizată necesită o etapă de învățare, în care clasificatorul deduce regulile de clasificare pornind de la un set de date pre-etichetat.

Clasificarea nesupervizată (clustering) atacă problema realizării împărțirii unei mulțimi în clase, astfel încât în interiorul unei clase să existe similaritate cât mai mare între membri, iar între clase să existe diferențe cât mai mari. În cazul clasificării nesupervizate, nu este nevoie de antrenare, algoritmul se auto-organizează fără intervenție externă.

Segmentarea se referă la partiționarea unei imagini în seturi de pixeli, în general pentru delimitarea obiectelor de interes. Deși din punct de vedere spațial segmentarea nu pare legată de clasificare, trecerea în spațiul culorilor reduce problema la o clasificare (adesea nesupervizată), cu câteva constrângeri suplimentare legate de poziționarea spațială relativă a pixelilor unii față de alții. De exemplu, în majoritatea prelucrărilor de imagini care implică segmentare există și operații de pre- sau post-procesare care să filtreze sau să elimine pixelii izolați.

Lucrarea este adresată în special studenților facultăților de inginerie, însă poate fi utilă oricărei persoane interesate de principiile clasificării și segmentării. Exemplele prezentate sunt generate în totalitate în MATLAB (versiunea 2019a), lucrarea de față venind în completarea cărții "MATLAB. *Un prim pas spre cercetare*" scrisă de aceeași autori. Diverșii algoritmi de clasificare și segmentare prezentați în această carte stau la baza multor aplicații, de la detecția și identificarea obiectelor, recunoaștere vocală sau de text, până la sortarea automată pe liniile de producție sau secvențierea ADN-ului.

Lucrarea este împărțită în 5 capitole astfel: *Capitolul 1* oferă o privire de ansamblu asupra particularităților programului MATLAB și a avantajelor pe care acesta le oferă pentru înțelegerea și implementarea algoritmilor descriși; *Capitolul 2* prezintă indicatorii de performanță cei mai folosiți pentru a interpreta rezultatele clasificării; următoarele capitole urmăresc prezentarea teoretică și implementarea în MATLAB a câtorva dintre cele mai întâlnite metode de clasificare supervizată (*Capitolul 3*), nesupervizată (*Capitolul 4*) și de segmentare (*Capitolul 5*).

Autorii Cătălina Neghină și Mihai Neghină sunt cadre didactice ale *Facultății de Inginerie*, Universitatea *Lucian Blaga* din Sibiu, predând materii pentru care clasificarea și segmentarea sunt subiecte importante, precum *Prelucrarea Numerică a Semnalelor*, *Prelucrarea Imaginilor* și *Rețele Neurale*.

Cuprins

Capitolul 1

Scurtă introducere în MATLAB

MATLAB-ul este un limbaj de nivel înalt, folosit intens în cercetare și în inginerie, ce permite implementarea cu ușurință a algoritmilor din diverse domenii. Fiind optimizat pentru calcul matriceal, MATLAB-ul oferă un bun suport pentru crearea și operarea cu vectori de proprietăți sau coordonate (frecvent întâlniți în probleme de clasificare) sau cu imagini (probleme de segmentare).

Generarea matricelor de confuzie sau implementarea relațiilor din rețelele neurale în formă matriceală este ușoară și intuitivă, evitându-se folosirea buclelor `for` inevitabile în alte limbaje. De asemenea, forma compactă inerentă scrierii matriceale permite utilizatorului să se concentreze mai degrabă pe ideea implementată decât pe sintaxa folosită. Spre exemplu, segmentarea unei imagini grayscale (variabila `imagGray`) cu un prag impus (variabila `pragT`) și obținerea de etichete logice (variabila `etichete`) se poate face cu o singură linie de cod, indiferent de dimensiunea imaginii:

```
>> etichete = (imagGray > pragT);
```

Un alt avantaj îl reprezintă încărcarea și vizualizarea datelor. MATLAB-ul este capabil să citească și să scrie informația dintr-o/într-o gamă largă de fișiere, de la fișiere text, csv (en. *comma separated values*), excel, semnale audio, semnale video și imagini cu diverse formate, atât formate commune (*.jpg, *.png etc) cât și formate de nișă specifice anumitor domenii (de exemplu imagini satelitare cu *n*-benzi).

Reprezentarea grafică a datelor sau rezultatelor în MATLAB este rapidă și extrem de flexibilă, fiind disponibile, printre altele, funcții de reprezentare 2D, 3D, histograme și reprezentări vectoriale:

- Funcții de reprezentare în spațiul 2D:
  - `plot, stem, bar, pie, hist, ezplot` etc
- Funcții de reprezentare în spațiul 3D:
  - `surf, mesh` etc
- Funcții pentru citirea și afișarea imaginilor:
  - `imread, imshow, image` etc
- Funcții pentru afișarea histogramelor:
  - `hist, imhist` etc

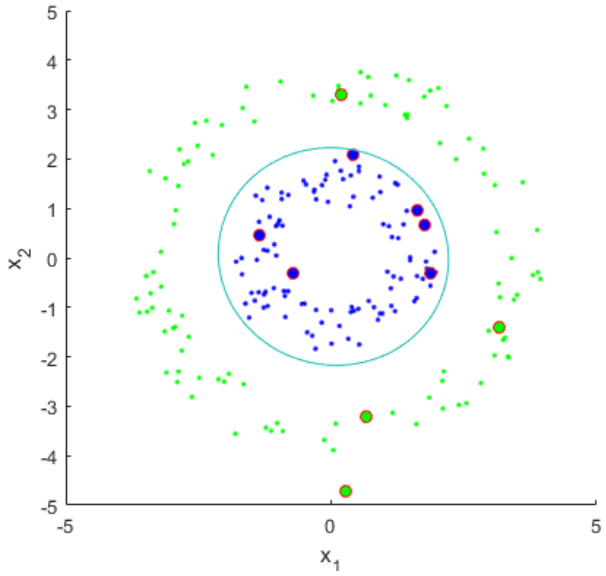


Figura 1.1. Reprezentarea vectorilor de forma $X = [x_1, x_2]$ cu funcția `plot` și a suprafeței de separație cu funcția `ezplot`

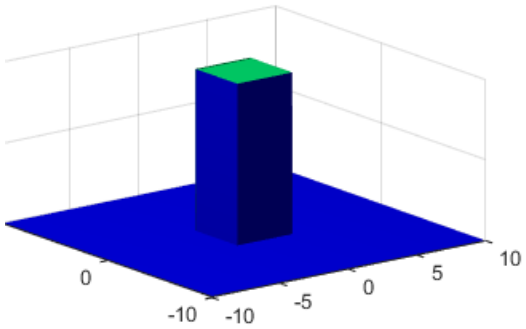


Figura 1.2. Reprezentarea vecinătății de tip *bubble* cu funcția `surf`

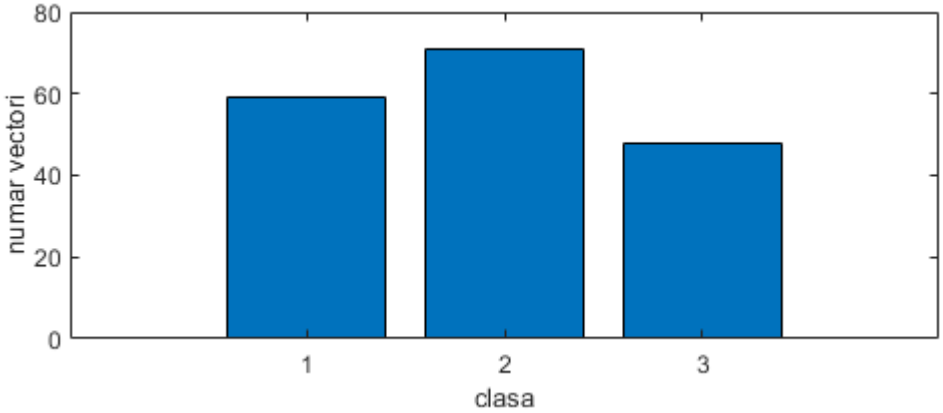


Figura 1.3. Calculare histogramă cu funcția `hist` și reprezentare cu funcția `bar`

Editorul MATLAB-ului este intuitiv și permite oprirea temporară a rulării cu ajutorul punctelor de întrerupere (*breakpoints*). In timpul unei astfel de întreruperi, utilizatorul poate interacționa cu datele atât din *Command Window* cât și din *Workspace*. In fereastra *Command Window* se pot scrie și rula linii de cod adiționale care să prelucreze datele existente în *Workspace* sau să introducă variabile noi. In *Workspace*, datele existente pot fi vizualizate sau modificate arbitrar, astfel încât la reluarea rulării să fie folosite noile valori.

Pe lângă suportul oferit cu operațiile matriceale, MATLAB-ul pune la dispoziție funcții de prelucrare a vectorilor și matricelor, precum:

- generarea de matrice unitate sau cu valori de 0 și 1: `ones, zeros, eye`
- funcții care operează cu dimensiunile matricelor: `kron, repmat, resize, size, length` etc
- generarea de matrice de numere aleatoare: `random, rand, normrnd` etc
- funcții uzuale pentru operații pe linii sau coloane: `min, max, floor, round, ceil, sum, prod, mean, median, std, abs, mod` etc
- generare de matrice de confuzie: `confusionmat`

MATLAB-ul oferă chiar și implementări optimizate pentru algoritmi clasici precum:

- funcția `fcm` (pentu algoritmul *fuzzy C-means*)
- funcția `kmeans` (pentru algoritmul *k-means clustering*)
- funcția `imbinarize` (realizează segmentarea în diverse moduri, depinzând de parametrii de intrare)
- funcția `knnsearch` (pentru algoritmul *k-Nearest Neighbour*)
- funcția `feedforwardnet` (pentru implementarea unei rețele MLP)
- funcția `selforgmap` (pentru implamentarea unei rețele SOM)
- etc

Una dintre cele mai importante proprietăți ale MATLAB-ului este existența toolbox-urilor, atât pentru aplicații demonstrative cât și pentru implementarea sau ajustarea algoritmilor studiați intr-o interfață grafică interactivă. Relevante pentru metodele prezentate în această lucrare sunt toolbox-urile din *tab*-ul *Apps*, categoria *Machine Learning and Deep Learning: Neural Net Clustering* și *Neural Net Pattern Recognition*

Figura 1.4. Toolbox-uri în MATLAB

Existența unui număr mare de algoritmi pre-implementați, inclusiv pentru clasificare, prelucrări de imagini și statistică, la care se adaugă toolbox-urile și multitudinea de exemple și aplicații demonstrative, fac posibilă înțelegera rapidă a metodelor și testarea ușoară a ideilor în MATLAB. Insă deși este o resursă excelentă pentru învățare și cercetare, acest limbaj vine la pachet și cu câteva dezavantaje:

- una dintre cele mai importante critici aduse limbajului MATLAB este indexarea de la 1, similară limbajului natural (în care numărătoarea obiectelor începe de la 1). Codul MATLAB este în general ușor de citit și explicat: `v(2)` este al doilea element al vectorului `v`. Această indexare devine însă o problemă atunci când se încearcă implementarea unor algoritmi matematici, deoarece convenția matematică începe numărătoarea de la 0.
- lentoarea rulărilor în MATLAB, comparativ cu alte limbaje precum C, și faptul că MATLAB-ul în sine este consumator de resurse (memorie și procesare). Pe de altă parte există posibilitatea de a genera cod C și de a apela librării externe precum *OpenCV*, care diminuează efectele negative menționate mai sus.

Așadar … MATLAB. *Metode de clasificare si segmentare.*

Capitolul 2

Interpretarea rezultatelor clasificării

În cazul clasificării, este foarte importantă evaluarea performanțelor algoritmilor implementați. În cazul în care se cunosc rezultatele dorite ale clasificării, un instrument util pentru evaluarea performanțelor algoritmilor îl reprezintă **matricea de confuzie**.

Matricea de confuzie (C) este o reprezentare tabelară a rezultatelor. Dacă se dorește clasificarea în K clase, atunci matricea de confuzie C va fi o matrice de ordin K (cu K linii și K coloane). Pe linii sunt clasele reale iar pe coloane clasele estimate (prezise).

$C(i, j)$ reprezintă numărul de elemente cu eticheta reală i ce au fost clasificate în clasa j. *Obs:* suma elementelor pe linia i reprezintă numărul total de elemente din clasa i. În consecință, diagonala matricei de confuzie conține deciziile corecte de clasificare ($i = j$).

În multe aplicații este util să se normeze matricea de confuzie, astfel încât elementele sale să devină procente și nu simple numere. O modalitate clasică de normare este următoarea: se împarte fiecare element al matricei de confuzie la suma elementelor de pe linia respectivă.

$$C_{normat}(i,j) = \frac{C(i,j)}{\sum_{n=1}^{K} C(i,n)} \tag{2.1}$$

Ceea ce înseamnă că dacă se face clasificarea în 2 clase iar $C_{normat}(1,2) = 0.15$, atunci 15% dintre elementele din prima clasă au fost greșit clasificate în a doua clasă. În cazul matricei de confuzie normate, suma pe fiecare linie va fi unitară.

2.1. Matricea de confuzie - caz particular pentru $K = 2$ clase

Fie o imagine precum cea din figura alăturată, ce conține două clase: clasa K_1 reprezentând pixelii din regiunea cu aluniță (vom considera că aceasta este clasa pozitivă) și clasa K_2 reprezentând pixelii de piele (vom considera că aceasta este clasa negativă). Dacă se face clasificarea în $K = 2$ clase (clasa K_1 și clasa K_2), atunci matricea de confuzie se poate scrie ca mai jos.

Tabel 2.1. Matricea de confuzie pentru $K = 2$ clase

Matricea de confuzie (C)		Valori prezise (de algoritmul de clasificare)	
		K_1	***K_2***
Valori cunoscute	***K_1***	**TP** (en. *true positive*)	**FN** (en. *false negative*)
	K_2	**FP** (en. *false positive*)	**TN** (en. *true negative*)

Observații

- colțul din stânga sus conține numărul de elemente din clasa K_1 clasificate corect în clasa K_1 (TP – *True Positive*)
- colțul din dreapta sus conține numărul de elemente din clasa K_1 clasificate greșit în clasa K_2 (FN - *False Negative*)
- colțul din stânga jos conține numărul de elemente din clasa K_2 clasificate greșit în clasa K_1 (FP - *False Positive*); *alarmă falsă*
- colțul din dreapta jos conține numărul de elemente din clasa K_2 clasificate corect în clasa K_2 (TN - *True Negative*)

Tabel 2.2. Matricea de confuzie normată pentru $K = 2$ clase

Matricea de confuzie normată		Valori prezise (de algoritmul de clasificare)	
		K_1	***K_2***
Valori cunoscute	***K_1***	$TPR = \frac{TP}{TP + FN}$	$FNR = \frac{FN}{TP + FN}$
	K_2	$FPR = \frac{FP}{TN + FP}$	$TNR = \frac{TN}{TN + FP}$

2.1.1. Indicatori de performanță ce reies din matricea de confuzie

1. *True Positive Rate (TPR)*. TPR reprezintă raportul dintre numărul total al elementelor pozitive clasificate corect și numărul total al elementelor pozitive.

$$TPR = \frac{TP}{TP + FN} \tag{2.2}$$

2. *True Negative Rate* (TNR). TNR reprezintă raportul dintre numărul total al elementelor negative clasificate corect și numărul total al elementelor negative.

$$TNR = \frac{TN}{TN + FP} \tag{2.3}$$

3. *False Negative Rate* (FNR). FNR reprezintă raportul dintre numărul total al elementelor pozitive clasificate greșit și numărul total al elementelor pozitive.

$$FNR = \frac{FN}{TP + FN} \tag{2.4}$$

4. *False Positive Rate* (FPR). FPR reprezintă raportul dintre numărul total al elementelor negative clasificate greșit și numărul total al elementelor negative.

$$FPR = \frac{FP}{TN + FP} \tag{2.5}$$

Observații: TPR + FNR = 1, FPR + TNR =1

5. *Acuratețea.* Reprezintă raportul dintre numărul de date etichetate corect împărțit la numărul total de date.

$$Acuratetea = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.6}$$

Observație: Acuratețea este un bun indicator atunci când clasele sunt echilibrate. Când clasele nu sunt echilibrate, acuratețea nu este tocmai relevantă pentru a evalua performanțele algoritmului de clasificare.

Exemplu în care acuratețea nu este relevantă: Fie două clase K_1 și K_2; K_1 conține 100 de elemente iar K_2 conține 5 elemente. Dacă toate elementele din clasa K_1 sunt clasificate corect și niciun element din clasa K_2 nu este clasificat corect atunci matricea de confuzie arată astfel:

Matricea de confuzie (C)		Valori prezise (de algoritmul de clasificare)	
		K_1	***K_2***
Valori cunoscute	***K_1***	100 (TP)	0 (FN)
	K_2	5 (FP)	0 (TN)

$$Acuratetea = \frac{TP+TN}{TP+TN+FP+FN} = \frac{100}{105} = 0.95 = 95\%.$$

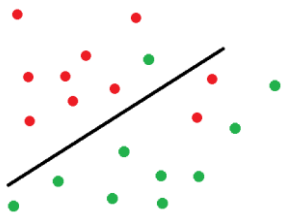
Deși niciun element din clasa K_2 nu a fost clasificat corect, scorul total de clasificare (acuratețea) este de 95%, un scor care duce cu gândul la o clasificare mult mai bună decât ceea ce se obține de fapt.

O clasificare cât mai bună presupune ca matricea de confuzie să conțină valori cât mai mari pe diagonala principală și valori cât mai mici în rest. În ceea ce privește matricea de confuzie normată, o clasificare bună presupune valori cât mai apropiate de 100% pe diagonala principală și valori cât mai apropiate de 0% în rest. Pentru exemplul dat însă, așa cum reiese și din tabelul de mai jos, acest lucru nu se respectă, deși valoarea acurateții este foarte mare.

Matricea de confuzie normată		Valori prezise (de algoritmul de clasificare)	
		K_1	***K_2***
Valori cunoscute	***K_1***	100%	0%
	K_2	100%	0%

De aceea, pentru clase dezechilibrate, o soluție de calcul a scorului global de clasificare corectă este: $\frac{TPR+TNR}{2}$. În cazul exemplului nostru, se obține un scor global de clasificare de 50%.

Exemplu 2.1. Fie 2 clase (clasa *punctelor roşii* şi clasa *punctelor verzi* – clasa de interes) clasificate ca în figura alăturată.



Cerințe:

1. Care este matricea de confuzie?
2. Care sunt valorile TPR și TNR?
3. Care este acuratețea clasificării?
4. Să se reprezinte matricea de confuzie normată.

Soluție: Considerăm cele două clase astfel:

- Clasa K_1, clasa pozitivă = reprezentată de punctele verzi
- Clasa K_2, clasa negativă = reprezentată de punctele roșii

1. Matricea de confuzie va fi:

Matricea de confuzie (C)		Valori prezise (de algoritmul de clasificare)	
		K_1(puncte verzi)	***K_2 (puncte roșii)***
Valori cunoscute	***K_1***	9 (TP)	1 (FN)
	K_2	2 (FP)	8 (TN)

2. Procentul de clasificare corectă a punctelor verzi:

$$TPR = \frac{TP}{TP + FN} = \frac{9}{9 + 1} = 0.9 = 90\%$$

Procentul de clasificare corectă a punctelor roșii:

$$TNR = \frac{TN}{TN + FP} = \frac{8}{8 + 2} = 0.8 = 80\%$$

3. Acuratețea clasificării va fi:

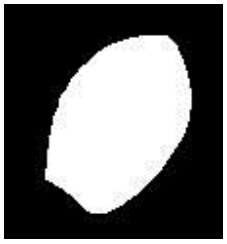
$$Acuratețea = \frac{TP + TN}{TP + TN + FP + FN} = \frac{9 + 8}{20} = 0.85 = 85\%$$

4. Matricea de confuzie normată va fi:

Matricea de confuzie normată		Valori prezise (de algoritmul de clasificare)	
		K_1(puncte verzi)	***K_2 (puncte roșii)***
Valori cunoscute	***K_1***	0.9 (90%)	0.1 (10%)
	K_2	0.2 (20%)	0.8 (80%)

Exemplu 2.2. Fie un clasificator antrenat pentru recunoașterea aluniţelor din imagini ce conţin doar piele şi aluniţe. Imaginea testată are dimensiunea 100x100 pixeli. Se ştie că regiunea reprezentând aluniţa are 4000 pixeli. În urma clasificării, pixelii au fost clasificaţi astfel:

- 3500 de pixeli de aluniţă au fost clasificaţi ca aluniţă.
- 5000 de pixeli de piele au fost clasificaţi ca piele.



Cerinţe

1. Care este procentul de clasificare corectă a pixelilor reprezentând aluniţa?
2. Care este procentul de clasificare corectă a pixelilor de piele?
3. Care este acurateţea clasificării?

Soluţie:

Considerăm cele două clase astfel:

- Clasa K_1, clasa pozitivă = reprezentată de pixelii de aluniţă
- Clasa K_2, clasa negativă = reprezentată de pixelii de piele

Matricea de confuzie va fi:

Matricea de confuzie (C)		Valori prezise (de algoritmul de clasificare)	
		K_1(aluniţă)	***K_2 (piele)***
Valori cunoscute	***K_1 (aluniţă)***	3500 (TP)	500 (FN)
	K_2 (piele)	1000 (FP)	5000 (TN)

1. Procentul de clasificare corectă a pixelilor reprezentând aluniţa va fi:

$$TPR = \frac{TP}{TP + FN} = \frac{3500}{3500 + 500} = 0.8750 = 87.5\%$$

2. Procentul de clasificare corectă a pixelilor reprezentând pielea va fi:

$$TNR = \frac{TN}{TN + FP} = \frac{5000}{5000 + 1000} = 0.8333 = 83.33\%$$

3. Acurateţea clasificării va fi:

$$Acuratetea = \frac{TP + TN}{TP + TN + FP + FN} = \frac{3500 + 5000}{10000} = 0.85 = 85\%$$

2.2. Matricea de confuzie pentru $K > 2$ clase

Dacă se dorește clasificarea în K clase ($K > 2$), atunci matricea de confuzie C va fi o matrice de ordin K (cu K linii și K coloane). Pe linii sunt clasele reale iar pe coloane clasele estimate (prezise).

$C(i, j)$ reprezintă numărul de elemente cu eticheta reală i ce au fost clasificate în clasa j. *Obs:* suma elementelor pe linia i reprezintă numărul total de elemente din clasa i. În multe aplicații este util să se normeze matricea de confuzie, astfel încât elementele sale să devină probabilități și nu simple numere. O modalitate clasică de normare este următoarea: se împarte fiecare element al matricei de confuzie la suma elementelor de pe linia respectivă.

$$C_{normat}(i,j) = \frac{C(i,j)}{\sum_{n=1}^{K} C(i,n)} \tag{2.7}$$

Ceea ce înseamnă că dacă se face clasificarea în 4 clase iar $C_{normat}(1,4) = 0.08$, atunci 8% dintre elementele din prima clasă au fost greșit clasificate în a patra clasă. În cazul matricei de confuzie normate, suma pe fiecare linie va fi unitară.

Pentru clasificarea în mai mult de 2 clase, nu se vor mai considera clase pozitive și clase negative. De exemplu, pentru clasificare în $K = 4$ clase, matricea de confuzie C va arăta astfel:

Matricea de confuzie (C)		Valori prezise (de algoritmul de clasificare)			
		K_1	K_2	K_3	K_4
Valori cunoscute	K_1	$C(1, 1)$	$C(1, 2)$	$C(1, 3)$	$C(1, 4)$
	K_2	$C(2, 1)$	$C(2, 2)$	$C(2, 3)$	$C(2, 4)$
	K_3	$C(3, 1)$	$C(3, 2)$	$C(3, 3)$	$C(3, 4)$
	K_4	$C(4, 1)$	$C(4, 2)$	$C(4, 3)$	$C(4, 4)$

2.2.1. Indicatori de performanță ce reies din matricea de confuzie

- **acuratețea** se va calcula similar clasificării în două clase, doar că se va generaliza astfel:

$$acuratete = \frac{\sum_{i=1}^{K} C(i,i)}{\sum_{i=1}^{K}\sum_{j=1}^{K} C(i,j)} \tag{2.8}$$

similar cu:

$$acuratete = \frac{suma\ elementelor\ de\ pe\ diagonala\ principală\ din\ C}{suma\ tuturor\ elementelor\ din\ matricea\ C}$$

- **procentul elementelor din clasa k clasificate corect** va fi:

$$scor_k = \frac{C(k,k)}{\sum_{n=1}^{K} C(k,n)} \tag{2.9}$$

similar cu:

$$scor_k = \frac{numărul\ elementelor\ din\ clasa\ k\ corect\ clasificate}{suma\ elementelor\ din\ matricea\ C\ de\ pe\ linia\ k}$$

Exemplu 2.3. Fie o imagine conținând 3 obiecte: clasa K_1(cer), clasa K_2 (verdeată) și clasa *K3* (apă). În urma clasificării s-a obținut următoarea matrice de confuzie.

Matricea de confuzie (C)		Valori prezise (de algoritmul de clasificare)		
		K1	***K2***	***K3***
Valori cunoscute	***K1***	1800	200	100
	K2	100	5000	400
	K3	100	300	3000

Cerințe:

1. Câți pixeli are imaginea?
2. Cum arată matricea de confuzie normată?
3. Pentru ce clasă s-a obținut cea mai bună clasificare?

Soluție:

1. Numărul total de pixeli este egal cu numărul total de elemente din matricea de confuzie ➔ 11000 de pixeli.
2. Matricea de confuzie normată se calculează cu formula:

$$C_{normat}(i,j) = \frac{C(i,j)}{\sum_{n=1}^{K} C(i,n)}, \text{ unde } K = 3$$

Matricea de confuzie normată (C_{normat})		Valori prezise (de algoritmul de clasificare)		
		K1	***K2***	***K3***
Valori cunoscute	***K1***	**85.72%**	9.52%	4.76%
	K2	1.82%	**90.91%**	7.27%
	K3	2.94%	8.82%	**88.24%**

3. Scorurile de clasificare corectă pentru fiecare clasă se regăsesc pe diagonala principală astfel:

- $C_{normat}(1,1)$ = 85.72%, reprezintă scorul de clasificare corectă pentru pixelii din clasa K_1 (cer).
- $C_{normat}(2,2)$ = 90.91%, reprezintă scorul de clasificare corectă pentru pixelii din clasa K_2 (verdeață).
- $C_{normat}(3,3)$ = 88.24%, reprezintă scorul de clasificare corectă pentru pixelii din clasa K_3 (apă).

Cel mai bine au fost clasificați pixelii din clasa K_2 (verdeață).

2.3. Calculul matricei de confuzie în MATLAB

Pentru calculul matricei de confuzie se poate folosi funcția `confusionmat`.

Sintaxă: `C = confusionmat(d1,d2)` unde:

- `d1` = date cunoscute
- `d2` = date obținute
- `C` = matricea de confuzie având pe verticală datele cunoscute și pe orizontală datele obținute; `C(i,j)` reprezintă numărul de elemente cu eticheta reală *i* ce au fost clasificate în clasa *j*.

Obs: pentru sintaxa completă `>> help confusionmat`

În cazul în care se dorește afișarea tabelară a matricei de confuzie se poate folosi funcția `confusionchart`.

Sintaxă: `confusionchart(C)` unde:

- `C` = matricea de confuzie

Observație: pentru a folosi funcția `confusionchart` nu este nevoie ca matricea de confuzie să fie deja cunoscută; aceasta se poate calcula și reprezenta în același timp cu ajutorul funcției `confusionchart`, astfel:

`C = confusionchart(d1,d2)` unde:

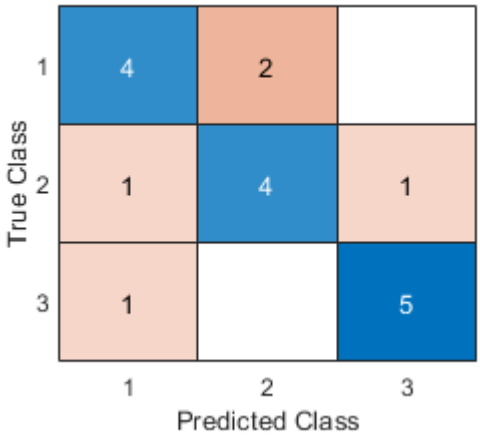
- `d1` = date cunoscute
- `d2` = date obținute
- `C` = matricea de confuzie

Aplicație 2.1. Fie un set de date conținând 18 vectori împărțiți în 3 clase. Se cunoaște faptul că primii 6 vectori sunt din *clasa 1*, următorii 6 vectori sunt din *clasa 2* iar ultimii 6 vectori sunt din *clasa 3*. În urma clasificării (pentru exemplul dat este irelevantă metoda folosită pentru clasificare) vectorii au fost împărțiți astfel:

Indice vector	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Clasă prezisă	1	1	2	1	1	2	2	1	2	2	3	2	3	1	3	3	3	3

Să se calculeze și să se afișeze matricea de confuzie.

```
clase_cunoscute = [1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3];
clase_obtinute  = [1, 1, 2, 1, 1, 2, 2, 1, 2, 2, 3, 2, 3, 1, 3, 3, 3, 3];
% calcul matrice de confuzie
C = confusionmat(clase_cunoscute, clase_obtinute);
% afisare tabelara a matricei de cofuzie
confusionchart(C)
```



Conform matricei de confuzie, se poate observa că:

- 4 elemente din *clasa 1* au fost clasificate corect în *clasa 1*
- 4 elemente din *clasa 2* au fost clasificate corect în *clasa 2*
- 5 elemente din *clasa 3* au fost clasificate corect în *clasa 3*
- 2 elemente din *clasa 1* au fost incorect clasificate în *clasa 2*
- 1 element din *clasa 2* a fost incorect clasificat în *clasa 3*
- 1 element din *clasa 3* a fost incorect clasificat în *clasa 1*

Pentru calcul acurateții se însumează toate elementele de pe diagonala principală (cele clasificate corect) și se împarte rezultatul la numărul total de vectori.

```
% calcul acuratete
acuratete = sum(diag(C))/sum(sum(C))*100;
disp(['Acuratetea clasificarii este = ', num2str(acuratete),' %'])
```

```
Acuratetea clasificarii este = 72.2222%
```

Pentru a calcula scorul de clasificare corectă pentru fiecare clasă, se împarte `C(i,i)` la suma elementelor de pe linia `i`.

```
% scor clasa_1
scor_clasa1 = C(1,1)/sum(C(1,:))*100;
% scor clasa_2
scor_clasa2 = C(2,2)/sum(C(2,:))*100;
% scor clasa_3
scor_clasa3 = C(3,3)/sum(C(3,:))*100;
disp(['scorul de clasificare pentru clasa_1 = ', num2str(scor_clasa1),'%'])
disp(['scorul de clasificare pentru clasa_2 = ', num2str(scor_clasa2),'%'])
disp(['scorul de clasificare pentru clasa_3 = ', num2str(scor_clasa3),'%'])
```

```
scorul de clasificare pentru clasa_1 = 66.6667%
scorul de clasificare pentru clasa_2 = 66.6667%
scorul de clasificare pentru clasa_3 = 83.3333%
```

Capitolul 3

Metode statistice și neurale folosite pentru clasificare

Având un set de date despre care se cunoaște împărțirea în clase și un caz de test (reprezentat de obicei ca un vector de proprietăți), *clasificarea supervizată* răspunde problemei de atribuire a etichetei uneia dintre clase pentru cazul de test. Spre exemplu, o planetă nou descoperită într-un alt sistem planetar este telurică precum Pământul, gigant gazos ca Jupiter, sau un bulgăre de gheață precum corpurile din centura Kuiper? Clasificarea supervizată necesită o etapă de învățare, în care clasificatorul deduce regulile de clasificare pornind de la un set de date pre-etichetat.

3.1.Clasificatorul cel mai apropiat prototip

(en. ***Nearest Prototype*** **- NP)**

3.1.1. Introducere teoretică

Fie K clase notate ω_1, ω_2... ω_k ... ω_K și un eșantion Y de test căruia dorim să-i aflăm apartenența la una dintre aceste clase. Clasificatorul tradiţional al *celui mai apropiat prototip* atribuie un eşantion de test *Y,* clasei ω_k care are cel mai apropiat prototip M_k faţă de Y (unde $k = 1$ *...* K*)*. Uzual, prototipul fiecărei clase este definit ca valoarea medie a eşantioanelor de antrenare ce aparţin clasei respective.
Algoritmul este următorul:

- Pentru fiecare clasă ω_k se calculează media M_k
- Se calculează distanța (de obicei *distanța euclidiană*) între eșantionul de test Y și toate mediile M_k.

Dacă vectorul Y este un vector *n*-dimensional de forma $Y = [y_1 \ y_2 \ y_3 \dots y_n]$ și prototipul clasei k este un vector *n*-dimensional de forma $M_k = [m_{1,}^{(k)} \ m_2^{(k)} \ m_3^{(k)} \dots m_n^{(k)}]$ atunci distanța euclidiană dintre vectorul Y și media M_k va fi:

$$d_k = \sqrt{(y_1 - m_1^{(k)})^2 + (y_2 - m_2^{(k)})^2 + \ \dots \ + (y_n - m_n^{(k)})^2} \text{ ,unde } k = 1 \dots K \quad (3.1)$$

- Eșantionul de test Y va aparține clasei ω_k pentru care s-a obținut distanța d_k minimă.

Exemplu. Fie 2 clase definite astfel:

Clasa ω_1 : $A = \begin{bmatrix}1\\5\end{bmatrix}$, $B = \begin{bmatrix}5\\5\end{bmatrix}$, $\mathrm{C} = \begin{bmatrix}1\\1\end{bmatrix}$, $D = \begin{bmatrix}5\\1\end{bmatrix}$

Clasa ω_2 : $E = \begin{bmatrix}4\\2\end{bmatrix}$, $F = \begin{bmatrix}8\\2\end{bmatrix}$, $\mathrm{G} = \begin{bmatrix}4\\-2\end{bmatrix}$, $H = \begin{bmatrix}8\\-2\end{bmatrix}$

Se dorește clasificarea vectorului $Y = \begin{bmatrix}4\\0\end{bmatrix}$ folosind clasificatorul NP.

Soluție:

Mediile celor 2 clase sunt $M_1 = \begin{bmatrix}3\\3\end{bmatrix}$ și $M_2 = \begin{bmatrix}6\\0\end{bmatrix}$.

- $d^2(Y, M_1) = (4 - 3)^2 + (0 - 3)^2 = 10$
- $d^2(Y, M_2) = (4 - 6)^2 + (0 - 0)^2 = 4$
- $d^2(Y, M_2) < d^2(Y, M_1)$ ➔ Y aparține *clasei* ω_2

3.1.2. Utilizarea clasificatorului NP în MATLAB

Aplicația 3. 1. Folosirea algoritmului NP pentru clasificarea în două clase a cifrelor din baza de date MNIST
Pentru învățare se vor folosi 300 de imagini cu *cifra 3* și 300 de imagini cu *cifra 8*. Vor fi testate 100 imagini cu *cifra 3* și 100 imagini cu *cifra 8* (altele decât cele din lotul de antrenare).

- Se vor afișa ca imagini mediile celor două clase.
- Se vor clasifica imaginile din lotul de test și se va afișa matricea de confuzie.
- Se va calcula acuratețea clasificării.

1. Se creează baza de date (antrenare și testare).

```
database_MNIST = xlsread('mnist_train_5k.xlsx');
% Baza de date folosita este una restransa, folosind doar primele 5000 de imagini.
% Dintre acestea se vor folosi pentru antrenare doar primele 300 de imagini cu cifra 3
si primele 300 de imagini cu cifra 8.
cifreInput = [3, 8];
NrCifre = length(cifreInput);
NrImaginiPerCifraTrain = 300;
NrImaginiPerCifraTest  = 100;
% se cauta in baza de date pozitiile din fisierul excel pe care se afla cifra 3
cifra_index_clasa1 = find((database_MNIST(:,1)==cifreInput(1)));
% se cauta in baza de date pozitiile din fisierul excel pe care se afla cifra 8
cifra_index_clasa2 = find((database_MNIST(:,1)==cifreInput(2)));
% se formeaza vectorul cifre_index ce contine pozitiile din fisierul excel
% pe care se afla cifra 3 si cifra 8
cifre_index_clasa1_train = [cifra_index_clasa1(1:NrImaginiPerCifraTrain)];
cifre_index_clasa2_train = [cifra_index_clasa2(1:NrImaginiPerCifraTrain)];
cifre_index_clasa1_test  =
[cifra_index_clasa1(NrImaginiPerCifraTrain+1:NrImaginiPerCifraTrain+NrImaginiPerCifraT
est)];
cifre_index_clasa2_test  =
[cifra_index_clasa2(NrImaginiPerCifraTrain+1:NrImaginiPerCifraTrain+NrImaginiPerCifraT
est)];
% se salveaza in matricea cifre toate imaginile cu cifrele 3 si 8;
% fiecare imagine este salvata pe cate o linie din matricea cifre;
% pe prima pozitie este cifra iar pe urmatoarele M x N coloane sunt valorile pixelilor
% baza de date ce va fi folosita pentru antrenare nu foloseste si prima coloana
database_clasa1_train = database_MNIST(cifre_index_clasa1_train,2:end)/255;
database_clasa2_train = database_MNIST(cifre_index_clasa2_train,2:end)/255;
database_clasa1_test  = database_MNIST(cifre_index_clasa1_test,2:end)/255;
database_clasa2_test  = database_MNIST(cifre_index_clasa2_test,2:end)/255;
database_test = [database_clasa1_test; database_clasa2_test];
% etichetele pentru imaginile din lotul de test
labels_test  = [ones(1,NrImaginiPerCifraTest), 2*ones(1,NrImaginiPerCifraTest)];
```

2. Se calculează și se afișează mediile celor 2 clase.

```
medie_clasa_1 = mean(database_clasa1_train);
imag_medie_clasa_1 = reshape(medie_clasa_1,28,28)';
figure(1), imshow(imag_medie_clasa_1), title('Media clasei 1')
medie_clasa_2 = mean(database_clasa2_train);
imag_medie_clasa2 = reshape(medie_clasa_2,28,28)';
figure(2), imshow(imag_medie_clasa2), title('Media clasei 2')
```

3. Se clasifică vectorii din *clasa 1*, lotul de test.

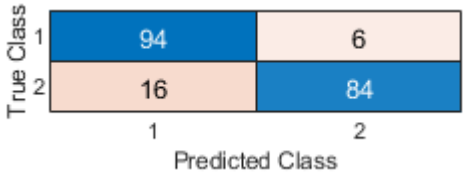
```
for i = 1:NrImaginiPerCifraTest
    dist1 = sum((database_clasa1_test(i,:)-medie_clasa_1).^2);
    dist2 = sum((database_clasa1_test(i,:)-medie_clasa_2).^2);
    [val, poz] = min([dist1, dist2]);
    labels_clasif_clasa_1(i) = poz;
end
```

4. Se clasifică vectorii din *clasa 2*, lotul de test.

```
for i = 1:NrImaginiPerCifraTest
    dist1 = sum((database_clasa2_test(i,:)-medie_clasa_1).^2);
    dist2 = sum((database_clasa2_test(i,:)-medie_clasa_2).^2);
    [val, poz] = min([dist1, dist2]);
    labels_clasif_clasa_2(i) = poz;
end
labels_clasif = [labels_clasif_clasa_1, labels_clasif_clasa_2];
```

5. Se calculează matricea de confuzie.

```
matrice_confuzie = confusionmat(labels_test, labels_clasif);
table(matrice_confuzie)
```



6. Se calculează acuratețea clasificării.

```
acuratete = sum(diag(matrice_confuzie))/sum(sum(matrice_confuzie))*100;
disp(['acuratete=', num2str(acuratete),'%'])
```

```
acuratete = 89%
```

3.2. Clasificatorul cel mai apropiat vecin

(*en*. k-*Nearest Neighbor* k-NN)

3.2.1. Introducere teoretică

Fie *N* eșantioane $X_1, X_2 \dots X_N$ împărțite în C clase $\omega_1, \omega_2 \dots \omega_C$ și un eșantion Y de test căruia dorim să-i aflăm apartenența la una dintre aceste C clase. Algoritmul tradiţional al *celui mai apropiat vecin* (1-NN) alocă un eșantionul de test *Y*, acelei clase ω_c care conține eșantionul X ce se află la distanță minimă de *Y*. Această idee poate fi extinsă la cele mai apropiate *k* eșantioane vecine ale lui *Y*, astfel încat Y este atribuit clasei care este reprezentată printr-o majoritate a celor mai apropiate *k* eșantioane vecine.
Algoritmul este următorul:

- Se calculează distanța d (de obicei Euclidiană) dintre eșantionul de test Y și toate cele N eșantioanele de antrenare (pentru care se cunoaște apartenența la clase). Dacă eșantioanele de antrenare și cel de test sunt vectori *n*-dimensionali, atunci distanța d se va calcula astfel:

$$d_i = \sqrt{(y_1 - x_1^{(i)})^2 + (y_2 - x_2^{(i)})^2 + \dots + (y_n - x_n^{(i)})^2} \text{ ,unde } i = 1 \dots N \tag{3.2}$$

- Eșantionul de test Y va aparține clasei ce conține eșantionul X ce se află la distanță minimă de *Y*.

Exemplu. Fie 2 clase definite astfel:

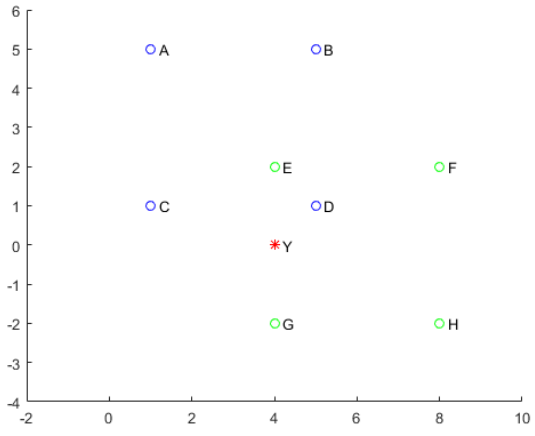
Clasa ω_1 : $A = \begin{bmatrix}1\\5\end{bmatrix}$, $B = \begin{bmatrix}5\\5\end{bmatrix}$, $\mathrm{C} = \begin{bmatrix}1\\1\end{bmatrix}$, $D = \begin{bmatrix}5\\1\end{bmatrix}$

Clasa ω_2 : $E = \begin{bmatrix}4\\2\end{bmatrix}$, $F = \begin{bmatrix}8\\2\end{bmatrix}$, $\mathrm{G} = \begin{bmatrix}4\\-2\end{bmatrix}$, $H = \begin{bmatrix}8\\-2\end{bmatrix}$

Se dorește clasificarea vectorului $Y = \begin{bmatrix}4\\0\end{bmatrix}$ folosind clasificatorii 1-NN și 3-NN.

Clasificare folosind 1-NN

Folosind distanța euclidiană, cel mai apropiat vecin este vectorul *D* care aparține *clasei* ω_1, deci *Y* va aparține *clasei* ω_1.



Clasificare folosind 3-NN

Dacă alegem $k = 3$ eșantioane, folosind distanța euclidiană, rezultă că cei mai apropiați 3 vectori față de Y sunt vectorii *E, D* și *G*. Dar știm că vectorii *E* și *G* aparțin *clasei* ω_2, deci *Y* va aparține *clasei* ω_2.

3.2.2. Utilizarea clasificatorului k-NN în MATLAB

Pentru a realiza clasificarea în MATLAB folosind clasificatorul k-NN se poate folosi funcția `knnsearch`.

Sintaxă: `Ind = knnsearch(X,Y,'dist','val_dist','k',val_k)` unde:

- `X` = matricea vectorilor de antrenare, așezați pe linii (despre care se cunoaște apartenența la clase)
- `Y` = matricea vectorilor de test, așezați pe linii
- `dist` = distanța utilizată: `'euclidean'`, `'cityblock'` etc. Dacă nu se specifică, *default* distanța folosită este cea euclidiană
- `k` = numărul de vecini căutați. Dacă nu se specifică, *default* `k` = 1
- `Ind` = vectorul coloană (în cazul în care `k` = 1) al indicilor rezultați în urma clasificării. `Ind(i)` conține indexul celui mai apropiat vector din matricea `X` față de `Y(i,:)`.

Obs: pentru sintaxa completă >> `help knnsearch`

Aplicația 3.2. Fie 2 clase definite astfel:

Clasa C_1 : $A = \begin{bmatrix}1\\5\end{bmatrix}$, $B = \begin{bmatrix}5\\5\end{bmatrix}$, $C = \begin{bmatrix}1\\1\end{bmatrix}$, $D = \begin{bmatrix}5\\1\end{bmatrix}$

Clasa C_2 : $E = \begin{bmatrix}4\\2\end{bmatrix}$, $F = \begin{bmatrix}8\\2\end{bmatrix}$, $G = \begin{bmatrix}4\\-2\end{bmatrix}$, $H = \begin{bmatrix}8\\-2\end{bmatrix}$

Se dorește clasificarea în MATLAB a vectorului $Y = \begin{bmatrix}4\\0\end{bmatrix}$ folosind :

- clasificatorul *1-NN*
- clasificatorul 3-*NN*

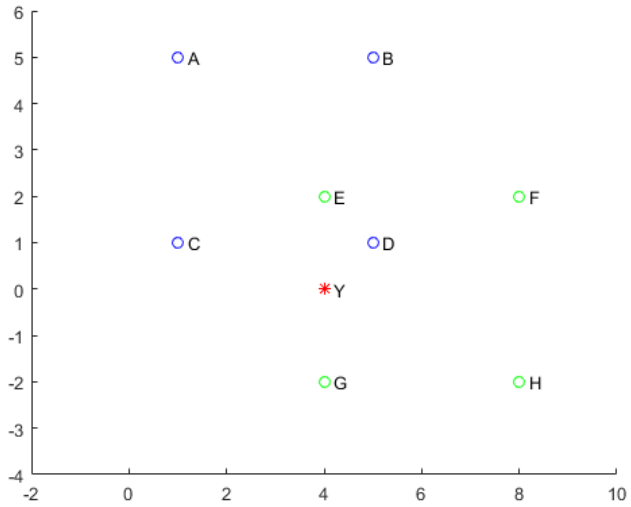
Pentru calculul distanței se va folosi *distanța euclidiană.*

1. Stabilirea setului de date.

```
% vectori antrenare
A = [1, 5]'; B = [5, 5]'; C = [1, 1]';  D = [5, 1]';
E = [4, 2]'; F = [8, 2]'; G = [4, -2]'; H = [8, -2]';
% vector test
Y = [4, 0]';
% lot antrenare
train = [A, B, C, D, E, F, G, H]';
N = 2; % N = numar de clase
% etichete lot antrenare (aparteneta la clase)
eticheta = [1, 1, 1, 1, 2, 2, 2, 2]';
```

2. Reprezentare grafică lot antrenare și vector de test.

```
% reprezentare grafica lot antrenare si vector test
figure(1)
hold on
plot(train(1,1:4), train(2,1:4),'ob')
plot(train(1,5:8), train(2,5:8),'og')
plot(4, 0,'*r')
hold off
text(1.2,1,'C'), text(1.2,5,'A'), text(5.2,1,'D'), text(5.2,5,'B')
text(4.2,2,'E'), text(8.2,2,'F'), text(4.2,-2,'G'), text(8.2,-2,'H')
text(4.2,0,'Y')
axis([-2 10 -4 6])
```



3. Clasificare vector Y folosind clasificatorul 1-NN

```
Ind = knnsearch(train,Y');
disp(['cel mai apropiat vecin al lui Y este in mat. X pe pozitia: ', num2str(Ind)])
disp(['vectorul Y a fost clasificat cu 1-NN in clasa: ', num2str(eticheta(Ind))])
```

```
cel mai apropiat vecin al lui Y se gaseste in matricea X pe pozitia: 4
vectorul Y a fost clasificat cu 1-NN in clasa: 1
```

4. Clasificare vector Y folosind clasificatorul 3-NN

```
K = 3; % K = numar de vecini cautati
Ind = knnsearch(train,Y','k',K);
disp(['cei mai apropiati 3 vecini ai lui Y sunt in mat. X pe pozitiile: ',
num2str(Ind)])
clasa_majoritara = max(hist(eticheta(Ind),N));
disp(['vectorul Y a fost clasificat cu 3-NN in clasa: ', num2str(clasa_majoritara)])
```

```
cei mai apropiati 3 vecini ai lui Y se gasesc in matricea X pe pozitiile: 4  5  7
vectorul Y a fost clasificat cu 3-NN in clasa: 2
```

Aplicația 3.3. Folosirea algoritmului 1-NN pentru clasificarea în două clase a cifrelor din baza de date MNIST

Pentru învățare se vor folosi 300 de imagini cu *cifra 3* și 300 de imagini cu *cifra 8*. Vor fi testate 100 imagini cu *cifra 3* și 100 imagini cu *cifra 8* (altele decât cele din lotul de antrenare).

- Se vor clasifica imaginile din lotul de test și se va afișa matricea de confuzie.
- Se va calcula acuratețea clasificării.
- Se vor afișa imaginile incorect clasificate.

1. Se creează baza de date (antrenare și testare).

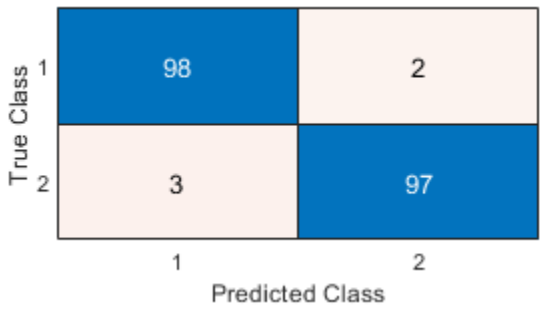
```
database_MNIST = xlsread('mnist_train_5k.xlsx');
% Baza de date folosita este una restransa, folosind doar primele 5000 de imagini.
% Dintre acestea se vor folosi pentru antrenare doar primele 300 de imagini cu cifra 3
si primele 300 de imagini cu cifra 8.
cifreInput = [3, 8];
NrCifre = length(cifreInput);
NrImaginiPerCifraTrain = 300;
NrImaginiPerCifraTest  = 100;
% se cauta in baza de date pozitiile din fisierul excel pe care se afla cifra 3
cifra_index1 = find((database_MNIST(:,1)==cifreInput(1)));
% se cauta in baza de date pozitiile din fisierul excel pe care se afla cifra 8
cifra_index2 = find((database_MNIST(:,1)==cifreInput(2)));
% se formeaza vectorul cifre_index ce contine pozitiile din fisierul excel
% pe care se afla cifra 3 si cifra 8
cifre_index_train = [cifra_index1(1:NrImaginiPerCifraTrain);
cifra_index2(1:NrImaginiPerCifraTrain)];
cifre_index_test  =
[cifra_index1(NrImaginiPerCifraTrain+1:NrImaginiPerCifraTrain+NrImaginiPerCifraTest);
cifra_index2(NrImaginiPerCifraTrain+1:NrImaginiPerCifraTrain+NrImaginiPerCifraTest)];
% se salveaza in matricea cifre toate imaginile cu cifrele 3 si 8;
% fiecare imagine este salvata pe cate o linie din matricea cifre;
% pe prima pozitie este cifra, iar pe urmatoarele M x N coloane (784) sunt valorile
pixelilor
% baza de date ce va fi folosita pentru antrenare nu foloseste si prima coloana
database_train = database_MNIST(cifre_index_train,2:end);
database_test = database_MNIST(cifre_index_test,2:end);
% etichetele pentru imaginile din lotul de test
labels_test  = [ones(1,NrImaginiPerCifraTest), 2*ones(1,NrImaginiPerCifraTest)];
```

2. Se clasifică fiecare imagine din lotul de test folosind clasificatorul 1-NN.

```
for i = 1:size(database_test,1)
    clasif(i) = knnsearch(database_train, database_test(i,:));
    if (clasif(i) < NrImaginiPerCifraTrain)
        labels_clasif(i) = 1;
    else
        labels_clasif(i) = 2;
    end
end
```

3. Se calculează matricea de confuzie.

```
matrice_confuzie = confusionmat(labels_test, labels_clasif);
table(matrice_confuzie)
```



4. Se calculează acuratețea clasificării.

```
acuratete = sum(diag(matrice_confuzie))/sum(sum(matrice_confuzie))*100;
disp(['acuratete = ', num2str(acuratete),'%'])
```

```
acuratete = 97.5%
```

5. Se afișează imaginile clasificate greșit.

```
index_imagini_gresite = find((labels_test-labels_clasif)~=0);
for i = 1 : length(index_imagini_gresite)
    imag = reshape(database_test(index_imagini_gresite(i),:),28,28)';
    figure(),imshow(imag);
end
```

3.3. Clasificare folosind funcţii discriminant de tip *Bayes*

3.3.1. Introducere teoretică

Proiectarea unui clasificator presupune calcularea explicită a unui set de *M* funcţii discriminant şi selectarea clasei care corespunde maximului $\underset{k}{MAX}(g_k)$:

$$g_k : R^n \rightarrow R,\ 1 \le k \le M, \text{ astfel că } g_i(X) > g_j(X), \text{ pentru } \forall\ i \ne j \qquad (3.3)$$

Fie forma *X* de intrare din spaţiul R^n cu densităţi condiţionate normale de forma:

$$f(X, \omega_i) \rightarrow N(\mu_i, \Sigma_i),\ 1 \le i \le M$$

Pentru un clasificator Bayes cu eroare minimă de clasificare pentru *M* clase, cu funcţiile discriminant $g_k(x) = \ln P(\omega_k) + \ln f(x \mid \omega_k)$, se obţine pentru vectorii repartizaţi normal relaţia:

$$g_k(X) = -\frac{1}{2}(X-\mu_k)^T\Sigma_k^{-1}(X-\mu_k) - \frac{n}{2}\ln(2\pi) - \frac{1}{2}\ln(det\Sigma_k) + \ln(P(\omega_k)) \qquad (3.4)$$

selectarea clasei corespunzând lui $\underset{k}{MAX}(g_k)$.

Exemplu de clasificare folosind clasificatorul Bayes pentru 2 clase

Fie două clase de semnale bidimensionale repartizate normal având $P(\omega_1) = P(\omega_2)$.

Clasa ω_1: $A = \begin{bmatrix}2\\0\end{bmatrix}$, $B = \begin{bmatrix}3\\1\end{bmatrix}$, $C = \begin{bmatrix}4\\0\end{bmatrix}$, $D = \begin{bmatrix}3\\-1\end{bmatrix}$

Clasa ω_2: $E = \begin{bmatrix}-2\\0\end{bmatrix}$, $F = \begin{bmatrix}-3\\1\end{bmatrix}$, $G = \begin{bmatrix}-4\\0\end{bmatrix}$, $H = \begin{bmatrix}-3\\-1\end{bmatrix}$

Se doreşte să se găsească:

- ecuaţia suprafeţei de separate
- regula de decizie Bayes
- să se clasifice vectorul V = $\begin{bmatrix}-2\\1\end{bmatrix}$

Soluţie:

- Se calculează mediile celor 2 clase:

$$\mu_1 = \frac{1}{N_1}\sum_{i=1}^{N_1} X_i = \frac{1}{4}(A+B+C+D) = \begin{bmatrix}3\\0\end{bmatrix}$$

$$\mu_2 = \frac{1}{N_2}\sum_{i=1}^{N_2} X_i = \frac{1}{4}(E+F+G+H) = \begin{bmatrix}-3\\0\end{bmatrix}$$

- Se calculează matricele de covariaţie pentru cele 2 clase:

$$\Sigma_1 = \frac{1}{N_1}\sum_{i=1}^{N_1}(X_i-\mu_1)(X_i-\mu_1)^T = \frac{1}{2}I_2$$

$$\Sigma_2 = \frac{1}{N_2}\sum_{i=1}^{N_2}(X_i - \mu_2)(X_i - \mu_2)^T = \frac{1}{2}I_2$$

- Ecuaţia suprafeţei de separaţie este:

$$(X-\mu_1)^T \Sigma_1^{-1}\cdot(X-\mu_1) - (X-\mu_2)^T \cdot\Sigma_2^{-1}\cdot(X-\mu_2) + \ln\frac{\det\Sigma_1}{\det\Sigma_2} = 2\ln\frac{P(\omega_1)}{P(\omega_2)}$$

Ştim însă că: $\Sigma_1 = \Sigma_2 = \frac{1}{2}I_2 \rightarrow \ln\frac{\det\Sigma_1}{\det\Sigma_2} = 0,\ P(\omega_1) = P(\omega_2) \rightarrow\ 2\ln\frac{P(\omega_1)}{P(\omega_2)} = 0$

Ecuaţia suprafeţei de separaţie se va scrie:

$$(X-\mu_1)^T \Sigma_1^{-1}\cdot(X-\mu_1) - (X-\mu_2)^T \cdot\Sigma_2^{-1}\cdot(X-\mu_2) = 0, \text{ unde}$$

$$X = \begin{bmatrix}x_1\\x_2\end{bmatrix},\ \Sigma_1^{-1} = \begin{bmatrix}2 & 0\\0 & 2\end{bmatrix} = 2I_2 \text{ şi } \Sigma_2^{-1} = \begin{bmatrix}2 & 0\\0 & 2\end{bmatrix} = 2I_2$$

Rezultă:

$$(x_1 - 3 \quad x_2)2I_2\begin{pmatrix}x_1-3\\x_2\end{pmatrix} - (x_1+3 \quad x_2)2I_2\begin{pmatrix}x_1+3\\x_2\end{pmatrix} = 0$$

Rezultă ecuaţia suprafeţei de separaţie **$-12x_1 = 0$ ➔ $x_1 = 0$**.

Regula de decizie va fi **$-12x_1 \lessgtr 0 \Rightarrow x_1 \gtrless 0$** $\mathbf{X} \in \{\begin{matrix}\omega_1\\\omega_2\end{matrix}$

Dacă se doreşte clasificarea vectorului V = $\begin{bmatrix}-2\\1\end{bmatrix}$, atunci $x_1 = -2 < 0$ ➔ $V \in \omega_2$

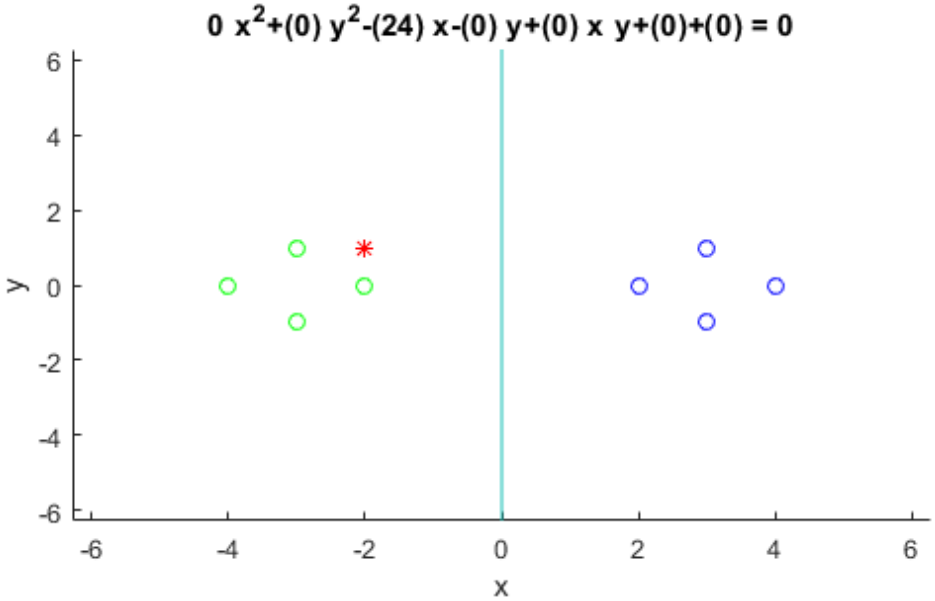


Figura 3.1. Reprezentarea grafică a punctelor din cele două clase, a punctului de test (cu *) şi a suprafeţei de separaţie

3.3.2. Utilizarea clasificatorului *Bayes* în MATLAB

Aplicația 3.4. Se dorește găsirea suprafeței de separație care să separe cât mai bine cele două clase generate astfel:

```
% lot de antrenare
nrElementeClasa = 100;
% clasa 1
M = [0,0];
R1min = 1; R1max = 2;
raze1 = unifrnd(R1min,R1max,nrElementeClasa,1);
theta = unifrnd(0,2*pi,nrElementeClasa,1);
clasa1 = [raze1.*cos(theta),raze1.*sin(theta)];
% clasa 2
R2min = 3; R2max = 4;
raze2 = unifrnd(R2min,R2max,nrElementeClasa,1);
theta = unifrnd(0,2*pi,nrElementeClasa,1);
clasa2 = [raze2.*cos(theta),raze2.*sin(theta)];
% reprezentarea grafica a lotului de antrenare
figure(1)
hold on
plot(clasa1(:,1),clasa1(:,2),'.b')
plot(clasa2(:,1),clasa2(:,2),'.g')
hold off
axis([-5 5 -5 5])
legend('clasa_1','clasa_2')
```

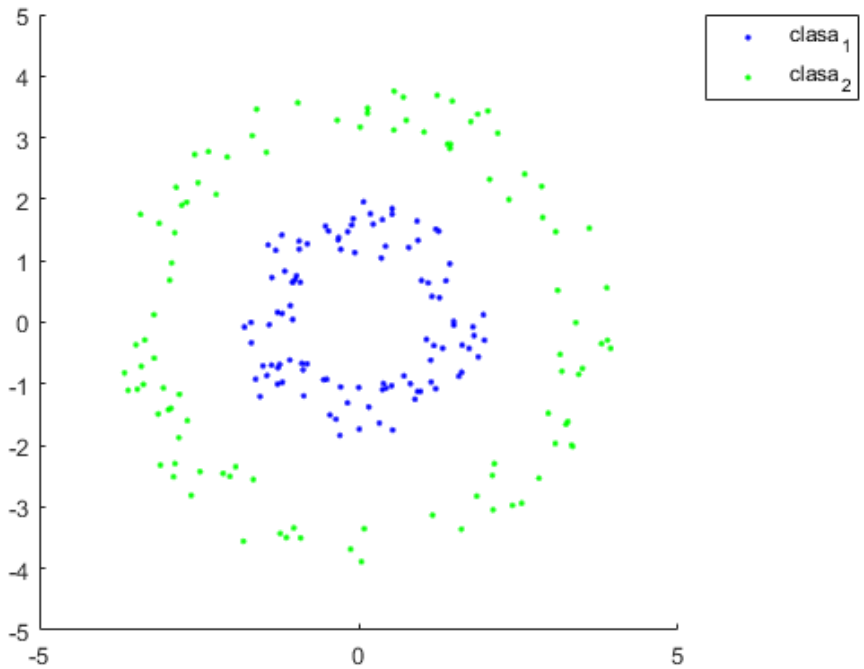


Figura 3.2. Lotul de antrenare

Pentru a determina suprafața de separație dintre cele două clase s-a implementat o funcție care să primească la intrare cele două clase (cu vectorii scriși pe linii) și să scoată ca parametru de ieșire ecuația suprafeței de separație.

```
function ec_suprafata = bayes_ecuatie(clasa1, clasa2)
% Se implementeaza analitic relatia:
% g1(X) = (X-media1)'*sigma1^(-1)*(X-media1)+ln(det(sigma1))
% unde X este de forma [x1 x2]
media1 = sum(clasa1,1)/length(clasa1(:,1));
sigma1 = cov(clasa1,1); inv_sigma1 = inv(sigma1);
m1 = media1(1); m2 = media1(2);
a11 = inv_sigma1(1, 1); a12 = inv_sigma1(1, 2);
a21 = inv_sigma1(2, 1); a22 = inv_sigma1(2, 2);
% definire coeficienti
coef1_clasa1 = (a11);
coef2_clasa1 = (a22);
coef3_clasa1 = (2*m1*a11 + m2*a21 + m2*a12);
coef4_clasa1 = (m1*a21 + m1*a12 + 2*m2*a22);
coef5_clasa1 = (a21 + a12);
coef6_clasa1 = (a11*m1^2 + a22*m2^2 + a21*m1*m2 + a12*m1*m2);
log_1 = (log(det(sigma1)));
% Se implementeaza analitic relatia:
% g2(X) = (X-media2)'*sigma2^(-1)*(X-media2)+ln(det(sigma2))
media2 = sum(clasa2,1)/length(clasa2(:,1));
sigma2 = cov(clasa2,1); inv_sigma2 = inv(sigma2);
m1 = media2(1); m2 = media2(2);
a11 = inv_sigma2(1, 1); a12 = inv_sigma2(1, 2);
a21 = inv_sigma2(2, 1); a22 = inv_sigma2(2, 2);
% definire coeficienti
coef1_clasa2 = (a11);
coef2_clasa2 = (a22);
coef3_clasa2 = (2*m1*a11 + m2*a21 + m2*a12);
coef4_clasa2 = (m1*a21 + m1*a12 + 2*m2*a22);
coef5_clasa2 = (a21 + a12);
coef6_clasa2 = (a11*m1^2 + a22*m2^2 + a21*m1*m2 + a12*m1*m2);
log_2 = (log(det(sigma2)));
coef1 = num2str( coef1_clasa1 - coef1_clasa2 );
coef2 = num2str( coef2_clasa1 - coef2_clasa2 );
coef3 = num2str( coef3_clasa1 - coef3_clasa2 );
coef4 = num2str( coef4_clasa1 - coef4_clasa2 );
coef5 = num2str( coef5_clasa1 - coef5_clasa2 );
coef6 = num2str( coef6_clasa1 - coef6_clasa2 );
coef_log = num2str( log_1 - log_2 );
% suprafata de separatie
ec_suprafata = strcat(coef1,'*x1^2','+(',coef2,')','*x2^2','-(',coef3,')','*x1','-
(',coef4,')','*x2','+(',coef5,')','*x1*x2','+(',coef6,')','+(',coef_log,')');
```

Folosind funcția `bayes_ecuatie` pentru cele două clase generate anterior, se obține următoarea suprafață de separație:

```
% suprafata de separatie colosind Bayes
ec_suprafata = bayes_ecuatie(clasa1, clasa2);
% reprezentarea grafica a suprafetei de separatie
figure(2)
hold on
plot(clasa1(:,1),clasa1(:,2),'.b')
plot(clasa2(:,1),clasa2(:,2),'.g')
ezplot(ec_suprafata)
hold off
axis([-5 5 -5 5])
legend('clasa_1','clasa_2','suprafata_separatie')
```

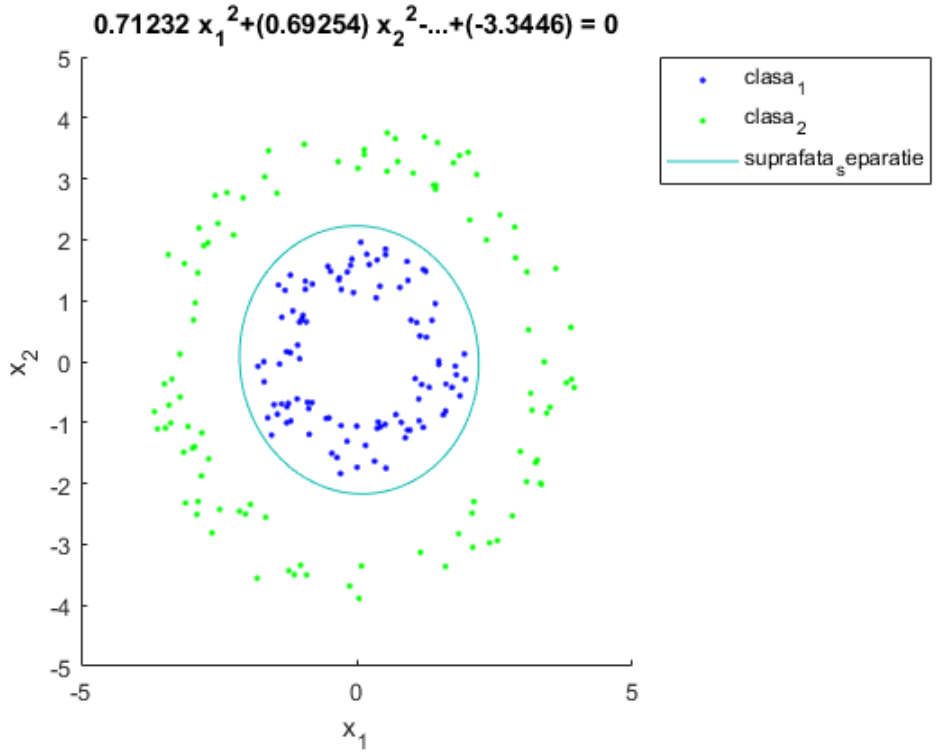


Figura 3.3. Suprafața de separație

Ecuația suprafeței de separație obținută este:

```
ec_suprafata =
    '0.71232*x1^2+(0.69254)*x2^2-(0.063175)*x1-
(0.040248)*x2+(0.036426)*x1*x2+(-0.00042915)+(-3.3446)'
```

Folosind suprafața de separație obținută se va clasifica un nou set de date de test, format din 10 vectori.

```
% lot de test
nrElementeTest = 10;
m = [0,0];dispersie = 2;
Y = [normrnd(m(1), dispersie,[1,nrElementeTest]);
     normrnd(m(2), dispersie,[1,nrElementeTest])]';
figure(3)
hold on
plot(clasa1(:,1),clasa1(:,2),'.b')
plot(clasa2(:,1),clasa2(:,2),'.g')
plot(Y(:,1),Y(:,2),'or')
ezplot(ec_suprafata)
hold off
axis([-5 5 -5 5])
legend('clasa_1','clasa_2','lot_test_','suprafata_separatie')
```

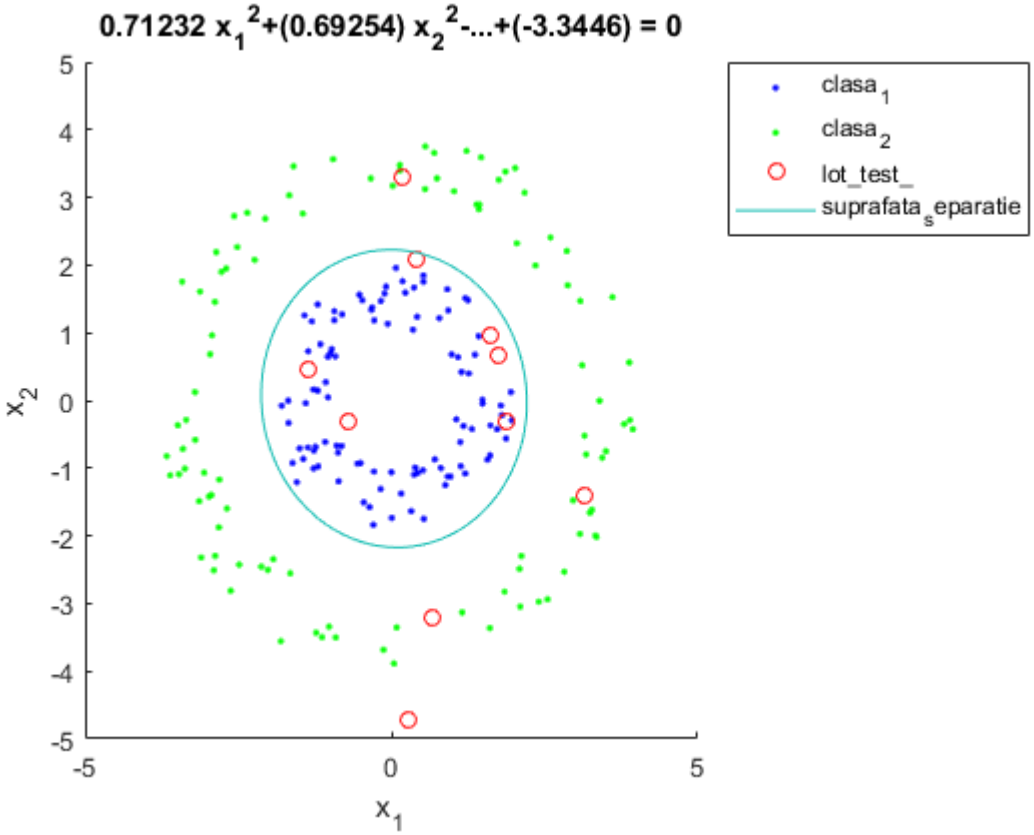


Figura 3.4. Lotul de test

Folosind clasificarea cu funcții discriminant de tip Bayes se obține următoare clasificare a punctelor de test.

```
% clasificare elemente din lotul de test
figure(4)
hold on
plot(clasa1(:,1),clasa1(:,2),'.b')
plot(clasa2(:,1),clasa2(:,2),'.g')
for i = 1:nrElementeTest
    x1 = Y(i,1);
    x2 = Y(i,2);
    clasificare = eval(ec_suprafata);
    if (clasificare < 0)
        clasa(i) = 1;
        plot(Y(i,1),Y(i,2),'o','MarkerEdgeColor','r','MarkerFaceColor','b')
    else
        clasa(i) = 2;
        plot(Y(i,1),Y(i,2),'o','MarkerEdgeColor','r','MarkerFaceColor','g')
    end
end
ezplot(ec_suprafata)
hold off
axis([-5 5 -5 5])
```

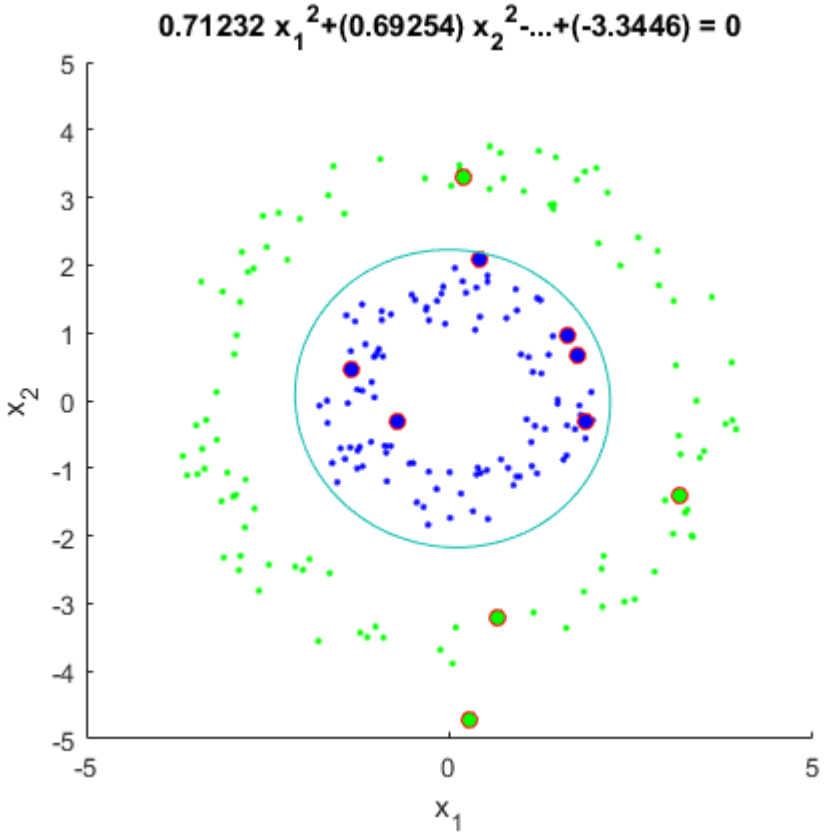


Figura 3.5. Clasificarea elementelor din lotul de test
(cu albastru în clasa 1 și cu verde în clasa 2)

3.4. Rețeaua perceptron multistrat (MLP)

3.4.1. Introducere teoretică

Neuronul biologic. În biologie, neuronul este o celulă adaptată la recepționarea și transmiterea informației. Transmiterea de informații (impulsuri electrice) dintre doi neuroni se face printr-o sinapsă, creierul uman conținând aproximativ 10^{15} neuroni interconectați printr-o rețea foarte densă de interconexiuni sinaptice (aproximativ 10^4 sinapse per neuron).

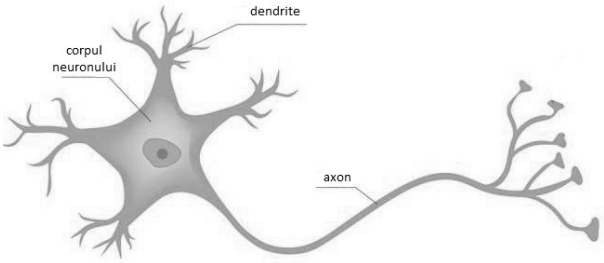


Figura 3.6. Neuronul biologic

Componentele neuronului biologic sunt:

- **corpul neuronului**, în care se procesează informația recepționată (impulsurile electrice recepționate) de pe intrări (dendrite).
- **dendritele** care formează o structură arborescentă foarte densă, prin care se recepționează informații (impulsuri electrice) de la neuronii vecini.
- **axonul**, o fibră cilindrică lungă prin care se transmite informația de ieșire din neuron (impulsuri electrice) către neuronii cu care acesta este interconectat (următorii neuroni din rețea).

Impulsurile de intrare în neuron (recepționate prin intermediul dendritelor) sunt agregate de-a lungul unei perioade de timp denumită perioadă latentă de însumare. Astfel se obține potențialul total la nivelul membranei celulei. Dacă potențialul total depășește un anumit nivel, neuronul se activează și emite un impuls pe ieșire (pe axon).

Clasificarea formelor (*pattern classification*) este una dintre cele mai utile sarcini pe care o poate rezolva o rețea neurală. Obiectivul clasificării este de a asigna unei forme (obiect fizic, eveniment sau fenomen) o clasă. În cele ce urmează vom considera că la intrarea clasificatorului se aplică vectori (X), n-dimensionali, care descriu formele ce trebuie clasificate.

Perceptronul sau neuronul artificial. Este un model computațional pentru neuronul biologic. Fiecare neuron primește impulsuri electrice de la neuronii alăturați prin dendrite și produce un semnal de ieșire în axon. In mod similar, perceptronul primește valorile de intrare (x_i) și realizează o sumă ponderată a acestora (*net*), unde ponderile corespund în modelul biologic cu tăria sinapselor, iar suma ponderată (*net*) corespunde cu potențialul membranei.

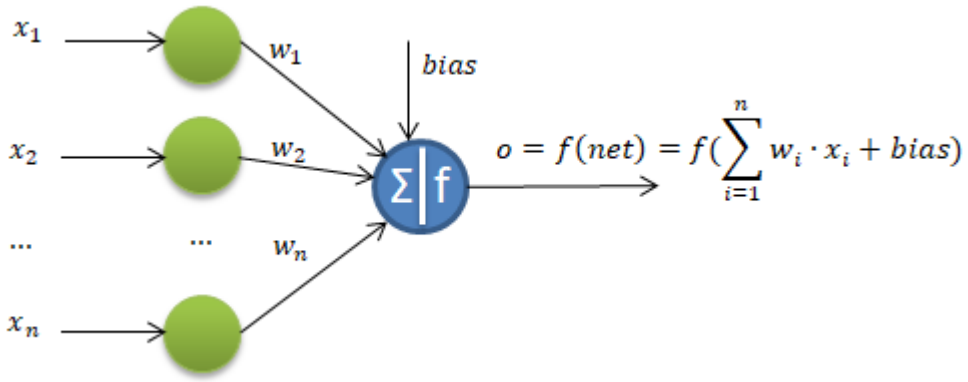


Figura 3.7. Modelul percepronului

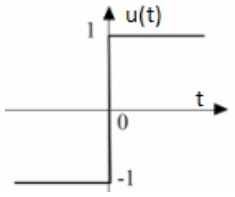
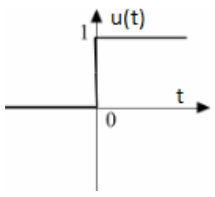
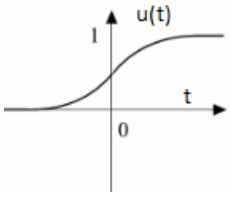
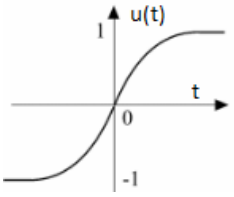
Relația care caracterizează perceptronul este:

$$o = f(net) = f(\sum_{i=1}^{n} w_i \cdot x_i + bias), \text{ unde:} \quad (3.5)$$

- $\boldsymbol{x_i}$ reprezintă valorile de intrare,
- $\boldsymbol{w_i}$ sunt ponderile,
- ***net*** este suma ponderată a intrărilor,
- ***f*** este funcția de activare,
- ***b*** este *bias* și are rolul de termen liber al sumei.

Funcția *f* care generează ieșirea perceptronului din suma ponderată a intrărilor (*net*) este denumită *funcție de activare*. În tabelul de mai jos sunt descrise cele mai întâlnite funcții de activare pentru rețeaua perceptron.

Tabel 3.1. Exemple de funcții de activare

			
treapta bipolară	*treapta unipolară*	funcția sigmoidală	funcția tangentă hiperbolică
$\boldsymbol{u(t) = \begin{cases} 1, t \geq 0 \\ -1, t < 0 \end{cases}}$	$u(t) = \begin{cases} 1, t \geq 0 \\ 0, t < 0 \end{cases}$	$u(t) = \frac{1}{1 + e^{-\lambda t}}$	$u(t) = \frac{e^t - e^{-t}}{e^t + e^{-t}}$

Reguli de învățare a rețelelor neurale

Regulile de învățare sunt metode de ajustare a ponderilor (și *bias*-urilor) prin care valorile calculate la ieșirea rețelei devin, în medie, mai apropiate de ieșirile dorite. Această ajustare se face prin minimizarea unei funcții de cost (de exemplu *eroarea pătratică medie* - `mse`).

Istoric, au fost propuse mai multe reguli de învățare, cele mai importante fiind:

- **Regula lui Hebb** (*Hebbian Learning Rule*) – Hebb 1949 [Hebb1949]
- **Regula Perceptronului** (*Perceptron Learning Rule*) – Rosenblatt 1958 [Rosenblatt1958]
- **Regula delta** (*Delta Learning Rule*) – Rumelhart, D. E., Hinton, G. E., & McClelland 1986 [Rumelhart1986]
- **Regula Widrow-Hoff** (*Widrow-Hoff Learning Rule*) – Widrow și Hoff 1962 **[**Widrow1960**] [**Widrow1962**]**
- **Regula corelației** (*Correlation Learning Rule*)
- **Regula outstar** (*Outstar Learning Rule*) – Grossberg 1969 [Grossberg1969]

Regula lui Hebb

Regula lui Hebb a fost dezvoltată pentru rețele nesupervizate și pornește de la premisa că dacă doi neuroni adiacenți sunt activați sau dezactivați în același timp, atunci tăria conexiunii ar trebui să crească, în timp ce dacă doar unul dintre ei este activat, atunci tăria conexiunii ar trebui să scadă.

Matematic, această rezultă în formula $\Delta w_{ij} = \eta \cdot o_i \cdot x_j$ prin faptul că ieșirea o_i și intrarea x_j cu același semn produc o modificare pozitivă a ponderii, pe când dacă au semne contrare produc o modificare negativă a ponderii; η este constanta/rata de învățare.

De la introducere, regula lui Hebb a evoluat în diferite variante. În anumite variante regula este ușor modificată pentru a contracara creșterea excesivă a ponderilor, care se produce atunci când intrarea (x_j) și ieșirea (o_i) sunt frecvent de același semn. O asemenea variantă este denumită *învățare hebbiană cu saturație* (când tind să crească excesiv, ponderile se saturează la un nivel prestabilit) [Zurada].

Regula Perceptronului

Regula perceptronului este importantă istoric, fiind asociată cu perceptronul descris de Rosenblatt (cu funcție de activare treaptă bipolară). Fiind o regulă de ajustare supervizată, are nevoie atât de ieșirile obținute cât și de ieșirile dorite. Conform acestei reguli:

$$\Delta w_{ij} = \eta \cdot (d_i - o_i) \cdot x_j$$

Ținând cont că ieșirile o_i nu pot avea decât valorile +/-1, ponderea fie nu se modifică, fie se reduce la:

$$\Delta w_{ij} = \pm 2 \cdot \eta \cdot x_j \tag{3.6}$$

În cazul acestei reguli, ajustarea ponderilor se face numai dacă răspunsul real (o_i) este incorect [Zurada].

Regula delta

Regula delta este o regulă care minimizează eroarea pătratică medie prin metoda scăderii de gradient și stă la baza majorității algoritmilor de *backpropagation*.

$$\Delta w_{ij} = \eta \cdot (d_i - o_i) \cdot f'(net_i) \cdot x_j \tag{3.7}$$

Regula delta reprezintă o generalizare a *regulii perceptronului* pentru neuronul cu funcție de activare continuă și derivabilă și este unul dintre motivele pentru care funcții de activare cu derivate ușor calculabile (precum tangenta hiperbolică) au fost adesea preferate în rețelele MLP.

Regula Widrow-Hoff

Regula Widrow-Hoff este o regulă supervizată de ajustare a ponderilor, independentă de funcțiile de activare ale neuronilor, deoarece minimizează eroarea pătratică dintre ieșirea dorită (*d*) și valoarea de activare a neuronului (*net*) [Zurada].

$$\Delta w_{ij} = \eta \cdot (d_i - net_i) \cdot x_j \tag{3.8}$$

Această regulă poate fi văzută ca un caz particular al *regulii delta* pentru funcții de activare liniare.

Regula corelației

Regula corelației se bazează pe principii similare *regulii lui Hebb*, cu diferența că este o regulă de ajustare supervizată care folosește răspunsul dorit în locul ieșirii obținute pentru calculul ajustării.

$$\Delta w_{ij} = \eta \cdot d_i \cdot x_j \tag{3.9}$$

unde *d* = ieșirea dorită, η este constanta/rata de învățare și *x* este valoarea de intrare.

Această regulă se aplică de obicei pentru înregistrarea datelor în rețelele de memorare realizate cu neuroni discreți [Zurada].

Regula outstar

Pentru această regulă, ajustarea ponderilor se face similar rețelelor cu autoorganizare (Kohonen).

$$\Delta w_{ij} = \eta \cdot (d_i - w_{ij}) \tag{3.10}$$

unde *d* = ieșirea dorită și η este constanta/rata de învățare.

Forma vectorială a relației ce caracterizează percepronul

Pentru o scriere mai compactă și o implementare software mai rapidă, se preferă scrierea vectorială a *relației 3.5* astfel:

$$o = f(W^t X + bias) \tag{3.11}$$

Unde W reprezintă vectorul de ponderi iar X este vectorul de intrare.

$$W = \begin{bmatrix} w_1 \\ w_2 \\ \dots \\ w_n \end{bmatrix} \text{ și } X = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$

Pentru a scăpa de termenul liber al sumei (parametrul *bias*) se folosesc vectorii augmentați astfel:

$$W = [w_0 = bias, w_1, w_2, \dots, w_n]^T \text{ și } X = [x_0 = 1, x_1, x_2, \dots, x_n]^T$$

Ecuația 3.5 devine astfel:

$$o = f(W^t \cdot X) \tag{3.12}$$

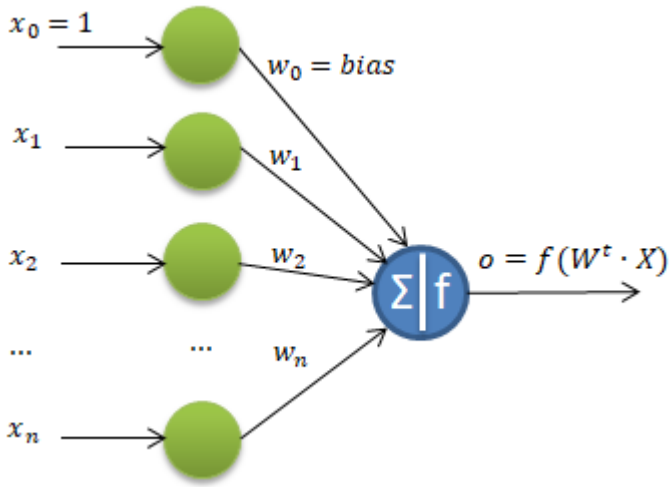


Figura 3.8. Modelul percepronului discret, scriere vectorială

Orice rețea neurală de tip perceptron are nevoie să fie antrenată pentru a putea învăța din exemplele date. Principalii pași ai algoritmului de învățare sunt:

- inițializare parametri rețea
- actualizare ponderi până se respectă condiția de oprire

Inițializare ponderi rețea

Inițializarea ponderilor se face cu valori aleatoare mici, urmând ca aceste valori să fie ajustate în urma antrenării. Valorile trebuie să fie mici (de același ordin de mărime) pentru ca niciuna dintre ponderi să nu domine ieșirile rețelei.

Antrenare rețea (actualizare ponderi)

In timpul antrenării, în interiorul unei epoci, modificarea ponderilor poate fi făcută după prezentarea fiecărui vector de antrenare sau după prezentarea tuturor vectorilor de antrenare (*batch training*). Din punct de vedere computațional, *batch training* este mai eficient, deoarece ieșirile tuturor vectorilor pot fi calculate în același timp și toți vectorii de antrenare sunt tratați identic (spre deosebire de modificarea ponderilor după fiecare vector de antrenare, când vectorii sunt tratați preferențial).

Există mai multe reguli de învățare (moduri de ajustare a ponderilor): *Hebbian learning rule*, *Rosenblatt (perceptron learning rule), delta learning rule*, *Widrow-Hoff learning rule*, *correlation learning rule* etc descrise anterior.

Una dintre cele mai simple reguli de învățare este *regula percepronului* care pentru o rețea fără straturi ascunse și cu un singur neuron pe stratul de ieșire, pentru care se folosește scrierea vectorială, este definită astfel:

$$W(k+1) = W(k) + c \cdot [d_p - o_p(k)] \cdot X_p \tag{3.13}$$

unde:

- $W(k)$ reprezintă setul de ponderi ai rețelei la iterația k
- $W(k+1)$ reprezintă setul de ponderi ai rețelei la iterația $k+1$
- o_p reprezintă ieșirea obținută de rețea pentru forma p de intrare
- d_p reprezintă ieșirea dorită pentru forma p de intrare
- c este rata de învățare. O rată de învățare mică înseamnă că ponderile se modifică relativ puțin, astfel că ajungerea la ponderile optime poate dura un număr mai mare de epoci. Pe de altă parte, o rată de învățare mare poate introduce instabilitate în ajustarea ponderilor, făcând ca ponderile să nu se poată stabiliza pe valoarea optimă
- X_p reprezintă vectorul p de la intrarea rețelei

Oprire antrenare

Criteriile de oprire a antrenării sunt condiții în care se consideră că antrenarea nu mai are efect, nu mai poate fi îmbunătățită sau nu mai merită să fie continuată. Exemple de criterii de oprire sunt:

- eroarea dintre ieșirile obținute și cele dorite este suficient de mică. Eroarea se poate calcula în multe moduri, cea mai întâlnită fiind eroarea pătratică medie. Eroarea este calculată pentru setul de validare (NU pentru setul de antrenare)!
- valorile ponderilor nu se mai modifică semnificativ
- a fost atins numărul maxim de epoci alocate antrenării. O epocă reprezintă trecerea tuturor vectorilor de antrenare prin rețea.

3.4.2. Rețeaua perceptron cu un singur strat (en. *Single Layer Percepron*)

3.4.2.1. Rețeaua perceptron discret cu un singur strat pentru clasificarea în două clase

Acest model, denumit **perceptronul discret**, introdus de Rosenblatt în anul 1958, a fost prima mașină capabilă să învețe.

Se dă setul de antrenament format din *P* perechi:

$\{(X_1, d_1), (X_2, d_2), \dots, (X_P, d_P)\}$ unde:

- X_P este vector de intrare $(n \times 1)$
- d_P este ieșirea dorită (1×1)

Se vor utiliza vectorii de intrare augmentați.

$$X = [1 \quad x_1 \quad x_2 \quad \dots \quad x_n]^T$$

Algoritm de învățare pentru perceptronul discret cu funcția de activare treaptă bipolară

1. **Inițializare**
   - Se inițializează indicii $k = 1$ și $p = 1$.
   - Se inițializează eroarea E cu valoarea 0; $E(k) = 0$.
   - Se inițializează ponderile cu valori mici aleatoare. Ponderile w_i formează vectorul de ponderi augmentat *W* cu dimensiunea $(n + 1) \times 1$.

$$W = \begin{bmatrix} bias \\ w_1 \\ w_2 \\ \dots \\ w_n \end{bmatrix}$$

Buclă iterativă de învățare

2. Se aplică vectorul de intrare X_p și se calculează ieșirea reală o_p:
$$o_p = sgn(W^t \cdot X_p)$$
3. Se ajustează ponderile conform *regulii percepronului:*
$$W(k+1) = W(k) + c \cdot [d_p - o_p(k)] \cdot X_p$$
4. Se calculează eroarea:
$$E(k+1) = E(k) + \frac{1}{2}(d_p - o_p(k))^2$$
5. Dacă p < P, atunci p←p+1, k←k + 1 și se reia de la *pasul 2*, altfel se sare la *pasul 6*.
6. Se testează condiția de stop:
   - dacă $E(k) = 0$, atunci **STOP**
   - dacă $E(k) > 0$ (sau $E(k) > prag\ ales$), atunci $k \leftarrow k+1$, $E(k) = 0$, $p \leftarrow 1$ și se reia de la *pasul 2*.

3.4.2.2. Rețeaua perceptron discret cu un singur strat pentru clasificarea în două sau mai multe clase

Clasificatorul în *R* categorii bazat pe perceptroni discreți poate fi proiectat în două variante:

- **Cu ieșiri decodificate** (cu așa zisa reprezentare locală). Pentru *R* categorii, clasificatorul va conține *R* perceptroni discreți. În acest caz, un *pattern* din clasa *i* aplicat la intrare va revendica la ieșiri: $o_i = 1$, $o_j = -1$, pentru $j = 1, 2 \ldots R$ și $j \neq i$.

Prin urmare, pentru fiecare *pattern* aplicat la intrare, o singură ieșire va fi în "+1", cea aferentă regiunii de decizie din care face parte *pattern*-ul respectiv. Toate celelalte ieșiri vor fi în "−1" (dacă funcția de activare utilizată este *treapta bipolară*), respectiv în "0" (dacă funcția de activare utilizată este *treapta unipolară*).

- **Cu ieșiri codificate** (cu așa zisa reprezentare distribuită). Pentru *R* categorii, clasificatorul va conține $r = lg_2R$ perceptroni discreți. În acest caz, un *pattern* din clasa *i* aplicat la intrare va revendica la cele *R* ieșiri, codul clasei *i*.

De exemplu, dacă $R = 8$, vom avea $r = lg_2 8 = 3$. Prin urmare, clasificatorul cu ieșiri codificate va conține în acest caz 3 perceptroni și un *pattern* din clasa 3 va revendica pe cele 3 ieșiri codul (vectorul) $o=[o_2, o_1, o_0] = [0, 1, 1]$, în reprezentare binară, respectiv $o = [o_2, o_1, o_0] = [-1, 1, 1]$, în reprezentare bipolară.

Ipoteza necesară (pe care o considerăm adevărată) este că, două câte două, clasele (categoriile) sunt separabile.

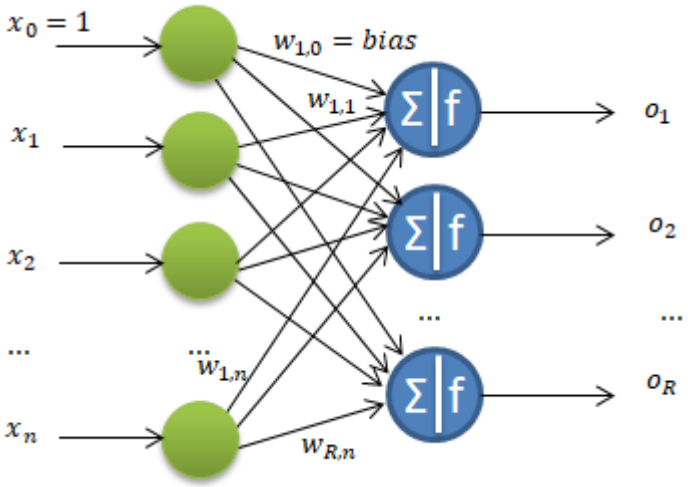


Figura 3.9. Rețeaua perceptron discret cu un singur strat pentru clasificarea în *R* clase

Convenție de notare

Ponderea sinaptică $w_{i,j}$ conectează neuronul *i* la intrarea *j*;

- primul indice (*i*) specifică destinația
- cel de-al doilea indice (*j*) specifică sursa.

Valoarea de activare a neuronului *i* va fi:

$$net_i = \sum_{j=0}^{n} w_{i,j} \cdot x_j + bias_j \quad \text{pentru} \quad i = 1,2,\dots,R$$

Ieșirile celor *R* neuroni vor fi:

$$o_i = f(net_i), \text{ pentru } i = 1,2,..,\ R,$$

f fiind funcția de activare.

Forma vectorială a relației ce caracterizează perceptronul

Se folosesc vectorii augumentați:

$$W_i = \begin{bmatrix} w_{i,0} \\ w_{i,1} \\ w_{i,2} \\ \dots \\ w_{i,n} \end{bmatrix} \qquad \text{X=} \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} \text{ unde:}$$

- W_i reprezintă setul de ponderi asociat neuronului *i* din stratul de ieșire
- $w_{i,j}$ reprezintă ponderea conexiunii neuronului *i* din stratul de ieșire cu nodul *j* din stratul de intrare
- *X* reprezintă vectorul de intrare

Ponderile asociate tuturor neuronilor pot fi scrise matriceal astfel:

$$W = \begin{bmatrix} W_1^T \\ W_2^T \\ \dots \\ W_R^T \end{bmatrix} = \begin{bmatrix} w_{1,0}\ w_{1,1} \dots\ w_{1,n} \\ w_{2,0}\ w_{2,1} \dots\ w_{2,n} \\ \dots . \\ w_{R,0}\ w_{R,1}\ \dots w_{R,n} \end{bmatrix}$$

În aceste condiții, valorile de activare ale neuronilor din stratul de ieșire vor fi:

$$NET = \begin{bmatrix} net_1 \\ net_2 \\ \dots \\ net_R \end{bmatrix} = W \cdot X$$

Iar ieșirile celor *R* neuroni din stratul de ieșire vor fi:

$$O = \begin{bmatrix} o_1 \\ o_2 \\ \dots \\ o_R \end{bmatrix} = f(NET)$$

f fiind funcția de activare, care este în cazul neuronului discret *treapta unipolară* sau *traptă bipolară.*

Algoritm de învățare pentru rețeaua perceptron discret cu un singur strat pentru clasificarea în două sau mai multe clase

Se dă setul de antrenament format din *P* perechi:

$\{(X_1,\ d_1),\ (X_2,\ d_2), \dots, (X_P,\ d_P)\}$ unde:

- X_P este vector de intrare $(n \times 1)$
- d_P este ieșirea dorită $(R \times 1)$

Se vor utiliza vectorii de intrare augmentați.

$$X = [1, x_1, x_2, \dots, x_n]^T$$

1. **Inițializare**.
   - Se inițializează indicii $k = 1$ și $p = 1$;
   - Se inițializează eroarea E cu valoarea 0; $E(k) = 0$
   - Se inițializează ponderile cu valori mici aleatoare.

$$W = \begin{bmatrix} w_{1,0}\ w_{1,1} \dots\ w_{1,n} \\ w_{2,0}\ w_{2,1} \dots\ w_{2,n} \\ \dots . \\ w_{R,0}\ w_{R,1}\ \dots w_{R,n} \end{bmatrix}$$

Buclă iterativă de învățare

2. Se aplică vectorul de intrare X_p și se calculează ieșirea reală O_p

$$O_p = [o_1,\ o_2, \dots,\ o_R]^T$$

$$o_i = f\left(W_i^T \cdot X_p\right),\ unde\ i = 1..R$$

3. Ajustarea ponderilor conform regulii perceptronului:

$$W_i(k+1) = W_i(k) + c \cdot [d_i - o_i(k)]^T \cdot X_p\ \ sau$$

$$W(k+1) = W(k) + c \cdot [D_p - O_p] \cdot X_p^T$$

4. Se calculează eroarea:

$$E(k+1) = E(k) + \frac{1}{2}\sum (d_i - o_i(k))^2$$

5. Dacă p< P, atunci p←p+1, k←k + 1 și se reia de la *pasul 2*, altfel se sare la *pasul 6*.

6. Testare condiție stop:
   - dacă $E(k) = 0$, atunci **STOP**
   - dacă $E(k) > 0$ (sau $E(k) >$ *prag ales*), atunci k←*k+1*, $E(k) = 0$, p←*1* și se reia de la *pasul 2*.

Depinzând de aplicație și de parametrii aleși pentru rețea, eroarea poate fi calculată în diverse moduri, ajustarea ponderilor poate fi făcută cu diverse formule și antrenarea poate fi oprită pe diverse criterii. Însă în toate cazurile, în momentul când antrenarea este finalizată, ponderile rețelei se îngheață (nu mai pot fi modificate) și rețelei îi pot fi prezentați vectorii de test. Aceasta face ca rețelele de perceptroni să aibă o antrenare lentă, consumatoare de timp și resurse, dar o folosire extrem de rapidă (orice vector nou trebuie să treacă o singură dată prin rețea).

3.4.2.3. Implementarea în MATLAB a rețelei percepron cu un singur strat

Aplicația 3.5. Ne propunem să rezolvăm o problemă de clasificare pentru o bază de date artificială conținând 3 clase de vectori; pentru o identificare mai ușoară a claselor le vom identifica vizual astfel: *clasa1* - clasa roșie, *clasa2* - clasa verde, *clasa3* - clasa albastră. Pentru fiecare clasă, punctele sunt generate aleator conform următoarei distribuții normale bidimensionale:

- Media *clasei roșii* $m_1 = [\cos(120^0), \sin(120^0)]$
- Media *clasei verzi* $m_2 = [\cos(240^0), \sin(240^0)]$
- Media *clasei albastre* $m_3 = [\cos(360^0), \sin(360^0)]$

Pentru toate cele 3 clase, dispersia atât pe *Ox* cât și pe *Oy* este $sigma = 0.2$

Etapele rezolvării exemplului de mai sus sunt:

1. generarea bazei de date artificiale
2. dimensionarea rețelei neurale
3. inițializarea ponderilor rețelei neurale
4. antrenarea rețelei neurale
   4.1.stabilirea criteriilor de oprire a antrenării
   4.2.trecerea unui vector/set de vectori (*batch*) prin rețea
   4.3.ajustarea ponderilor
5. evaluarea performanțelor rețelei

1. Generarea bazei de date artificiale

MATLAB-ul nu are funcție pentru distribuția normală bidimensională, dar coordonatele punctelor se pot genera separat cu ajutorul funcției `normrnd(m,sigma)` (distribuția normală unidimensională). Dacă se folosește aceeași valoare `sigma` și pentru axa *Ox* și pentru axa *Oy*, se vor obține mulțimi de puncte asemănătoare cu figura de mai jos, în care punctele nu au o direcție preferențială de împrăștiere. Vectorii de antrenare sunt aliniați în matricea *X*, fiecare coloană a matricii *X* reprezintă un vector de antrenare.

```
N = 100; % N = numar elemente/clasa (pentru lotul de antrenare)
% Mediile claselor si dispersia
m1 = [cos(2*pi/3),sin(2*pi/3)];
m2 = [cos(4*pi/3),sin(4*pi/3)];
m3 = [cos(6*pi/3),sin(6*pi/3)];
sigma = 0.2;% Vectori antrenare
X = [normrnd(m1(1),sigma,N,1), normrnd(m1(2),sigma,N,1)
    normrnd(m2(1),sigma,N,1), normrnd(m2(2),sigma,N,1)
    normrnd(m3(1),sigma,N,1), normrnd(m3(2),sigma,N,1)].';
```

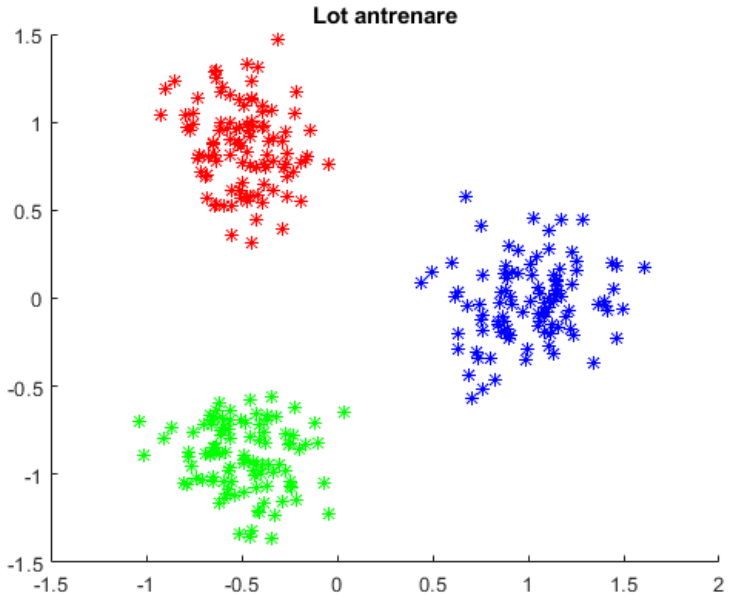


Figura 3.10. Lot antrenare conținând câte 100 de vectori în fiecare clasă

2. Dimensionarea rețelei neurale

Pentru a discuta de dimensiunile rețelei vom considera două cazuri:

a. Rețeaua nu are *bias*. Pentru cazul când nu există *bias*, fiecare vector al bazei de date are 2 elemente (coordonatele pe *Ox* și *Oy*), deci numărul nodurilor de intrare este $n = 2$. Numărul de clase este 3 (roșu, verde, albastru), deci numărul de neuroni ai stratului de ieșire este $m = 3$. Rețeaua are un singur strat de neuroni (stratul de ieșire), deci nu există alte straturi intermediare (ascunse) de neuroni.

Matricea ponderilor W are dimensiunea m x n ➔ $W = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix}$.

b. Rețeaua are *bias*. Dacă rețeaua are *bias*, acest parametru se reflectă în numărul nodurilor de intrare, rezultând $n = 3$ și $W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$.

În continuare vom exemplifica pentru prima variantă, fără *bias*.

Fiecare dintre punctele generate în baza de date are o culoare asociată, fie roșu $[1, 0, 0]^T$, fie verde $[0, 1, 0]^T$, fie albastru $[0, 0, 1]^T$. Una dintre metodele folosite pentru clasificare cu ajutorul rețelelor neurale este alegerea numărului de neuroni pe stratul de ieșire egal cu numărul de clase. Astfel, pentru toate punctele roșii (vectorii de antrenare reprezentați cu roșu în figură), ieșirea dorită a rețelei ar fi $[1, 0, 0]^T$ (primul neuron de ieșire să scoată "1", ceilalți "0"). Similar, pentru punctele verzi ieșirea dorită ar fi $[0, 1, 0]^T$ iar pentru punctele albastre ieșirea dorită ar fi $[0, 0, 1]^T$. Fiecare dintre neuronii de ieșire devine reprezentantul uneia dintre clase.

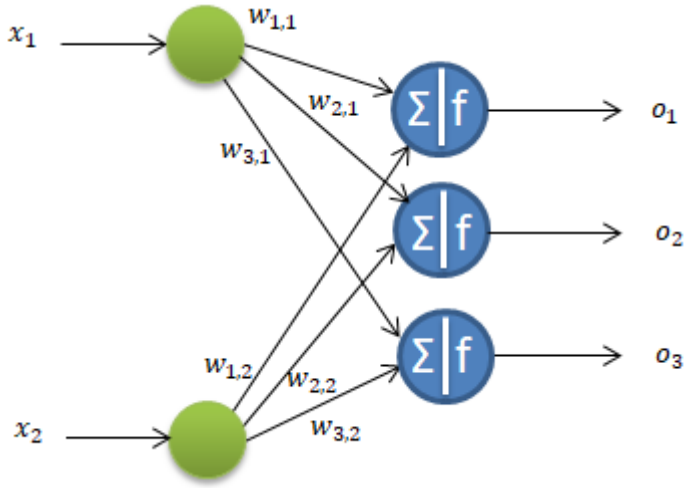


Figura 3.11. Arhitectura rețelei pentru clasificarea în 3 clase a vectorilor bidimensionali fără a folosi *bias*

In mod ideal, când prezentăm rețelei un nou vector, unul dintre neuroni ar avea ieșirea "1" și ceilalți ieșirea "0", iar vectorul va fi clasificat în clasa al cărei neuron reprezentant produce ieșirea "1". In realitate însă, neuronii "concurează" pentru a câștiga vectorul în clasa lor.

Deoarece primele $N = 100$ de coloane reprezintă puncte roșii, următoarele 100 reprezintă puncte verzi, iar ultimele 100 reprezintă puncte albastre, matricea ieșirilor dorite D poate fi construită direct:

```
% iesiri dorite pentru lotul de antrenare
D = [ones(1,N),zeros(1,N),zeros(1,N)
    zeros(1,N), ones(1,N),zeros(1,N)
    zeros(1,N),zeros(1,N), ones(1,N)];
```

3. Inițializarea ponderilor

Inițializarea ponderilor se face cu valori aleatoare mici, urmând ca aceste valori să fie ajustate în urma antrenării. Valorile trebuie să fie mici (de același ordin de mărime) pentru ca niciuna dintre ponderi să nu domine ieșirile rețelei. Se va inițializa matricea ponderilor W din problema de mai sus folosind distribuția uniformă între -0.1 și 0.1.

Inițializarea aleatoare a matricii ponderilor de fiecare dată este problematică deoarece fiecare rulare a aceluiași program produce rezultate diferite. Pentru repetabilitatea experimentelor, este necesar să se inițializeze aleator ponderile o singură dată, apoi matricea de inițializare să fie salvată într-un fisier **.mat* și încărcată de fiecare dată când este nevoie.

```
% initializare ponderii
n = 2; % numar noduri intrare (fara bias)
m = 3; % numar clase
W = unifrnd(-0.1,0.1,[m,n])
```

```
W =
    0.0629    0.0827
    0.0812   -0.0265
   -0.0746   -0.0805
```

4. Antrenarea rețelei neurale

Este foarte puțin probabil ca ponderile inițializate aleator să producă ieșirile dorite, astfel încât este necesară ajustarea acestor ponderi sau antrenarea rețelei. Presupunând că rețeaua de perceptroni a fost corect dimensionată și ponderile au fost inițializate aleator, antrenarea are următoarele etape:

4.1. strabilirea criteriilor de oprire a antrenării
cât timp aceste criterii nu sunt îndeplinite:
4.2. se prezintă unul sau mai mulți vectori de antrenare la intrarea rețelei și se calculează ieșirile obținute *O*
4.3. se ajustează ponderile, în sensul minimizării erorii dintre ieșirile obținute și ieșirile dorite

4.1. Stabilirea criteriilor de oprire a antrenării

Se antrenează rețeaua de perceptroni cu un singur strat până când ponderile nu se mai modifică semnificativ. Rata de învățare $c = 0.001$.

4.2. Trecerea unui vector/set de vectori (*batch*) prin rețea

Trecerea unui vector X prin rețea reprezintă efectuarea calculelor necesare determinării neuronilor de ieșire O.

Exemplu. Să se calculeze ieșirile rețelei de perceptroni cu un singur strat, fără *bias*, cu funcție de ativare liniară pentru stratul de ieșire, pentru vectorul de intrare $X = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ și matricea ponderilor $W = \begin{bmatrix} 0.0629 & 0.0827 \\ 0.0812 & -0.0265 \\ -0.0746 & -0.0805 \end{bmatrix}$. $NET = W \cdot X = \begin{bmatrix} -0.0198 \\ 0.1077 \\ 0.0059 \end{bmatrix}$.

Deoarece funcția de activare este liniară ➔ $O = \varphi(NET) = NET = \begin{bmatrix} -0.0198 \\ 0.1077 \\ 0.0059 \end{bmatrix}$.

Exemplu. Să se calculeze ieșirile rețelei de perceptroni cu un singur strat, fără *bias*, cu funcție de ativare liniară pentru stratul de ieșire, pentru următorii vectori de intrare: $X_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, X_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, X_3 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, X_4 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$ și matricea ponderilor $W = \begin{bmatrix} 0.0629 & 0.0827 \\ 0.0812 & -0.0265 \\ -0.0746 & -0.0805 \end{bmatrix}$.

Se pot calcula ieșirile pe rând, însă mai eficient este să folosim forma matriceală a relațiilor. Astfel, construim matricea X a vectorilor de intrare prin alăturarea (alinierea) tuturor vectorilor de intrare:

$$X = [X_1, X_2, X_3, X_4] = \begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

Astfel, se pot calcula toate ieșirile în același timp:

$$O = \varphi(NET) = \varphi(W \cdot X) = \begin{bmatrix} 0.1456 & -0.0198 & 0.0198 & -0.1456 \\ 0.0547 & 0.1077 & -0.1077 & -0.0547 \\ -0.1551 & 0.0059 & -0.0059 & 0.1551 \end{bmatrix}$$

4.3.Ajustarea ponderilor

Pentru ajustarea ponderilor se folosește regula *Widrow-Hoff* (*Ecuația 3.8*).

Exemplu. Să se determine forma matriceală a regulii de antrenare *Widrow-Hoff* pentru un vector de intrare $X_k = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, pentru care rețeaua de perceptroni cu un singur strat produce vectorul de ieșire $O_k = \begin{bmatrix} o_1 \\ o_2 \\ o_3 \end{bmatrix}$ iar ieșirea dorită este $D_k = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix}$.

Sistemul de ecuații reprezentând contribuția vectorului X_k la modificarea ponderilor este:

$$\begin{cases} \Delta w_{11} = \eta \cdot (d_1 - o_1) \cdot x_1 \\ \Delta w_{12} = \eta \cdot (d_1 - o_1) \cdot x_2 \\ \Delta w_{21} = \eta \cdot (d_2 - o_2) \cdot x_1 \\ \Delta w_{22} = \eta \cdot (d_2 - o_2) \cdot x_2 \\ \Delta w_{31} = \eta \cdot (d_3 - o_3) \cdot x_1 \\ \Delta w_{32} = \eta \cdot (d_3 - o_3) \cdot x_2 \end{cases}$$

In cazul modificării ponderilor prin metoda *batch* (după ce au fost calculate contribuțiile tuturor vectorilor de antrenare), forma matriceală oferă posibilitatea însumării direct a contribuțiilor, unde vectorii de antrenare sunt aliniați pe coloanele matricii $X = [X_1, X_2, \dots, X_N, \dots]$, ieșirile obținute sunt coloane ale matricii $O = [O_1, O_2, \dots, O_N, \dots]$, iar ieșirile dorite sunt coloane ale matricii $D = [D_1, D_2, \dots, D_N, \dots]$. X^T este transpusa matricii X.

$$\Delta W = \begin{bmatrix} \Delta w_{11} & \Delta w_{12} \\ \Delta w_{21} & \Delta w_{22} \\ \Delta w_{31} & \Delta w_{32} \end{bmatrix} = c \cdot (D - O) \cdot X^T$$

Această formă matriceală este valabilă pentru orice dimensiuni ale vectorilor de antrenare, de ieșire obținuți și de ieșire doriți, dacă rețeua a fost dimensionată corect. In fiecare epocă, modificarea ponderilor poate fi pozitivă sau negativă, deci însumarea tuturor elementelor din ΔW poate fi sub pragul ales fără ca ponderile să se fi stabilizat. Pentru a evita această situație, ar trebui însumate valorile absolute sau pătratele elementelor matricii ΔW.

Se realizează antrenarea rețelei de perceptroni cu un singur strat până când ponderile nu se mai modifică semnificativ. Rata de învățare $c = 0.001$.

```
% ajustare ponderi folosind regula Widrow-Hoff
eta = 0.001; % rata de invatare
prag = 0.003;
k = 1; % k = iteratia
deltaW = 1; % orice valoare mai mare de prag
while sum(deltaW(:).^2) > prag
    O = W * X;
    deltaW = eta*(D-O)*X';
    W = W + deltaW;
    disp(['la iteratia k = ',num2str(k),' , ', 'W = '])  %ponderile
    disp(W)
    k = k+1;
end
```

```
la iteratia k = 1 , W =
    0.0030    0.1568
    0.0179   -0.1075
    0.0384   -0.0699
la iteratia k = 2 , W =
   -0.0467    0.2188
   -0.0354   -0.1758
    0.1330   -0.0606
la iteratia k = 3 , W =
   -0.0881    0.2705
   -0.0802   -0.2332
    0.2120   -0.0523
la iteratia k = 4 , W =
   -0.1224    0.3138
   -0.1179   -0.2816
    0.2782   -0.0451
la iteratia k = 5 , W =
   -0.1510    0.3499
   -0.1496   -0.3223
    0.3335   -0.0388
la iteratia k = 6 , W =
   -0.1747    0.3801
   -0.1763   -0.3566
    0.3798   -0.0333
la iteratia k = 7 , W =
   -0.1945    0.4054
   -0.1987   -0.3854
    0.4186   -0.0285
la iteratia k = 8 , W =
   -0.2109    0.4265
   -0.2176   -0.4097
    0.4510   -0.0244
```

5. Evaluarea performanțelor rețelei

Prima metodă de a verifica rezultatele rețelei este de a face testarea rețelei pe lotul de antrenare. Pentru lotul de antrenare se știe deja care sunt ieșirile dorite. Se pot prezenta vectorii de antrenare la intrarea rețelei, se calculează ieșirile obținute și fiecărui vector de antrenare i se asociază o clasă estimată în funcție de neuronul câștigător (care are valoarea maximă). Ideal ar fi ca ieșirile să fie "1" pentru unul dintre neuroni și "0" pentru ceilalți, însă acest lucru nu este întotdeauna adevărat.

Considerând funcția de activare liniară și folosind forma matriceală a vectorilor de intrare $X = [X_1, X_2, \dots, X_{100}, \dots]$, se obține forma matriceală a ieșirilor $O = [O_1, O_2, \dots, O_{100}, \dots] = \varphi(W \cdot X) = W \cdot X$.

```
% ne intereseaza nu atat valorile maxime de pe fiecare coloana, cat pozitia neuronului
care produce aceste valori
[valO,posO] = max(O);
[valD,posD] = max(D);
% numarul de erori este numarul de cazuri in care neuronul castigator nu este cel
asteptat
Nr_erori = sum(posO ~= posD);
disp(['numarul de vectori din lotul de antrenare clasificati gresit = ',
num2str(Nr_erori)])
```

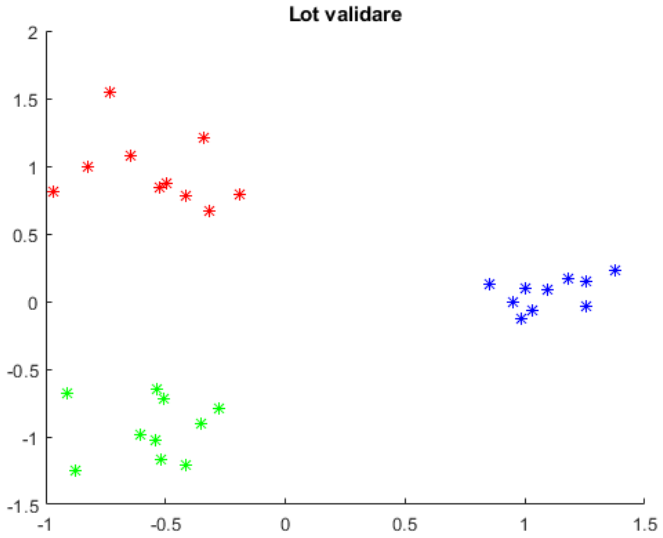
```
numarul de vectori din lotul de antrenare clasificati gresit = 0
```

De remarcat că la etapa de verificare nu se mai modifică ponderile, ci doar se consemnează câți vectori nu sunt clasificați corect. Testarea vectorilor de antrenare nu este foarte relevantă pentru capacitățile rețelei de a rezolva problema, deoarece rețeaua a fost antrenată cu acești vectori. Pentru o validare corectă a rețelei, ar trebui folosit alt set de vectori, despre care rețeaua nu știe nimic, dar pentru care se știu clasele din care ar trebui să facă parte. Pentru a nu amesteca seturile între ele, vectorii de antrenare și cei de validare ar trebui să fie salvați în variabile distincte. In tabelul de mai jos este corespondența elementelor din setul de antrenare și cel de validare.

Antrenare	Validare
Vectori de antrenare $\boldsymbol{X} = [\boldsymbol{X_1 X_2} \dots \boldsymbol{X_{100}} \dots]$	Vectori de validare $Z = [Z_1 Z_2 \dots Z_{100} \dots]$
Ieșirile dorite $\boldsymbol{D} = [\boldsymbol{D_1 D_2} \dots \boldsymbol{D_{100}} \dots]$	Ieșirile dorite (*targets*) $T = [T_1 T_2 \dots T_{100} \dots]$
Ieșirile obținute $\boldsymbol{O} = [\boldsymbol{O_1 O_2} \dots \boldsymbol{O_{100}} \dots]$	Ieșirile obținute $Y = [Y_1 Y_2 \dots Y_{100} \dots]$

Exemplu. Să se genereze un lot de validare și să se determine numărul de erori pentru setul de validare.

```
% lot validare
M = 10; % M = numar elemente/clasa (pentru lotul de antrenare)
% Vectori lot validare
Z = [...
    normrnd(m1(1),sigma,M,1), normrnd(m1(2),sigma,M,1)
    normrnd(m2(1),sigma,M,1), normrnd(m2(2),sigma,M,1)
    normrnd(m3(1),sigma,M,1), normrnd(m3(2),sigma,M,1)].';
figure(2)
hold on
plot(Z(1,      1:M),Z(2,    1:M)  ,'r*');
plot(Z(1,  M+1:2*M),Z(2,  M+1:2*M),'g*');
plot(Z(1,2*M+1:3*M),Z(2,2*M+1:3*M),'b*');
hold off
title('Lot validare')
```

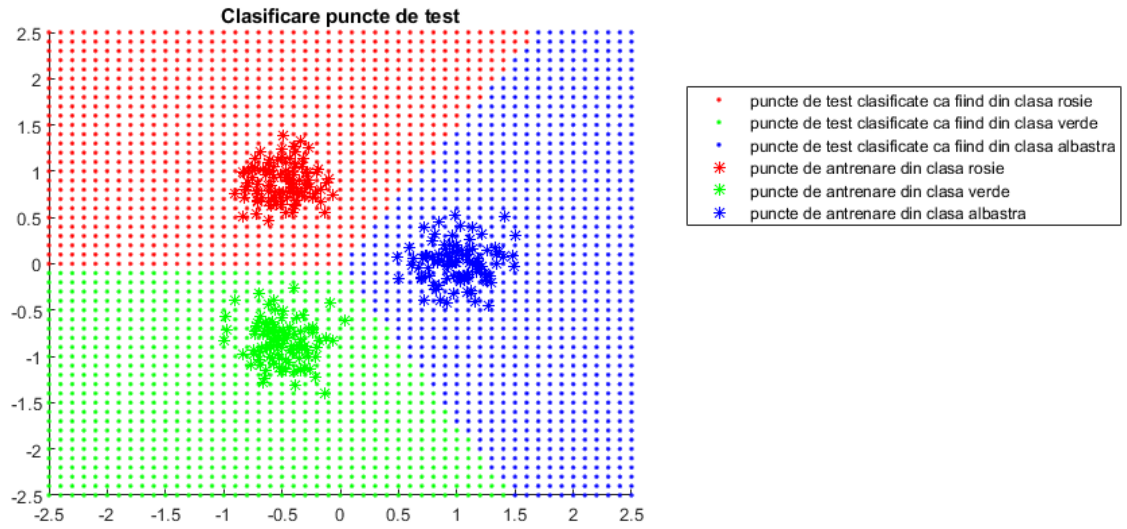



```
% iesiri dorite pentru lotul de validare
T = [ones(1,M),zeros(1,M),zeros(1,M)
    zeros(1,M), ones(1,M),zeros(1,M)
    zeros(1,M),zeros(1,M), ones(1,M)];
Y = W * Z;
[valY,posY] = max(Y);
[valT,posT] = max(T);
% numarul de erori este numarul de cazuri in care neuronul castigator nu este cel
asteptat
Nr_erori_validare = sum(posY ~= posT)
disp(['numarul de vectori din lotul de validare clasificati gresit = ',
num2str(Nr_erori_validare)])
```

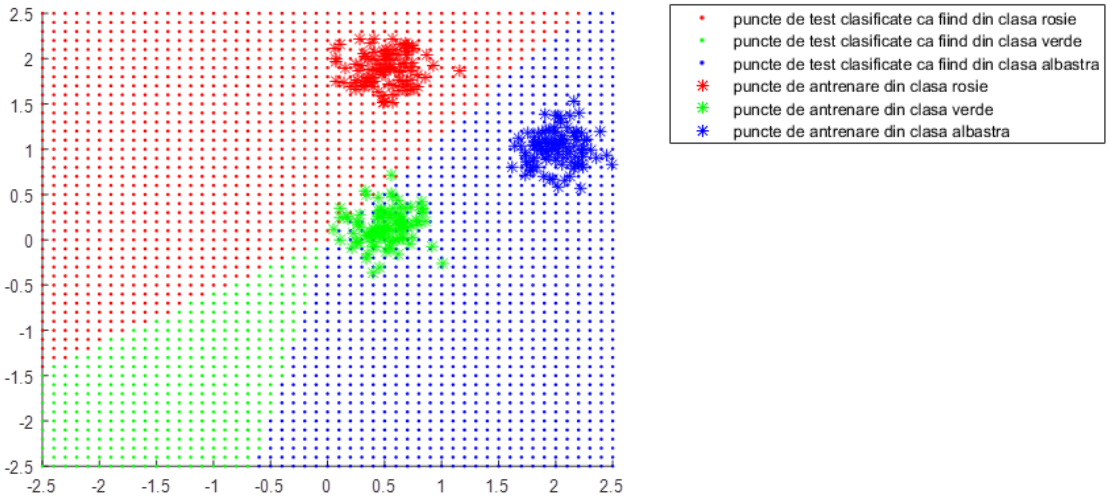
```
numarul de vectori din lotul de validare clasificati gresit = 0
```

În continuare vor fi clasificate mai multe puncte despre a căror apartenență la clase nu se știe nimic, rezultatul clasificării fiind interpretat strict vizual.

```
% vizualizare puncte de test clasificate
figure(3)
hold on
for i = -2.5 : 0.1 : 2.5
    for j = -2.5 : 0.1 : 2.5
        C = [i,j].'; % noul vector de intrare (un punct in plan)
        curent_O = W * C;
        [val,pos] = max(curent_O);
        % se cauta ce neuron a obtinut max si se coloreaza corespunzator
        if pos == 3
            plot(i,j,'.','color',[0,0,1])
        end
        if pos == 2
            plot(i,j,'.','color',[0,1,0])
        end
        if pos == 1
            plot(i,j,'.','color',[1,0,0])
        end
    end
end
```



Atunci când nu se folosește *bias*, punctul de intrare $X_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ va fi mereu punctul de intersecție al (semi)dreptelor de separație. Având această constrângere, chiar și clase care sunt ușor separabile (vizual) pot deveni imposibil de separat de către o rețea de perceptroni cu un singur strat. De exemplu, pentru baza de date de mai jos, dacă nu folosim *bias*, partiționarea spațiului arată astfel:



Bias reprezintă termenul liber din ecuațiile perceptronilor și permite "mutarea" intersecției planelor de separație. În cazul în care se folosește *bias*, arhitectura rețelei arată ca mai jos.

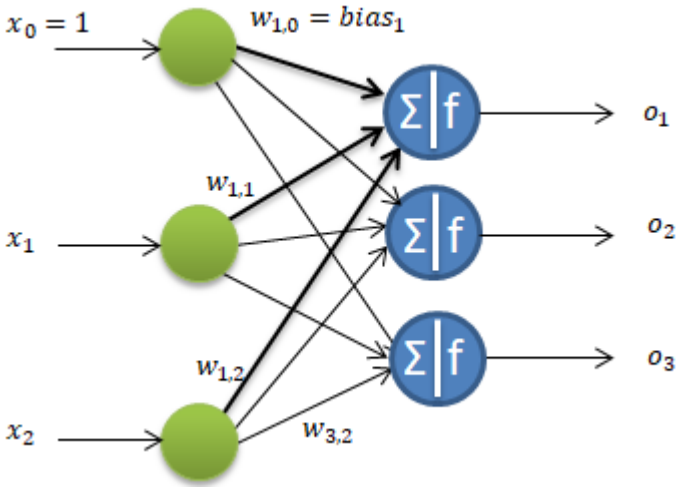


Figura 3.12. Arhitectura rețelei pentru clasificarea în 3 clase a vectorilor bidimensionali folosind *bias*

Relațiile care descriu rețeaua sunt:

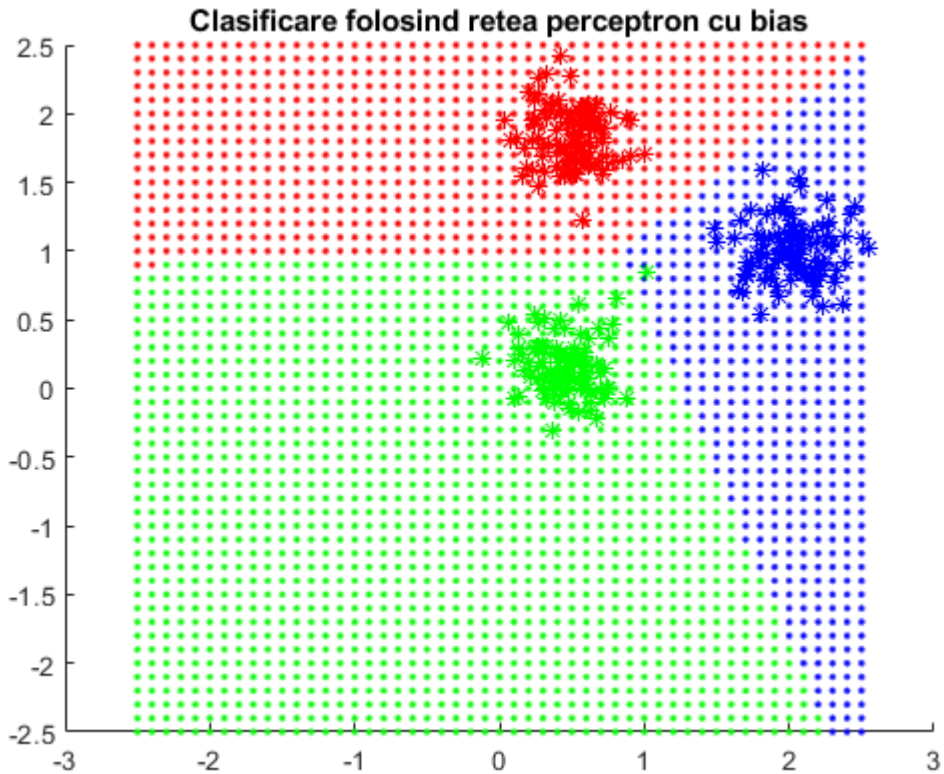
$$\begin{cases} net_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x_3 \\ net_2 = w_{21}x_1 + w_{22}x_2 + w_{23}x_3 \\ net_3 = w_{31}x_1 + w_{32}x_2 + w_{33}x_3 \end{cases} \qquad \begin{cases} o_1 = \varphi(net_1) = net_1 \\ o_2 = \varphi(net_2) = net_2 \\ o_3 = \varphi(net_3) = net_3 \end{cases}$$

unde $w_{13} = bias_1$, $w_{23} = bias_2$, $w_{33} = bias_3$ și $x_3 = 1$.
Sistemul de ecuații devine uniform și poate fi scris sub formă matriceală astfel:

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}, X = \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix}, NET = \begin{bmatrix} net_1 \\ net_2 \\ net_3 \end{bmatrix} = W \cdot X, O = \begin{bmatrix} o_1 \\ o_2 \\ o_3 \end{bmatrix} = \varphi(NET)$$

```
% N = numar elemente/clasa (pentru lotul de antrenare)
N = 100;

% Mediile claselor si dispersia
m1 = [cos(2*pi/3), sin(2*pi/3)]+1;
m2 = [cos(4*pi/3), sin(4*pi/3)]+1;
m3 = [cos(6*pi/3), sin(6*pi/3)]+1;
sigma = 0.2;

% Vectori antrenare
x3 = 1;
% deoarece folosim vectori augmentati pentru a putea utiliza scrierea
% matriceala, vom introduce pe ultima pozitie a vectorului de intrare
% elementul x3 corespunzator ponderii wk3 (pentru bias)
X = [normrnd(m1(1),sigma,N,1),normrnd(m1(2),sigma,N,1),x3*ones(N,1)
    normrnd(m2(1),sigma,N,1),normrnd(m2(2),sigma,N,1),x3*ones(N,1)
    normrnd(m3(1),sigma,N,1),normrnd(m3(2),sigma,N,1),x3*ones(N,1)] .';

% Iesiri dorite pentru lotul de antrenare
D = [ones(1,N),  zeros(1,N), zeros(1,N)
    zeros(1,N),  ones(1,N), zeros(1,N)
    zeros(1,N), zeros(1,N), ones(1,N)];

n = 3; % numar noduri intrare (cu bias)
m = 3; % numar clase
W = unifrnd(-1,1,n,m); % Initializare ponderi
eta = 0.001; % Rata de invatare
% Ajustare ponderi folosind regula Widrow-Hoff
deltaW = 1;
prag = 0.001;
while sum(deltaW(:).^2) > prag
    O = W * X;
    deltaW = eta*(D-O)*X.';
    W = W + deltaW;
end
```

```
% Vizualizare puncte de test clasificate
figure(1)
hold on
for i = -2.5 : 0.1 : 2.5
    for j = -2.5 : 0.1 : 2.5
        C = [i,j,x3].'; % intrare
        curent_O = W * C;
        [val,pos] = max(curent_O); % max din curent_O
        if pos == 3
            plot(i,j,'.','color',[0,0,1])
        end
        if pos == 2
            plot(i,j,'.','color',[0,1,0])
        end
        if pos == 1
            plot(i,j,'.','color',[1,0,0])
        end
    end
end
plot(X(1,      1:N),X(2,    1:N)  ,'r*');
plot(X(1,  N+1:2*N),X(2,  N+1:2*N),'g*');
plot(X(1,2*N+1:3*N),X(2,2*N+1:3*N),'b*');
hold off
title('Clasificare folosind retea perceptron cu bias')
```



3.4.3. Rețeaua perceptron multistrat (en. *Multi Layer Perceptron*)

În capitolele anterioare am utilizat rețelele neurale pentru clasificarea *pattern*-urilor **liniar separabile**. Dacă *pattern*-urile nu sunt **liniar separabile**, apelăm la arhitecturi neurale mai complexe, de exemplu **rețele multistrat**. Ca și în cazul rețelelor monostrat, învățarea va fi **supervizată** și se bazează pe aplicarea repetată, ciclică, la intrarea rețelei a tuturor prototipurilor (*pattern*-urilor din cadrul setului de antrenament).

Reţelele multistrat pot fi formate prin simpla cascadare de straturi, ieşirile unui strat devenind intrări pentru stratul următor. Majoritatea reţelelor *MultiLayer Perceptron* (MLP) utilizează algoritmul de *backpropagation* pentru învăţare. Algoritmul de *backpropagation* foloseşte eroarea între ieşirile actuale şi ieşirile aşteptate, pentru a ajusta fiecare pondere. Ajustarea ponderilor se face secvenţial, plecând de la ultimul strat (cel de ieşire), spre primul strat (cel de intrare). O arhitectură tipică pentru un perceptron cu două straturi (un strat intermediar și stratul de ieșire) este prezentată în *Figura 3.13*.

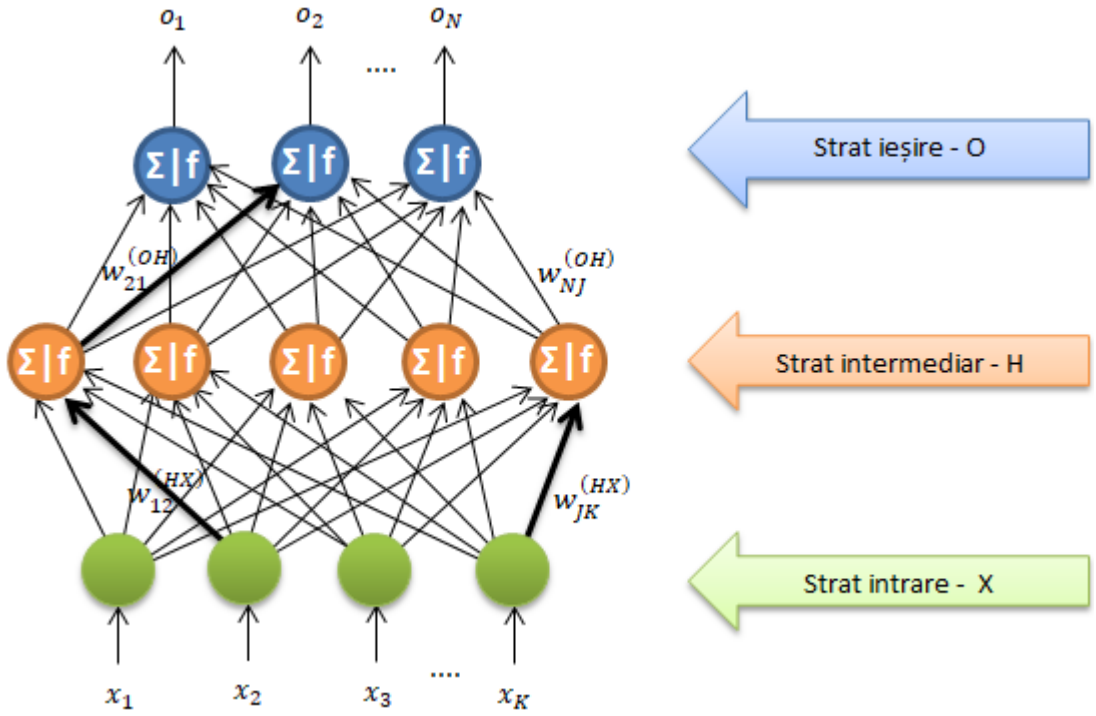


Figura 3.13. Rețeaua MLP cu un strat ascuns

3.4.3.1. Arhitectura rețelei MLP

- **Stratul de intrare.** Numărul neuronilor din stratul de intrare este egal cu dimensiunea spațiului vectorului de intrare.

Obs! Nodurile din stratul de intrare nu realizează nicio procesare; ele doar distribuie componentele vectorului de intrare pe intrările neuronilor din primul strat de neuroni (stratul ascuns). Prin urmare, rețeaua *feedforward* de mai sus este o rețea cu două straturi (stratul ascuns și stratul de ieșire).

- **Straturi ascunse.** Rețeaua poate avea unul sau mai multe straturi ascunse, depinzând de aplicație. Nu există o formulă cu care să se calculeze numărul de neuroni necesari într-un strat ascuns, acesta variind în funcție de aplicație.

Obs! Un număr prea mare de unități ascunse poate provoca supra-antrenare (rețeaua extrage nu doar informațiile utile din setul de antrenare ci și zgomotul).

- **Stratul de ieşire.** Numărul neuronilor din stratul de ieşire este egal cu numărul de clase distincte pe care trebuie să le recunoască rețeaua.

Din punct de vedere al relațiilor matematice, neuronii de pe stratul intermediar se comportă la fel ca neuronii de pe stratul de ieșire, însă este posibil ca funcțiile de activare să fie diferite între cele 2 straturi.

În continuare sunt prezentate relaţiile care descriu funcţionarea perceptronului multistrat.

$$\begin{cases} o_n^{(p)} = f(\sum_{j=1}^{J} w_{nj}^{(OH)} \cdot y_j^{(p)} + bias) \\ y_j^{(p)} = f(\sum_{k=1}^{K} w_{jk}^{(HX)} \cdot x_k^{(p)} + bias) \end{cases} \tag{3.14}$$

- N: numărul de neuroni din stratul de ieşire;
- J: numărul de neuroni din stratul ascuns;
- K: dimensiunea vectorului de intrare;
- $o_n^{(p)}$: ieşirea neuronului n din stratul de ieşire pentru forma p de intrare;
- $y_j^{(p)}$: ieşirea neuronului j din stratul intermediar pentru forma p de intrare;
- $x_k^{(p)}$: valoarea aplicată intrării k în reţea pentru forma p de intrare;
- $w_{nj}^{(OH)}$: ponderea conexiunii neuronului n din stratul de ieşire cu neuronul j din stratul intermediar;
- $w_{jk}^{(HX)}$: ponderea conexiunii neuronului j din stratul intermediar cu nodul k din stratul de intrare;

Obiectivul antrenării reţelei neurale este de a ajusta ponderile astfel încât aplicarea unui set de intrări să producă ieşirea dorită. Antrenarea atribuie fiecărui vector de intrare un vector pereche denumit vector ţintă care reprezintă ieşirea dorită.

Funcţia de eroare corespunzătoare formei de intrare *p* este dată de:

$$E_p = \frac{1}{2}\sum_{n=1}^{N}(d_n^{(p)} - o_n^{(p)})^2 \tag{3.15}$$

unde *o* este ieşirea reţelei şi *d* este ieşirea dorită.

Scrierea matriceală a relațiilor ce descriu rețeaua MLP

Față de rețeaua de perceptroni cu un singur strat, la rețeaua MLP există 2 seturi de ponderi:

- $W^{(HX)}$ este matricea de ponderi dintre intrare și stratul ascuns (*hidden*)
- $W^{(OH)}$ este matricea de ponderi dintre stratul ascuns (*hidden*) și stratul de ieșire. Valorile interne ale neuronilor și funcțiile de activare sunt:
- NET^H și φ^H pentru stratul ascuns (*hidden*)
- NET^O și φ^O pentru stratul de ieșire.

Având aceste notații, relațiile (în forma matriceală) care definesc rețeaua MLP sunt:

$$NET^H = W^{(HX)} \cdot X, \qquad H = \varphi^H(NET^H)$$
$$NET^O = W^{(OH)} \cdot H, \qquad O = \varphi^O(NET^O)$$

De remarcat că ieșirile H ale stratului ascuns devin intrări pentru stratul de ieșire. Dacă ar fi fost mai multe straturi ascunse, ieșirile fiecărui strat intermediar ar fi devenit intrările următorului strat intermediar.

Puterea de clasificare a rețelelor MLP este dată de funcțiile de activare neliniare de pe straturile intermediare. Dacă toate funcțiile de activare ar fi liniare, atunci pentru orice rețea MLP, cu oricâte straturi ascunse, ar exista o rețea *single layer perceptron* cu ponderi calculate astfel încât să aibă performanțe identice cu rețeaua MLP.

3.4.3.2. Antrenarea rețelei MLP

Pasul 1: Se inițializează ponderile. Se setează toate ponderile nodurilor la valori aleatoare mici. În continuare se consideră funcția de activare *sigmoidală bipolară.*

Pasul 2: Se aplică intrarea X_i şi ieşirea dorită D_i. Dacă reţeaua este folosită drept clasificator, de obicei toate ieșirile dorite sunt setate la "0", exceptând cele care corespund clasei valorilor de intrare, care sunt setate la "1".

Pasul 3: Se calculează ieșirile rețelei $O=\{ o_1, o_2 ... o_N \}$

Pasul 4: Se modifică ponderile rețelei:

$$\begin{cases} w_{nj}^{(OH)}(t+1) = w_{nj}^{(OH)}(t) + \eta \cdot \delta_n(t) \cdot y_j^{(p)}(t), \text{pentru n } = 1,2,..,\text{N} \\ w_{jk}^{(HX)}(t+1) = w_{jk}^{(HX)}(t) + \eta \cdot \delta_j(t) \cdot x_k^{(p)}(t), \text{pentru j } = 1,2,..,\text{J} \end{cases} \quad (3.16)$$

unde:

- $\eta =$ rata de învățare, $\eta \in [0,1]$
- $\delta_n^{(O)}$= erorile aferente neuronilor din stratul de ieșire
  $\delta_n^{(O)} = \frac{1}{2} \cdot (d_n - o_n)(1 - o_n^2)$, pentru $n = 1,2,..,N$
- $\delta_j^{(H)}$= erorile aferente neuronilor din stratul intermediar
  $\delta_j^{(H)} = \frac{1}{2} \cdot (1 - y_j^2) \sum_{n=1}^{N} \delta_n^{(O)} \cdot w_{nj}^{(OH)}$, pentru $j = 1,2,..,J$

Pasul 5: Dacă nu s-au terminat de introdus toţi vectorii din lotul de antrenare, atunci $i = i + 1$ şi salt la **Pasul2**. Dacă s-au terminat de introdus toţi vectorii din lotul de antrenare, atunci salt la **Pasul6**

Pasul 6: Dacă nu este îndeplinită **condiţia de stop**, atunci salt la **Pasul 2**, dacă da, atunci STOP.

Condiţia de stop poate fi una dintre următoarele:

- număr prestabilit de paşi (epoci);
- eroarea medie patratică globală scade sub un prag; criteriul de oprire se bazează pe eroarea calculată pentru setul de validare (NU pentru un setul de antrenare)
- numărul de erori de clasificare pentru setul de antrenare este zero.

După un număr suficient de epoci, eroarea dintre ieşirea actuală şi vectorul ţintă se va reduce ca valoare, iar reţeaua se spune că este antrenată. Din acest punct, reţeaua este folosită pentru recunoaştere şi ponderile nu se mai schimbă.

3.4.4. Implementarea rețelei MLP în *MATLAB*

1. Pentru definirea rețelei MLP se poate folosi funcția `feedforwardnet`:

Sintaxă: `MLPnet = feedforwardnet(N)` unde:

- *N* = numărul de neuroni de pe stratul sau straturile ascunse. Dacă *N* este un număr, atunci funcția consideră că există un singur strat ascuns de *N* neuroni; dacă *N* este vector, atunci numărul de straturi ascunse este egal cu numărul de elemente din vector, iar numărul de neuroni din fiecare strat este egal cu valoarea elementelor din vectorul *N*. Numărul de neuroni din stratul ascuns este dat explicit funcției, însă numărul de noduri din stratul de intrare și numărul de neuroni din stratul de ieșire este determinat implicit de dimensiunea datelor de intrare, respectiv de ieșirile dorite.

2. Antrenarea rețelei se face cu funcția `train`.

Sintaxă: `MLPnet = train(MLPnet,X,D)` unde:

- `MLPnet =` rețeaua MLP creată anterior cu funcția `feedforwardnet`
- `X =` matricea vectorilor de antrenare, așezați pe coloane
- `D =` ieșirile dorite (etichetele vectorilor din lotul de antrenare)

În urma antrenării, rețeaua este dimensionată automat în funcție de intrarea `X` și ieșirile dorite `D`.

3. Pentru a vizualiza rețeaua se poate folosi funcția `view`.

Sintaxă: `view(MLPnet)`

4. Pentru a testa rețeaua se poate folosi sintaxa:

Sintaxă: `O = MLPnet(Y)` unde:

- `MLPnet =` rețeaua MLP creată anterior cu funcția `feedforwardnet`
- `Y` = lotul de test, `O` = ieșirile obținute în urma clasificării

Obs: pentru sintaxa completă a funcțiilor de mai sus consultați *help*-ul din MATLAB

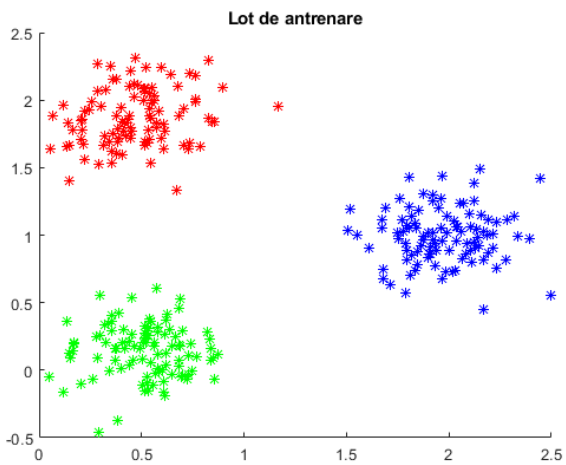
Aplicația 3.6. Clasificare în 3 clase – exemplu didactic

Ne propunem să rezolvăm o problemă de clasificare pentru o bază de date artificială conținând 3 clase de vectori; pentru o identificare mai ușoară a claselor, acestea vor fi identificate vizual astfel: *clasa1* - clasa roșie, *clasa2* - clasa verde, *clasa3* - clasa albastră. Pentru fiecare clasă, punctele sunt generate aleator conform următoarei distribuții normale:

- Media *clasei roșii* $m_1 = [1+\cos(120^0), 1+\sin(120^0)]$
- Media *clasei verzi* $m_2 = [1+\cos(240^0), 1+\sin(240^0)]$
- Media *clasei albastre* $m_3 = [1+\cos(360^0), 1+\sin(360^0)]$
- Pentru toate cele 3 clase dispersia este $sigma = 0.2$.

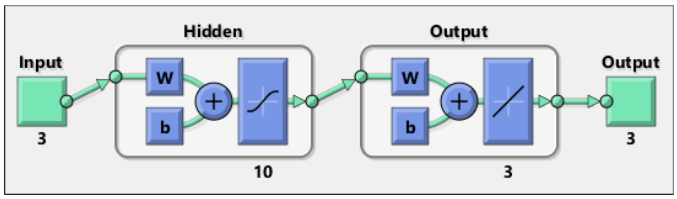
Observație: spre deosebire de exemplul anterior, de această dată vom folosi funcțiile MATLAB-ului pentru crearea, antrenarea și testarea rețelelor neurale.

```
% P = numar elemente/clasa (pentru lotul de antrenare)
P = 100;

% Mediile claselor si dispersia
m1 = [cos(2*pi/3),sin(2*pi/3)]+1;
m2 = [cos(4*pi/3),sin(4*pi/3)]+1;
m3 = [cos(6*pi/3),sin(6*pi/3)]+1;
sigma = 0.2;

% Vectori antrenare
x3 = 1;
% deoarece folosim vectori augmentati pentru a putea utiliza scrierea
% matriceala, vom introduce pe ultima pozitie a vectorului de intrare
% elementul x3 corespunzator ponderii wk3 (pentru bias)
X = [...
   normrnd(m1(1),sigma,P,1), normrnd(m1(2),sigma,P,1) ,x3*ones(P,1)
   normrnd(m2(1),sigma,P,1), normrnd(m2(2),sigma,P,1) ,x3*ones(P,1)
   normrnd(m3(1),sigma,P,1), normrnd(m3(2),sigma,P,1) ,x3*ones(P,1)
].';

% Iesirile dorite
D = [...
     ones(1,P), zeros(1,P), zeros(1,P)
    zeros(1,P),  ones(1,P), zeros(1,P)
    zeros(1,P), zeros(1,P),  ones(1,P)];
figure(1)
hold on
plot(X(1,      1:P),X(2,    1:P)  ,'r*');
plot(X(1,  P+1:2*P),X(2,  P+1:2*P),'g*');
plot(X(1,2*P+1:3*P),X(2,2*P+1:3*P),'b*');
hold off
title('Lot de antrenare')
```



Se va implementa o rețea MLP cu 10 neuroni pe stratul intermediar.

```
J = 10; % J = numar neuroni strat intermediar
MLPnet = feedforwardnet(J); % implementare retea MLP cu 10 neuroni pe stratul ascuns
MLPnet = train(MLPnet,X,D); % antrenare retea MLP
figure(2), view(MLPnet) % vizualizare retea
```



Pentru antrenare este deschisă o fereastră (`nntraintool`) care conține informațiile despre algorimii de antrenare și partiționarea datelor, criteriile de oprire și butoane pentru vizualizarea performanțelor rețelei.

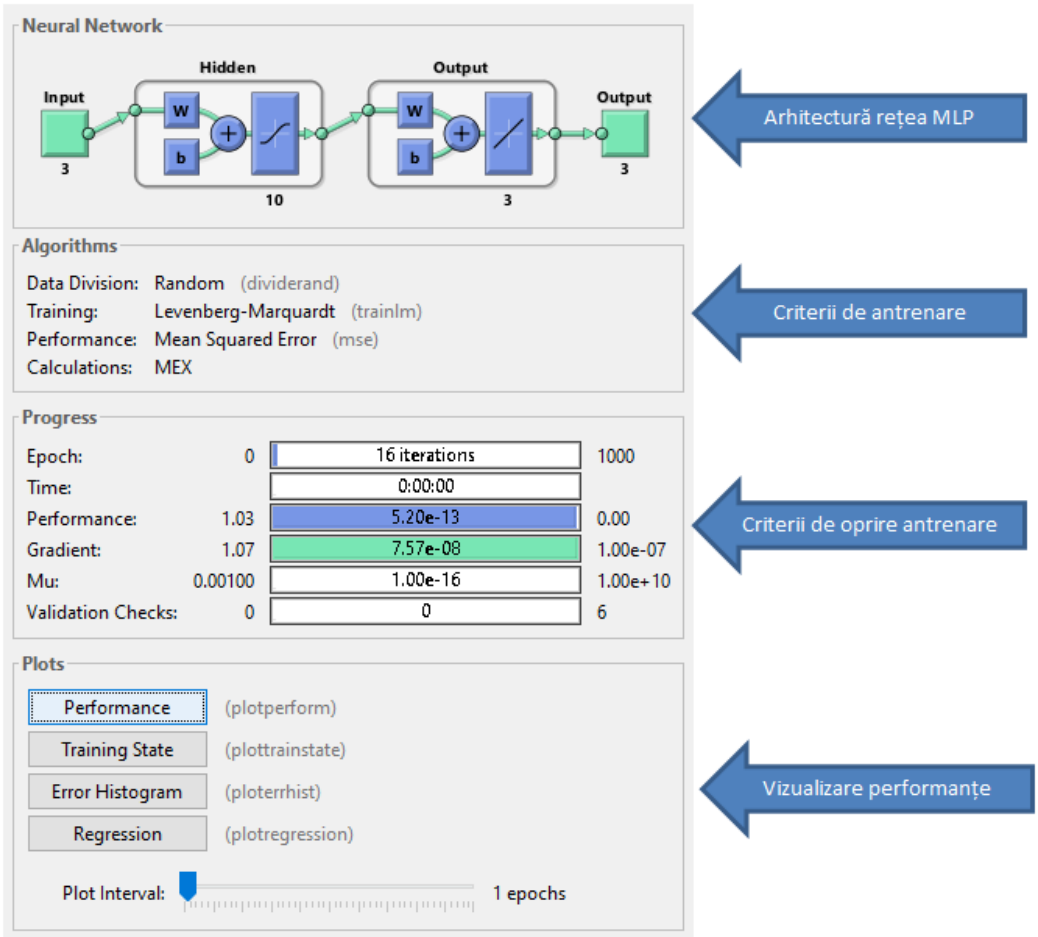


Figura 3.14. Fereastră `nntraintool`

Criterii de antrenare [Mathwork1]

Atunci când se antrenează o rețea MLP, implicit funcția `train` din MATLAB împarte lotul de antrenare în trei subseturi:

- Primul subset este *setul de instruire*, care este utilizat pentru calcularea gradientului și pentru actualizarea ponderilor și parametrilor *bias*.
- Al doilea subset este *setul de validare*. Eroarea de pe setul de validare este monitorizată în timpul procesului de instruire. În mod normal, eroarea de validare scade în faza inițială a antrenamentului, la fel ca și eroarea pe lotul de instruire. Cu toate acestea, atunci când rețeaua începe să supra-ajusteze ponderile (proces de *overfiting*), eroarea de pe setul de validare începe de obicei să crească.
- Al treilea set este *setul de testare*. Eroarea setului de testare nu este utilizată în timpul antrenamentului, dar este utilizată pentru a compara diferite modele.

Există patru funcții furnizate pentru împărțirea datelor în seturi de instruire, validare și testare. Ele sunt `dividerand`, `splitblock`, `divideint` și `divide`. Împărțirea datelor se realizează în mod automat atunci când se antrenează rețeaua.

Funcția implicită (*default*) utilizată de funcția `train` din MATLAB este `dividerand`, care împarte random setul de date în 70% lot de instruire, 15% lot de validare și 15% lot de testare.

Dacă pentru o rețea `MLPnet` se dorește o altă împărțire a datelor din lotul de antrenare decât cea implicită realizată de funcția `dividerand`, atunci i se poate specifica acest lucru rețelei `MLPnet` astfel:

```
% lotul de instruire contine 70% dintre vectorii de antrenare
MLPnet.divideParam.trainRatio = 0.7;
% lotul de validare contine 30% dintre vectorii de antrenare
MLPnet.divideParam.valRatio = 0.3;
% lotul de testare nu contine niciun vector
% pentru testarea retelei va fi folosit un alt set de date
MLPnet.divideParam.testRatio = 0;
```

Procesul de antrenare a unei rețele neurale presupune ajustarea ponderilor și parametrilor *bias* pentru a optimiza performanța rețelei, așa cum este definită de funcția de performanță a rețelei `MLPnet`:

- Funcția implicită folosită pentru a calcula performanța pentru rețelele MLP este eroarea medie pătratică (`mse`) între ieșirile obținute de rețea *O* și ieșirile dorite *D*.
- Funcția implicită folosită pentru actualizarea ponderilor și a parametrilor *bias* este `trainlm` care implementează algoritmul *Levenberg-Marquardt* [Marquardt]

Progresul antrenării. Criterii de oprire a antrenării [Mathwork2]

În timpul etapei de antrenare, progresul rețelei este actualizat constant în fereastra `nntraintool`, în secțiunea *Progress*. Cei mai relevanți parametri pentru a evalua progresul antrenării sunt:

- *Epoch (Numărul de epoci).* Numărul de epoci reprezintă numărul de iterații al algoritmului de antrenare. Implicit numărul maxim de epoci este 1000. Dacă se atinge numărul maxim de epoci algoritmul se încheie. Acest parametru se poate modifica folosind funcția `MLPnet.trainParam.epochs`.
- *Time* (*Timp).* Reprezintă durata maximă cât să ruleze algoritmul de antrenare. Implicit valoarea parametrului *timp* este `Inf`. Acest parametru se poate modifica folosind funcția `MLPnet.trainParam.time`.
- *Performance (Performanța).* Reprezintă eroarea medie pătratică (dacă se folosește varianta implicită `mse`) . Implicit, dacă `mse` ajunge să aibă valoarea 0, antrenamentul se va opri. Acest parametru se poate modifica folosind funcția `MLPnet.trainParam.goal`.
- *Gradient (Valoarea gradientului de performanță)*. Implicit, dacă mărimea gradientului este mai mică de 1e-7, antrenamentul se va opri. Acest parametru se poate modifica folosind funcția `MLPnet.trainParam.min_grad`.
- *Validation Cheks (Numărul de verificări de validare).* Reprezintă numărul de iterații succesive pentru care performanța pe lotul de validare nu scade. Implicit, dacă acest număr ajunge la 6, algoritmul de antrenare se oprește. Acest parametru se poate modifica folosind funcția `MLPnet.trainParam.max_fail`.

Vizualizare performanțe algoritm de antrenare

După încheierea etapei de antrenare pot fi vizualizate performanțele antrenării. Cel mai relevant grafic îl reprezintă evoluția parametrului `mse` pentru lotul de instruire și cel de validare de-a lungul epocilor de antrenare. Acest grafic poate fi vizualizat din fereastra `nntraintool`, secțiunea *Plots*, butonul *Performance.*

Pentru exemplul de mai sus în care trebuie clasificate cele 3 clase, pentru cazul în care se setează împărţirea lotului de antrenare doar în lot de instruire (70% din vectori) şi lot de validare (30% din vectori), antrenarea durează 14 epoci. După 14 epoci s-a ajuns ca valoarea gradientului să fie $3.66 \cdot 10^{-8}$, ceea ce a dus la oprirea algoritmului de antrenare ($< 10^{-7}$). Pentru această valoare a gradientului, valoarea parametrului `mse` este $7.32 \cdot 10^{-14}$ (aproape nulă), ceea ce înseamnă că vectorii de validare sunt foarte bine clasificați.

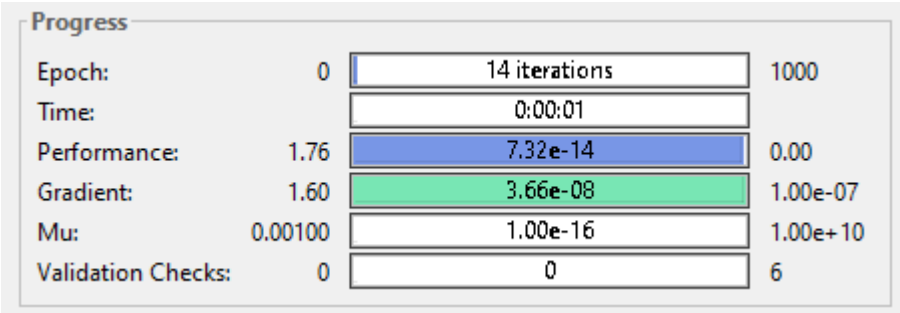


Figura 3.15. Fereastra `nntraintool`, secțiunea *Progress*

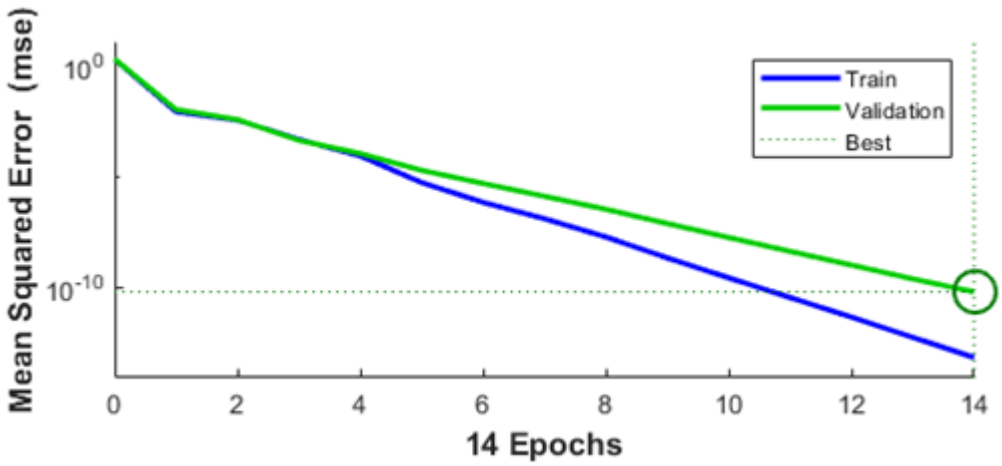


Figura 3.16. Evoluția parametrului `mse` în funcție de epocă

Pentru rețeaua MLP, nu doar caracteristicile din fereastra `nntraintool` pot fi modificate, ci toate proprietățile rețelei. De exemplu:

- Pentru a vizualiza parametrii primului strat ascuns:

```
>> MLPnet.layers{1}
```

- Pentru a modifica funcția de activare a neuronilor primului strat ascuns:

```
>> MLPnet.layers{1}.transferFcn = functiaDorita
```

- Pentru a modifica funcția de antrenare:

```
>> MLPnet.trainFcn = functiaDorita
```

- Pentru a modifica criteriile de oprire a antrenării:

```
>> MLPnet.trainParam.[numeleParametruluiDorit] = valoareDorita
```

Pentru a vedea structura folosită de MATLAB pentru stocarea rețelei `MLPnet` creată, este suficient să fie scrisă în *Command Window* comanda:

```
>> MLPnet
```

În *Command Windowse* va afișa:

```
MLPnet =
    Neural Network
                name: 'Feed-Forward Neural Network'
            userdata: (your custom info)
     dimensions:
         numInputs: 1
         numLayers: 2
        numOutputs: 1
    numInputDelays: 0
    numLayerDelays: 0
 numFeedbackDelays: 0
 numWeightElements: 63
        sampleTime: 1
    connections:
       biasConnect: [1; 1]
      inputConnect: [1; 0]
      layerConnect: [0 0; 1 0]
     outputConnect: [0 1]
     subobjects:
              input: Equivalent to inputs{1}
             output: Equivalent to outputs{2}

             inputs: {1x1 cell array of 1 input}
             layers: {2x1 cell array of 2 layers}
            outputs: {1x2 cell array of 1 output}
             biases: {2x1 cell array of 2 biases}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}
     functions:
           adaptFcn: 'adaptwb'
        adaptParam: (none)
          derivFcn: 'defaultderiv'
         divideFcn: 'dividerand'
       divideParam: .trainRatio, .valRatio, .testRatio
        divideMode: 'sample'
           initFcn: 'initlay'
        performFcn: 'mse'
      performParam: .regularization, .normalization
          plotFcns: {'plotperform', plottrainstate, ploterrhist,
                    plotregression}
        plotParams: {1x4 cell array of 4 params}
          trainFcn: 'trainlm'
        trainParam: .showWindow, .showCommandLine, .show, .epochs,
                    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
                    .mu_inc, .mu_max
     weight and bias values:
                 IW: {2x1 cell} containing 1 input weight matrix
                 LW: {2x2 cell} containing 1 layer weight matrix
                  b: {2x1 cell} containing 2 bias vectors
     methods:
              adapt: Learn while in continuous use
          configure: Configure inputs & outputs
             gensim: Generate Simulink model
               init: Initialize weights & biases
            perform: Calculate performance
                sim: Evaluate network outputs given inputs
              train: Train network with examples
               view: View diagram
        unconfigure: Unconfigure inputs & outputs

    evaluate:       outputs = MLPnet(inputs)
```

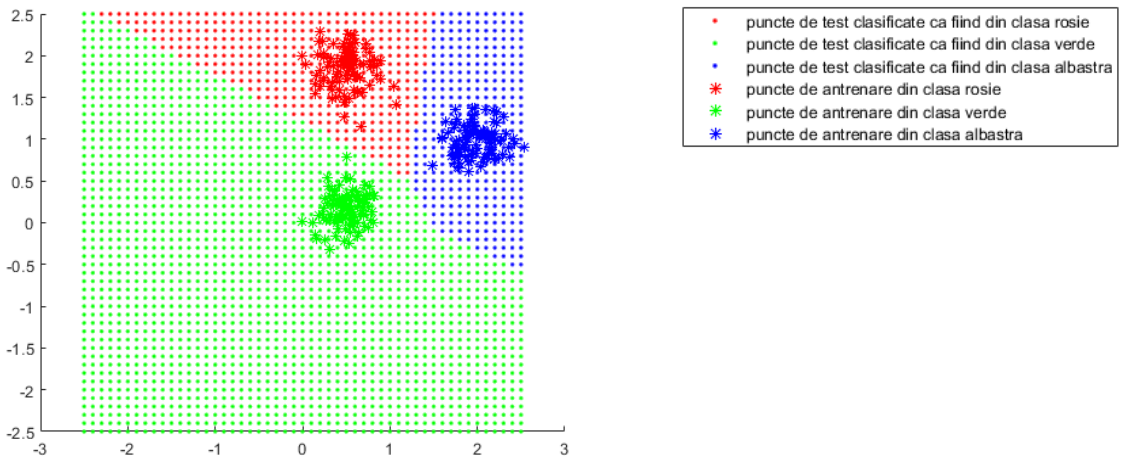
Testare rețea

Pentru testarea rețelei MLP implementate și antrenate vom folosi un set de date nou, despre care nu știm apartenența la clase.

Observații:

- Vectorii din lotul de test au pe ultima poziție valoarea $x_3 = 1$, așa cum s-a procedat și pentru vectorii din lotul de antrenare (datorită parametrului *bias*).
- Pentru obținerea ieșirilor *O*, este suficient să apelăm `MLPnet`, nefiind nevoie să calculăm explicit relațiile.

```
% testare retea
figure(2)
hold on
for i = -2.5 : 0.1 : 2.5
    for j = -2.5 : 0.1 : 2.5
        C = [i,j,x3].'; % intrare
        curent_O = MLPnet(C);
        [val,pos] = max(curent_O); %max din curent_O
        if pos == 3
            plot(i,j,'.','color',[0,0,1])
        end
        if pos == 2
            plot(i,j,'.','color',[0,1,0])
        end
        if pos == 1
            plot(i,j,'.','color',[1,0,0])
        end
    end
end
plot(X(1,      1:P),X(2,    1:P)  ,'r*');
plot(X(1,  P+1:2*P),X(2,  P+1:2*P),'g*');
plot(X(1,2*P+1:3*P),X(2,2*P+1:3*P),'b*');
hold off
```



Aplicația 3.7. Recunoașterea de caractere utilizând rețeaua MLP

În continuare vom prezenta un exemplu didactic de clasificare a literelor folosind rețeaua MLP. Pentru antrenare se vor folosi imagini de dimensiune 7 x 7 pixeli conținând literele O, X, Y și Z (literele sunt cu alb iar fundalul cu negru). De asemenea, tot pentru antrenare se vor folosi imagini cu literele O, X, Y și Z pentru care s-a modificat câte un pixel față de imaginea originală pură.

1. Generare litere pure (fără pixeli distorsionați)

Pentru început se generează imagini de dimensiune 7 x 7 pixeli conținând literele O, X, Y și Z pure (fără pixeli distorsionați).

```
O_alb = [0 0 0 0 0 0 0;
         0 0 1 1 1 0 0;
         0 1 1 0 1 1 0;
         0 1 0 0 0 1 0;
         0 1 1 0 1 1 0;
         0 0 1 1 1 0 0;
         0 0 0 0 0 0 0;];
X_alb = [0 0 0 0 0 0 0;
         0 1 0 0 0 1 0;
         0 0 1 0 1 0 0;
         0 0 0 1 0 0 0;
         0 0 1 0 1 0 0;
         0 1 0 0 0 1 0;
         0 0 0 0 0 0 0;];
Y_alb = [0 0 0 0 0 0 0;
         0 1 0 0 0 1 0;
         0 0 1 0 1 0 0;
         0 0 0 1 0 0 0;
         0 0 0 1 0 0 0;
         0 0 0 1 0 0 0;
         0 0 0 0 0 0 0;];
Z_alb = [0 0 0 0 0 0 0;
         0 1 1 1 1 1 0;
         0 0 0 0 1 0 0;
         0 0 0 1 0 0 0;
         0 0 1 0 0 0 0;
         0 1 1 1 1 1 0;
        0 0 0 0 0 0 0;];
figure(1), imshow(O_alb)
figure(2), imshow(X_alb)
figure(3), imshow(Y_alb)
figure(4), imshow(Z_alb)
```

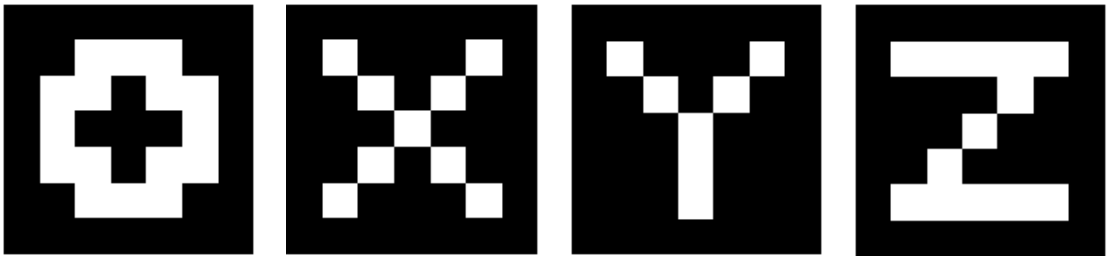


Figura 3.17. Literele O, X, Y și Z pure (fără pixeli distorsionați)

2. Generarea bazei de date (intrări și ieșiri dorite)

Se generează matricea vectorilor de antrenare *X* și cea a ieșirilor dorite *D* pentru rețeaua MLP, care să conțină atât imaginile pure O, X, Y și Z (alb pe fundal negru) cât și toate imaginile cu un pixel alterat față de imaginile pure.

Fiecare vector de antrenare pentru rețeaua MLP este de fapt o imagine vectorizată prin copierea una sub alta a coloanelor imaginii. De exemplu, vectorizarea imaginilor pure și alăturarea vectorilor rezultați în matricea *X* poate fi obținută astfel:

$$vectO = \begin{bmatrix} O_{alb}(1,1) \\ O_{alb}(2,1) \\ \dots \\ O_{alb}(7,1) \\ O_{alb}(1,2) \\ O_{alb}(2,2) \\ \dots \\ O_{alb}(7,2) \\ \dots \\ \dots \\ O_{alb}(7,7) \end{bmatrix} \quad vectX = \begin{bmatrix} X_{alb}(1,1) \\ X_{alb}(2,1) \\ \dots \\ X_{alb}(7,1) \\ X_{alb}(1,2) \\ X_{alb}(2,2) \\ \dots \\ X_{alb}(7,2) \\ \dots \\ \dots \\ X_{alb}(7,7) \end{bmatrix} \quad vectY = \begin{bmatrix} Y_{alb}(1,1) \\ Y_{alb}(2,1) \\ \dots \\ Y_{alb}(7,1) \\ Y_{alb}(1,2) \\ Y_{alb}(2,2) \\ \dots \\ Y_{alb}(7,2) \\ \dots \\ \dots \\ Y_{alb}(7,7) \end{bmatrix} \quad vectZ = \begin{bmatrix} Z_{alb}(1,1) \\ Z_{alb}(2,1) \\ \dots \\ Z_{alb}(7,1) \\ Z_{alb}(1,2) \\ Z_{alb}(2,2) \\ \dots \\ Z_{alb}(7,2) \\ \dots \\ \dots \\ Z_{alb}(7,7) \end{bmatrix}$$

$$X = [vectO, vectX, vectY, vectZ] = \begin{bmatrix} vectO(1) & vectX(1) & vectY(1) & vectZ(1) \\ vectO(2) & vectX(2) & vectY(2) & vectZ(1) \\ \dots & \dots & \dots & \dots \\ vectO(49) & vectX(49) & vectY(49) & vectZ(49) \end{bmatrix}$$

Matricea ieșirilor dorite *D* va fi:

$$D = [D_O, D_X, D_Y, D_Z] = \begin{bmatrix} 1\,0\,0\,0 \\ 0\,1\,0\,0 \\ 0\,0\,1\,0 \\ 0\,0\,0\,1 \end{bmatrix}$$

Pentru a genera vectorii cu câte un pixel alterat se face întâi vectorizarea, apoi se modifică pixelul și se alătură vectorii modificați în matricea *X* iar ieșirile corespunzătoare în matricea *D*.

```
X = [O_alb(:),X_alb(:),Y_alb(:),Z_alb(:)];
D = [[1;0;0;0],[0;1;0;0],[0;0;1;0],[0;0;0;1]];
for i=1:49
        currO = O_alb(:);
        currO(i) = 1-currO(i);
        currX = X_alb(:);
        currX(i) = 1-currX(i);
        currY = Y_alb(:);
        currY(i) = 1-currY(i);
        currZ = Z_alb(:);
        currZ(i) = 1-currZ(i);
        X = [X,currO,currX,currY,currZ];
        D = [D,[1;0;0;0],[0;1;0;0],[0;0;1;0],[0;0;0;1]];
end
```

Baza de date va avea 200 de imagini (câte 50 pentru fiecare literă: imaginea pură + 49 de imagini alterate). Deoarece fiecare imagine este vectorizată și transformată în vector coloană, matricea *X* va avea 49 de linii și 200 de coloane. Deoarece sunt 4 litere (deci 4 clase) matricea ieșirilor dorite *D* va avea 4 linii și 200 de coloane.

3. Antrenare rețea MLP

Se antrenează o rețea MLP cu un strat ascuns având 10 neuroni folosind matricea vectorilor de antrenare *X* și cea a ieșirilor dorite *D*. Lotul de instruire conține 70% dintre vectorii matricei *X* iar lotul de validare conține 30% dintre vectorii lui *X*.

```
N = 10; % N = numar de neuroni pe stratul ascuns
MLPnet = feedforwardnet(N);
MLPnet.divideParam.trainRatio = 0.7;
MLPnet.divideParam.valRatio = 0.3;
MLPnet.divideParam.testRatio = 0;
% antrenare retea MLP
MLPnet = train(MLPnet,X,D);
```

Arhitectura rețelei arată ca în figura de mai jos.

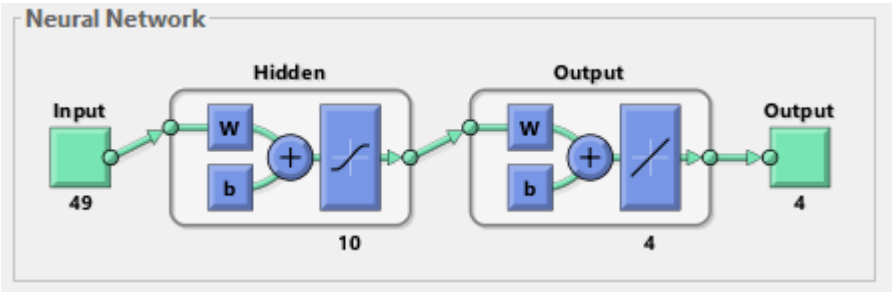


Figura 3.18. Arhitectura rețelei MLP

Conform parametrilor din fereastra *nntraintool,* algoritmul de antrenare al rețelei a încetat atunci când s-a îndeplinit criteriul *Validation Cheks (Numărul de verificări de validare).* Așa cum se poate vedea, s-a ajuns la 6 iterații succesive (valoarea *default* maximă) pentru care performanța pe lotul de validare nu scade.

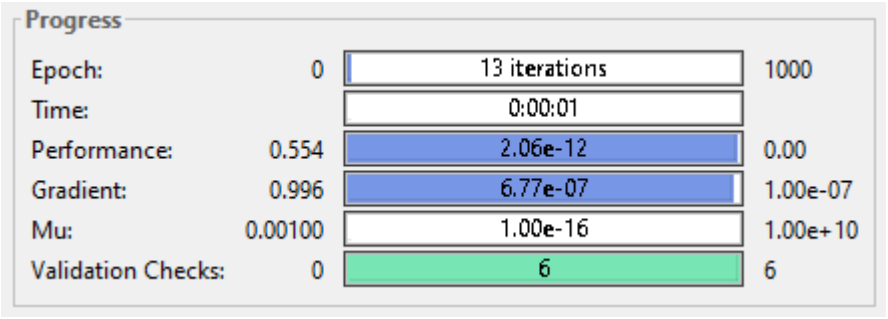


Figura 3.19. Criterii de oprire

Conform graficului evoluției erorii `mse`, cel mai bun scor de clasificare (cea mai mică eroare `mse`) pentru lotul de validare s-a obținut pentru epoca 7; algoritmul mai rulează însă încă 6 epoci (*numărul de verificări de validare*) după care se oprește.

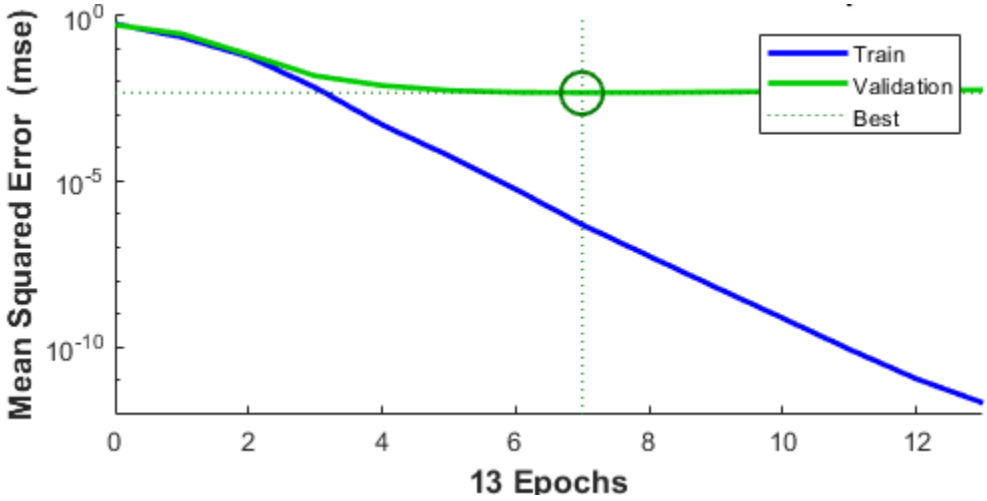


Figura 3.20. Evoluția parametrului `mse` în funcție de epocă

4. Evaluarea performanțelor pentru lotul de antrenare

Pentru evaluarea performanțelor rețelei vom folosi în primă fază cei 200 de vectori folosiți la antrenare. Această testare nu este însă tocmai relevantă pentru că testăm rețeaua cu exact ceea ce a învățat. Dar într-o primă fază este bine să știm și cât reușește să recunoască rețeaua din ceea ce a învățat, deoarece un scor mic de clasificare pe lotul de antrenare presupune o antrenare nu tocmai bună și atunci trebuie reluată etapa de antrenare.

Când antrenarea a luat sfârșit, ponderile nu se mai modifică, însă vectorii de antrenare pot fi testați, în sensul verificării dacă, cu ponderile finale, rețeaua recunoaște corect caracterele învățate.

În cazul de față, ieșirile obținute vor fi $O = [O_1, O_2, \dots, O_k, \dots, O_{200}]$. Așa cum am mai menționat, importante nu sunt valorile fiecărui vector O_k din matricea O, cât mai degrabă poziția pe care se găsește valoarea maximă din fiecare vector O_k. Numărul poziției pe care se găsește valoarea maximă din vectorul O_k indică clasa din care rețeaua prezice că face parte vectorul X_k.

Se vor afișa numărul de erori, scorul de clasificare corectă și matricea de confuzie.

```
% performante pentru lotul de antrenare
O = sim(MLPnet,X);
[valO,posO] = max(O);
[valD,posD] = max(D);
NrEroriTrain = length(find(posO~=posD))
EroareProcentualaTrain = NrEroriTrain/size(X,2)*100;
ScorClasificareTrain = 100 - EroareProcentualaTrain
% matricea de confuzie pentru lotul de antrenare
ConfuzieTrain = confusionmat(posD, posO)
```

```
NrErori =
     2

ScorClasificareTrain =
    99

ConfuzieTrain =
    50     0     0     0
     0    49     1     0
     0     1    49     0
     0     0     0    50
```

Pentru exemplul de mai sus, rețeaua clasifică eronat 2 imagini din lotul de antrenare, ceea ce înseamnă un scor de clasificare greșită de 1%, adică un scor de clasificare corectă de 99%. Conform matricei de confuzie, o imagine din clasa 2 (*X*) a fost clasificată greșit în clasa 3 (*Y*) și o imagine din clasa 3 (*Y*) a fost clasificată greșit în clasa 2 (*X*).

5. Evaluarea performanțelor pentru lotul de testare

Pentru o testare corectă a rețelei ar trebui folosite alte imagini despre care rețeaua nu știe nimic, dar pentru care se cunosc clasele din care ar trebui să facă parte. Se va crea un set de test (*T*) compus din imagini cu 2 pixeli alterați față de imaginile pure originale. Ieșirile dorite sunt salvate în matricea (*Z*).

```
% creare lot de test
Z = [];
T = [];
for i=1:48
    for j = i+1:49
         currO = O_alb(:);
         currO(i) = 1-currO(i);
         currO(j) = 1-currO(j);
         currX = X_alb(:);
         currX(i) = 1-currX(i);
         currX(j) = 1-currX(j);
         currY = Y_alb(:);
         currY(i) = 1-currY(i);
         currY(j) = 1-currY(j);
         currZ = Z_alb(:);
         currZ(i) = 1-currZ(i);
         currZ(j) = 1-currZ(j);
         Z = [Z,currO,currX,currY,currZ];
         T = [T,[1;0;0;0],[0;1;0;0],[0;0;1;0],[0;0;0;1]];
    end
end
```

Se vor genera 4704 imagini având câte 2 pixeli alterați (1176 de imagi alterate pentru fiecare imagine). Pentru lotul de test performanțele clasificării vor fi:

```
Y = sim(MLPnet,Z);
[valY,posY] = max(Y);
[valT,posT] = max(T);
NrEroriTest = length(find(posY~=posT))
EroareProcentualaTest = NrEroriTest/size(Z,2)*100;
ScorClasificareTest = 100 - EroareProcentualaTest
% matricea de confuzie pentru lotul de test
ConfuzieTest = confusionmat(posT, posY)
```

```
NrEroriTest =
     44

ScorClasificareTest =
   99.0646

ConfuzieTest =
        1175           0           1           0
           0        1174           2           0
           0           2        1156          18
           0           0          21        1155
```

In funcție de cât de bine s-a antrenat rețeaua, rezultatele pot fi mai bune sau mai slabe. Pentru îmbunătățirea clasificării, antrenarea poate fi reluată cu aceiași parametri (dar inițializare a ponderilor diferită) sau cu parametri diferiți (număr diferit de neuroni pe stratul intermediar, criteriile de oprire mai stricte, etc.).

Aplicația 3.8. Recunoașterea cifrelor utilizând rețeaua MLP

Să se antreneze o rețea MLP cu două straturi ascunse (16 neuroni pe primul strat intermediar și 8 neuroni pe cel de-al doilea strat intermediar), pentru recunoașterea cifrelor 1 și 2 din baza de date MNIST (pentru o execuție mai rapidă a programului, pentru antrenare se vor folosi doar primele 1000 de imagini din baza de date MNIST: 500 de imagini cu *cifra 1* și 500 de imagini cu *cifra 2*).

Obs: Detalii despre baza de date MNIST găsiți în *Anexa 2.*

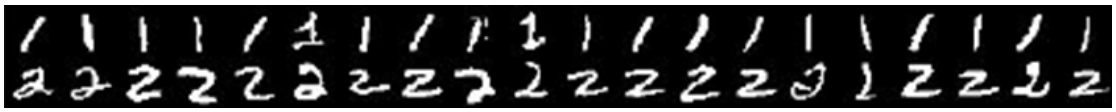


Figura 3.21. Selecție de imagini cu cifrele 1 și 2 din baza de date MNIST

Implementarea în MATLAB este următoarea:

1. Pregătirea setului de antrenare (intrări și ieșiri dorite)

```
database_train = xlsread('mnist_train_5k.xlsx');
cifreInput = [1, 2];
NrCifre = length(cifreInput);
NrImaginiPerCifra = 500;
% se cauta in baza de date pozitiile din fisierul excel pe care se afla cifra 1
cifra_index1 = find((database_MNIST(:,1)==cifreInput(1)));
% se cauta in baza de date pozitiile din fisierul excel pe care se afla cifra 2
cifra_index2 = find((database_MNIST(:,1)==cifreInput(2)));
cifre_index_train = [cifra_index1(1:NrImaginiPerCifra);
cifra_index2(1:NrImaginiPerCifra)];
% baza de date ce va fi folosita pentru antrenare nu foloseste si prima coloana
database_train = database_MNIST(cifre_index_train,2:end)';
database_train = database_train/255;
% iesiri dorite pentru lotul de antrenare
targets_train = zeros(NrCifre, size(database_train,2));
targets_train(1,1:NrImaginiPerCifra)=1;
targets_train(2,NrImaginiPerCifra+1:2*NrImaginiPerCifra)=1;
```

Lotul de antrenare `database_train` este o matrice cu 784 de linii (deoarece dimensiunea unei imagini este de 28 x 28 pixeli) și 1000 de coloane (deoarece se folosesc 1000 de imagini cu cele 2 cifre).

Matricea ieșirilor dorite `targets_train` are 2 linii și 1000 de coloane. Dacă un vector de antrenare este reprezentat de *cifra 1* vectorizată, atunci eticheta asociată (coloana din `targets`) va fi $[1, 0]^T$; dacă vectorul de antrenare este *cifra 2* vectorizată, atunci eticheta asociată va fi $[0, 1]^T$.

2. Antrenarea rețelei MLP

Se va implementa o rețea MLP cu două straturi ascunse: 16 neuroni pe primul strat ascuns și 8 neuroni pe cel de-al doilea strat ascuns. Pentru antrenare, cei 1000 de vectori vor fi împărțiți random astfel: 70% pentru instruire și 30% pentru validare. După antrenarea rețelei aceasta va fi salvată în variabila `MLPnet`.

```
% se antreneaza o retea MLP cu 2 straturi intermediare
% 16 neuroni pentru primul strat ascuns
NrNeuroniIntermediari1 = 16;
% 8 neuroni pentru al doilea strat ascuns
NrNeuroniIntermediari2 = 8;
MLPnet = feedforwardnet([NrNeuroniIntermediari1, NrNeuroniIntermediari2]);
% pentru instruire se folosesc 70% dintre vectori
MLPnet.divideParam.trainRatio = 0.70;
% pentru validare se folosesc 30% dintre vectori
MLPnet.divideParam.valRatio = 0.30;
MLPnet.divideParam.testRatio = 0.00;
MLPnet = train(MLPnet,database_train,targets_train);
% se salveaza reteaua MLPnet
save('MLPnet','MLPnet')
```

Arhitectura rețelei MLP implementată mai sus arată astfel:

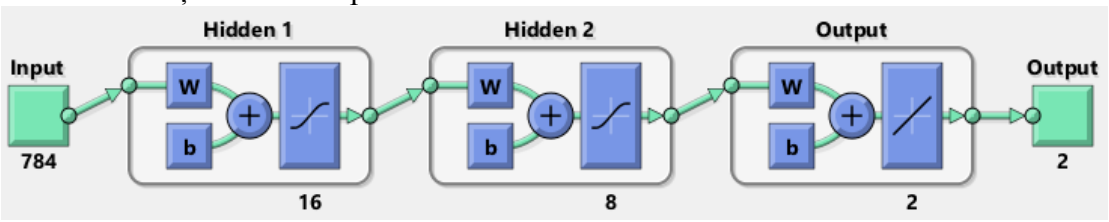


Figura 3.22. Arhitectura rețelei MLP cu 2 straturi

Rețeaua va avea pe stratul de intrare 784 de noduri (câte unul pentru fiecare pixel din imagine) și 2 neuroni pe stratul de ieșire deoarece sunt 2 clase (*cifra 1* și *cifra 2*). Antrenarea rețelei s-a oprit după 16 epoci.

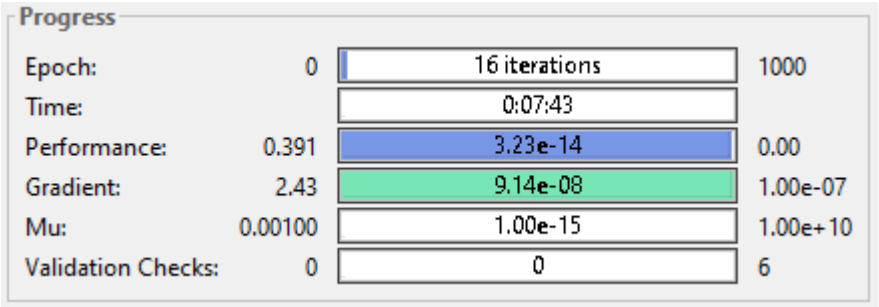


Figura 3.23. Progresul antrenării rețelei

Evoluția parametrului `mse` în timpul antrenării este următoarea.

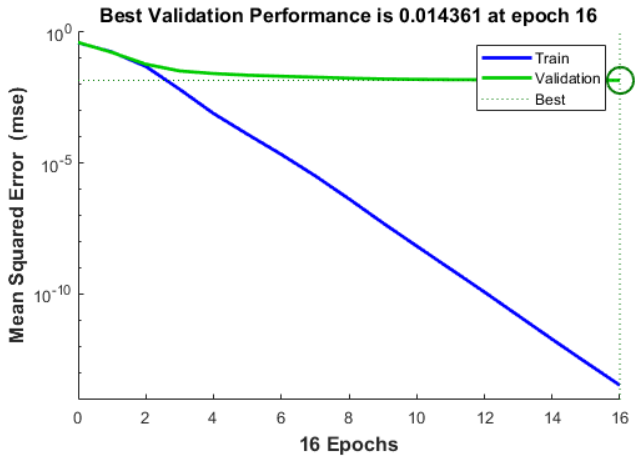


Figura 3.24. Evoluția parametrului `mse` în timpul antrenării

3. Performanțele antrenării

```
iesire_retea = MLPnet(database_train);
[max_iesire_retea, pozO] = max(iesire_retea);
[max_target, pozT] = max(targets_train);
% matricea de confuzie pentru lotul de antrenare
C_train = confusionmat(pozT, pozO);
confusionchart(C_train)
% acuratetea clasificarii pentru lotul de antrenare
acuratete_train = sum(diag(C_train))/sum(sum(C_train)) * 100;
disp(['acuratetea clasificarii pe lotul de antrenare este = ',
num2str(acuratete_train),'%'])
```

```
acuratetea clasificarii pe lotul de antrenare este = 99.5%
```

Figura 3.25. Matricea de confuzie pentru lotul de antrenare

Așa cum se poate observa din matricea de confuzie, o singură imagine cu *cifra 1* din lotul de antrenare nu este corect clasificată și 4 imagini cu *cifra 2* sunt incorect clasificate ca fiind *cifra 1*. Rezultă asfel un scor de clasificare pentru lotul de antrenare de 99.5%

Așa cum am mai menționat însă, pentru ca scorul de clasificare să fie relevant, trebuie să testăm rețeaua pe un set de date care nu a fost folosit la antrenare. Vom testa în continuare rețeaua pe un set de date nou, conținând 627 de imagini cu *cifra 1* și 491 de imagini cu *cifra 2*.

4. Pregătirea setului de testare.

```
cifre_index_test =  [cifra_index1(NrImaginiPerCifra+1:length(cifra_index1));
cifra_index2(NrImaginiPerCifra+1:length(cifra_index2))];
database_test = database_MNIST(cifre_index_test,2:end)';
database_test = database_test/255;
% % iesirile cunoscute pentru lotul de testare, pentru a putea calcula acuratetea
targets_test = zeros(NrCifre, size(database_test,2));
targets_test(1,1:length(cifra_index1)-NrImaginiPerCifra)=1;
targets_test(2,length(cifra_index1)-NrImaginiPerCifra+1:size(database_test,2))=1;
```

5. Testarea rețelei MLP cu noul set de date.

```
iesire_retea = MLPnet(database_test);
[max_iesire_retea, pozO] = max(iesire_retea);
[max_target, pozT] = max(targets_test);
% matricea de confuzie pentru lotul de test
C_test = confusionmat(pozT, pozO);
confusionchart(C_test)
% acuratetea clasificarii pentru lotul de testare
acuratete_test = sum(diag(C_test))/sum(sum(C_test)) * 100;
disp(['acuratetea clasificarii pe lotul de test este = ',
num2str(acuratete_train),'%'])
```

```
acuratetea clasificarii pe lotul de antrenare este = 97.94%
```

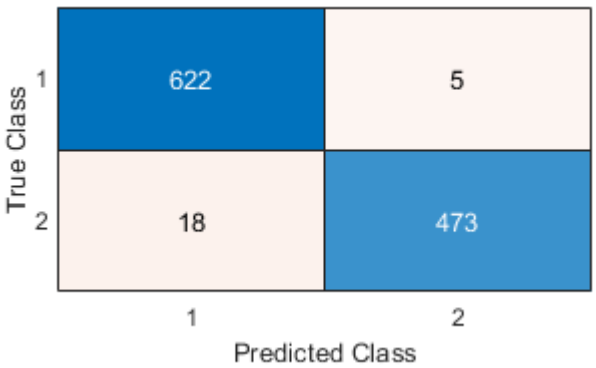


Figura 3.26. Matricea de confuzie pentru lotul de antrenare

Așa cum se poate observa din matricea de confuzie, 5 imagini cu *cifra 1* nu sunt corect clasificate și 18 imagini cu *cifra 2* sunt incorect clasificate ca fiind *cifra 1*. Rezultă asfel un scor de clasificare corectă pentru lotul de test de 97.94%.

3.4.5. Implementarea rețelei MLP cu ajutorul *toolbox*-urilor MATLAB

Pentru implementarea rapidă a aplicațiilor folosind rețele neurale, MATLAB-ul pune la dispoziție (pe lângă funcțiile optimizate dedicate rețelelor neurale) și *toolbox*-uri cu interfețe grafice și baze de date reale. În continuare, pentru exemplificarea utilizării *toolbox*-urilor pentru antrenarea și testarea unei rețele MLP, se va folosi baza de date *Wine* (descrierea acestei baze de date se găsește în *Anexa 1*).

Pentru versiunea de MATLAB *2019a*, se accesează *tab*-ul *Apps*, categoria *Machine Learning and Deep Learning*, *toolbox*-ul *Neural Net Pattern Recognition*.

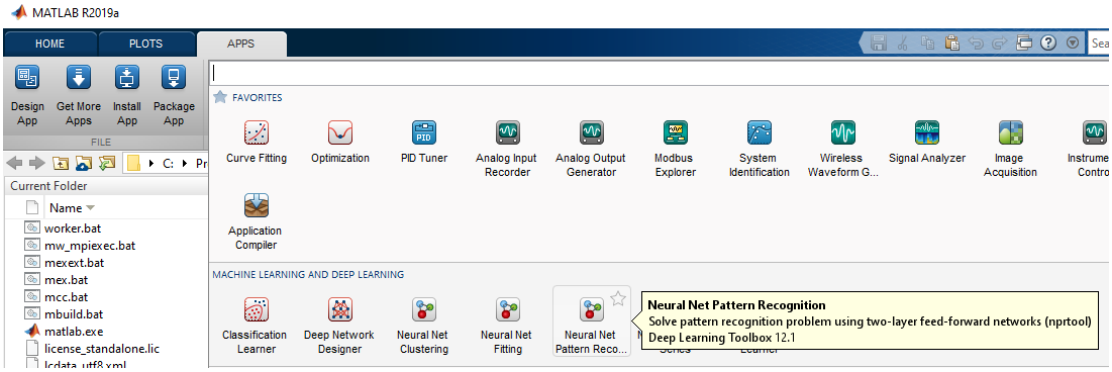


Figura 3.27. *Toolbox*-ul *Neural Net Pattern Recognition*

Toolbox-ul *Neural Net Pattern Recognition* va deschide următoarea interfață grafică.

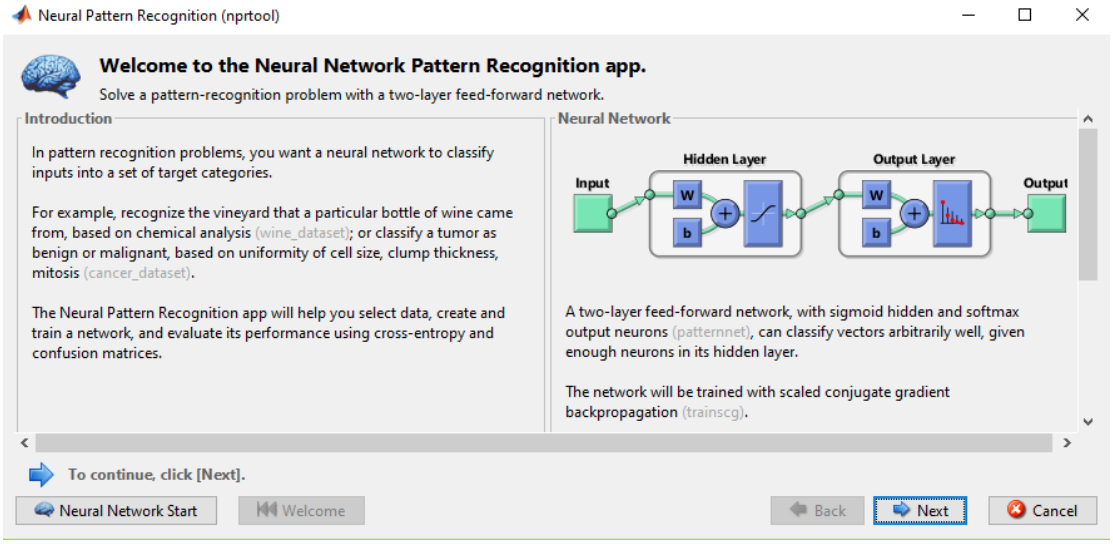


Figura 3.28. *Toolbox*-ul *Neural Net Pattern Recognition (fereastra nprtool)*

În continuare, pentru a încărca baza de date pentru care se dorește antrenarea rețelei se va alege opțiunea *Next*. Se va deschide următoarea fereastră:

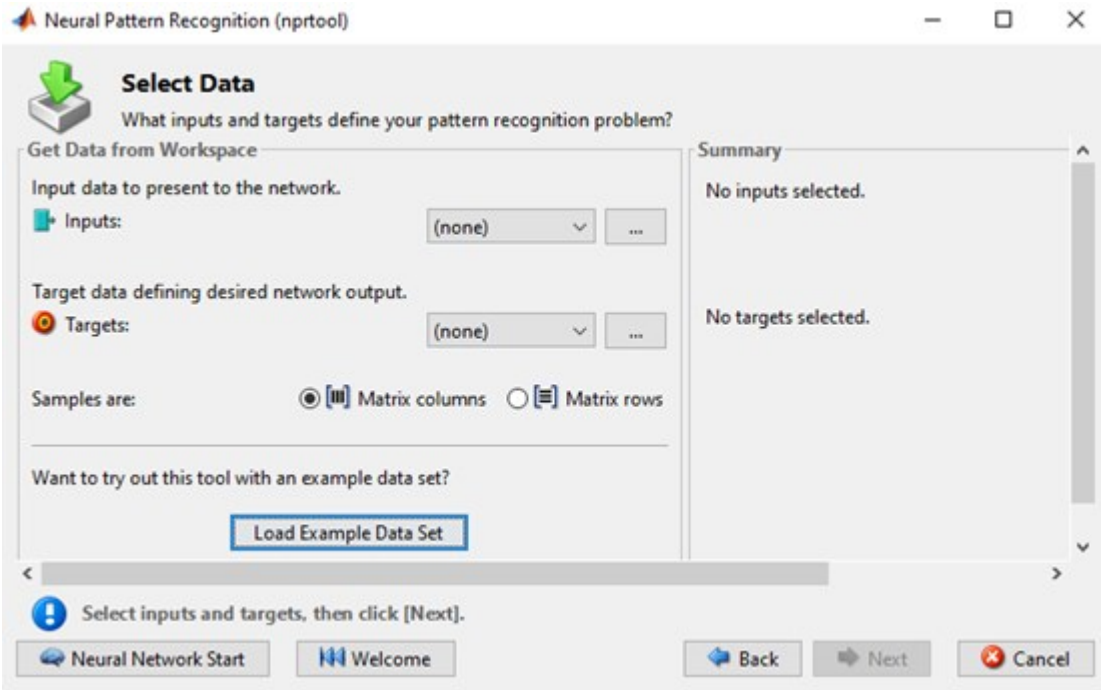


Figura 3.29. Încărcarea bazei de date

Incărcarea datelor se poate face din fereastra *Workspace*, dacă acestea există acolo. Dacă nu există, trebuie mai întâi încărcate. Pentru exemplificare vom folosi in continuare baza de date *Wine* care este inclusă în librăria MATLAB-ului.

Pentru utilizarea bazei de date *Wine* se poate folosi comanda `load wine_dataset` care încarcă în *Workspace* două variabile, `wineInputs` şi `wineTargets`, care corespund cu matricele vectorilor de intrare *X* şi a ieşirilor dorite `D` prezentate anterior.

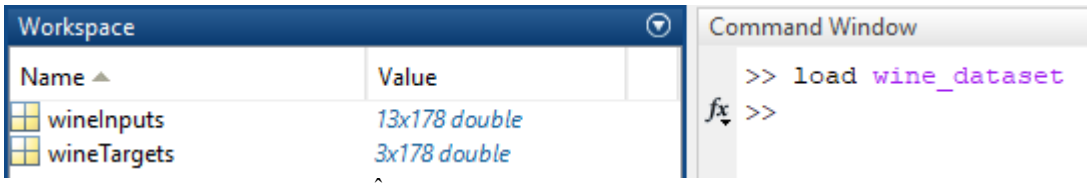


Figura 3.30. Încărcarea în *Workspace* a bazei de date *Wine*

Cele două variabile, `wineInputs` şi `wineTargets` devin acum disponibile pentru utilizare, pentru parametrii *Inputs* şi *Targets*. Pentru *Inputs* (datele folosite pentru antrenare - *X*) vom selecta `wineInputs` iar pentru *Targets* (ieşirele dorite - *D*) variabila `WineTargets`.

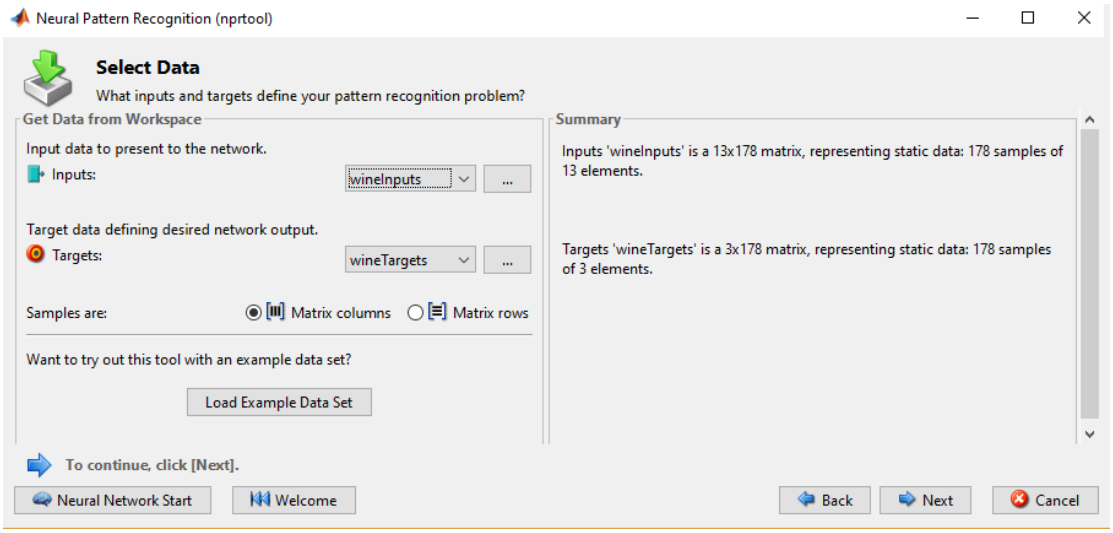


Figura 3.31. Încărcarea bazei de date *Wine* - selectarea intrărilor și ieșirilor dorite pentru antrenare

Alternativ, deoarece baza de date *Wine* este deja în librăria MATLAB-ului, se poate încărca apăsând butonul *Load Example Data Set*, selectând *Wine vintage* și apăsând pe butonul *Insert*. Variabilele sunt încărcate în *Workspace* și în același timp și în interfața *toolbox*-ului.

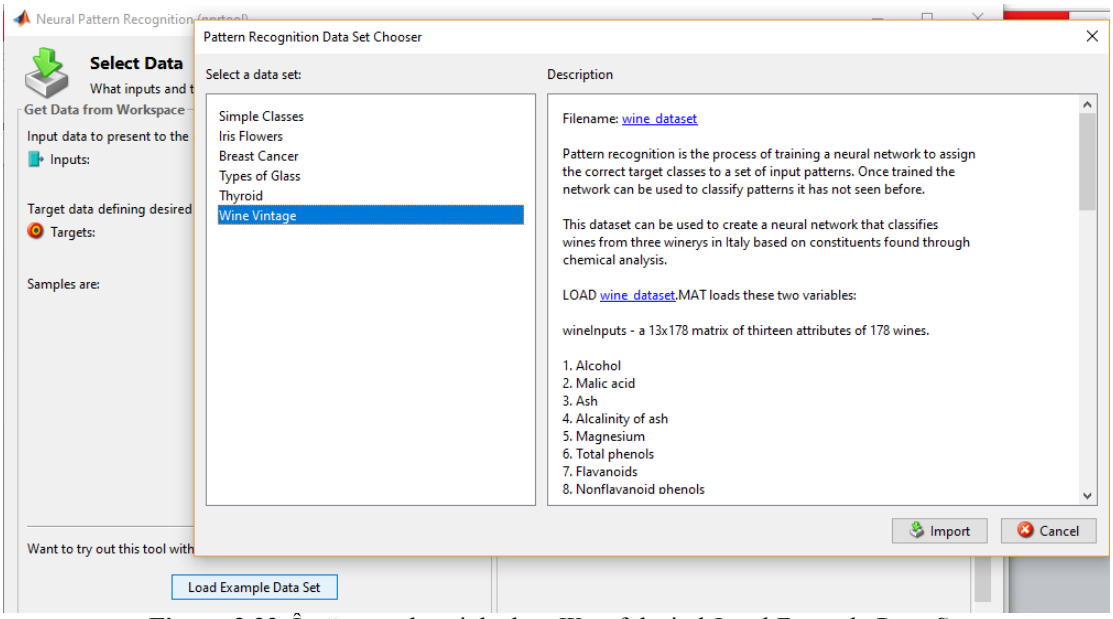


Figura 3.32. Încărcarea bazei de date *Wine* folosind *Load Example Data Set*

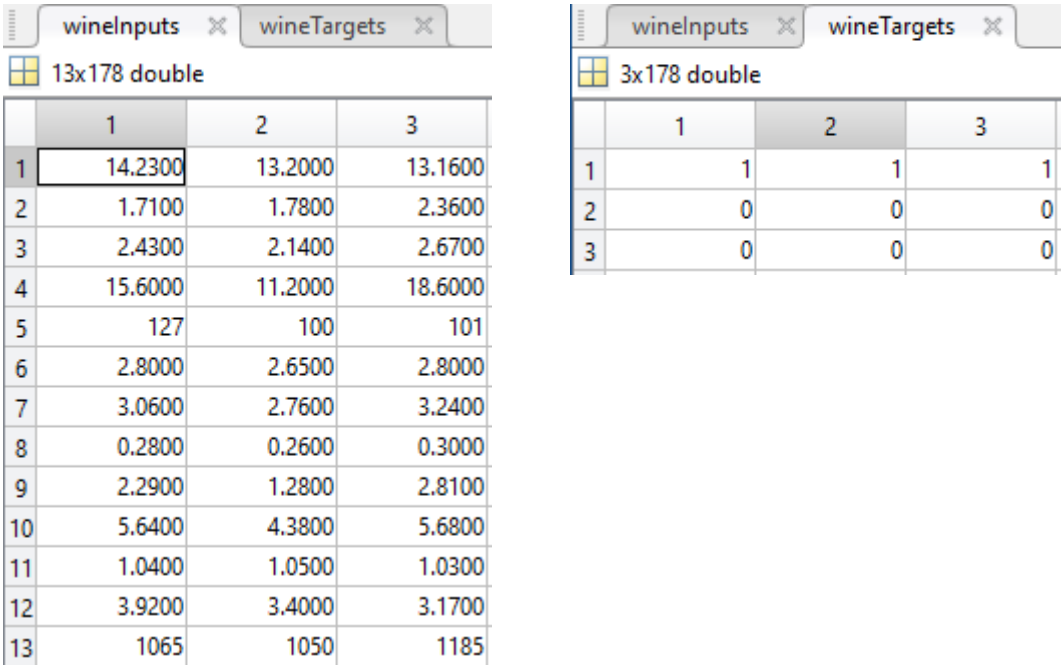
Toolbox-ul oferă posibilitatea ca vectorii să fie așezați în matrice ca vectori coloană (similar cu convenția folosită în formule) sau vectori linie.

Figura 3.33. Modul de aranjare al vectorilor în matrice: pe linii sau pe coloane

Baza de date *Wine* conține 178 de vectori, obținuți prin măsurarea a 13 parametri fizico-chimici ai vinurilor din 3 podgorii. Scopul aplicației este de a antrena o rețea MLP care să identifice podgoria de proveniență a vinurilor. Mai multe informații despre această bază de date găsiți în *Anexa 1*.

Prin urmare, alegând varianta *Matrix columns* de așezare a vectorilor de intrare, se va obține:

- Matricea *Inputs* cu dimensiunea de 13 linii și 178 de coloane. Fiecare coloană va conține cei 13 parametri fizico-chimici ai unui vin.
- Matricea *Targets* cu dimensiunea de 3 linii și 178 de coloane. Fiecare coloană va conține clasa (podgoria) din care face parte un vin. Dacă primul vin face parte din podgoria 1, atunci prima coloană din *Targets* va fi $[1, 0, 0]^T$.



wineInputs | wineTargets

13x178 double

	1	2	3
1	14.2300	13.2000	13.1600
2	1.7100	1.7800	2.3600
3	2.4300	2.1400	2.6700
4	15.6000	11.2000	18.6000
5	127	100	101
6	2.8000	2.6500	2.8000
7	3.0600	2.7600	3.2400
8	0.2800	0.2600	0.3000
9	2.2900	1.2800	2.8100
10	5.6400	4.3800	5.6800
11	1.0400	1.0500	1.0300
12	3.9200	3.4000	3.1700
13	1065	1050	1185

wineInputs | wineTargets

3x178 double

	1	2	3
1	1	1	1
2	0	0	0
3	0	0	0

Figura 3.34. Variabilele `wineInputs` și `wineTargets`

Următorul pas îl reprezintă împărțirea vectorilor în *set de instruire*, *set de validare* și *set de test*. Se va folosi împărțirea implicită realizată de MATLAB și anume:

- *lot de instruire*: conține 70% dintre vectori (124 de vectori)
- *lot de validare*: conține 15% dintre vectori (27 de vectori)
- *lot de test*: conține 15% dintre vectori (27 de vectori)

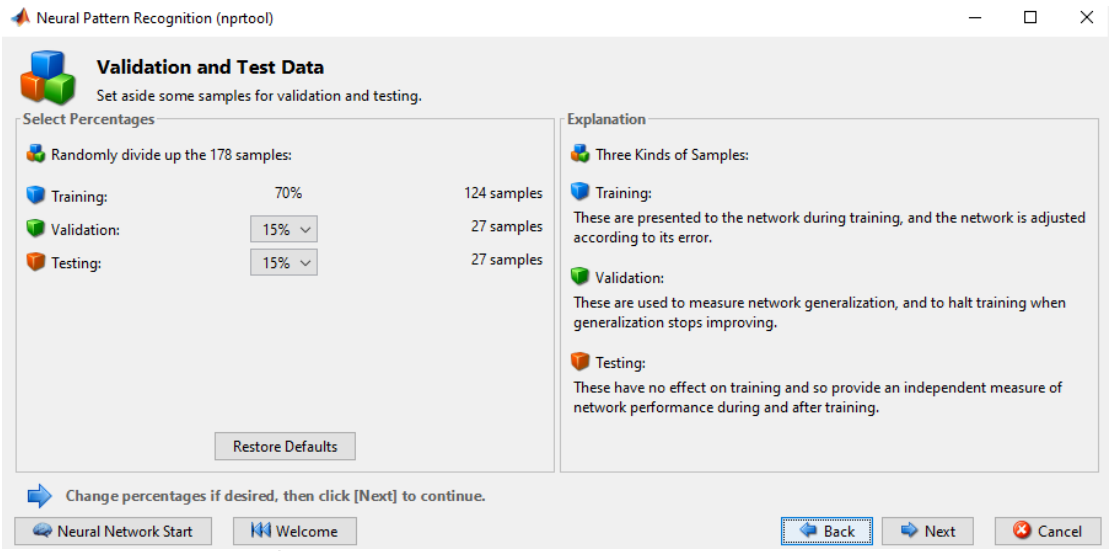


Figura 3.35. Împărțirea datelor în *set de antrenare*, *set de validare* și *set de test*

Următorul pas îl constituie configurarea rețelei MLP, care se rezumă la alegerea numărului de neuroni din stratul intermediar (ascuns); în cazul de față s-au folosit 10 neuroni pentru stratul intermediar. Ca și pentru funcțiile MATLAB, numărul de noduri de intrare și numărul de neuroni de ieșire este determinat automat de dimensiunile matricelor vectorilor de intrare, respectiv a ieșirilor dorite; sunt 13 noduri în stratul de intrare (deoarece fiecare vin are 13 parametri) și 3 neuroni în stratul de ieșire deoarece sunt 3 podgorii (clase). Spre deosebire de folosirea directă a funcțiilor MATLAB, *toolbox*-ul nu permite configurarea rețelelor MLP cu mai multe straturi ascunse și nici modificarea parametrilor rețelei precum criteriile de oprire a antrenării sau funcția de activare pentru neuronii de pe stratul intermediar.

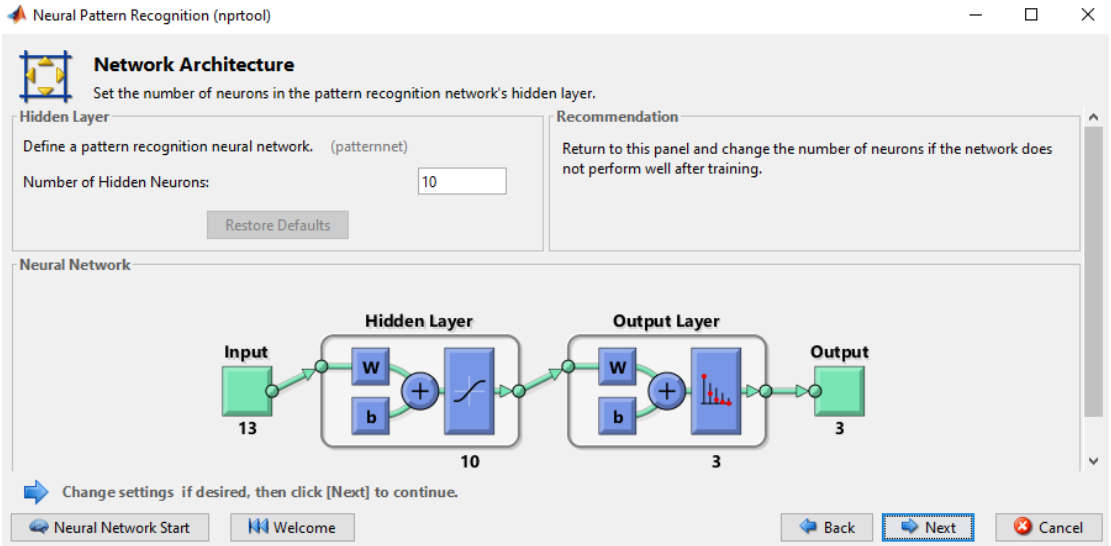


Figura 3.36. Arhitectura rețelei MLP

Următorul pas este cel de antrenare a rețelei.

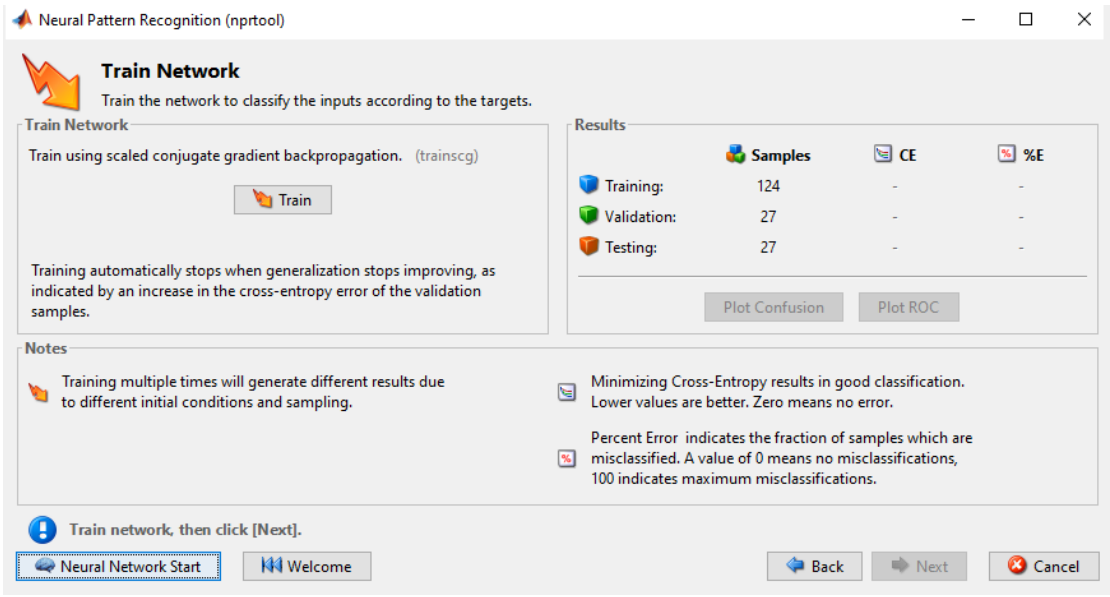


Figura 3.37. Antrenarea rețelei MLP

Prin apăsarea butonului de *Train* este deschisă fereastra *nntraintool* cu parametrii *default* setați de interfață.

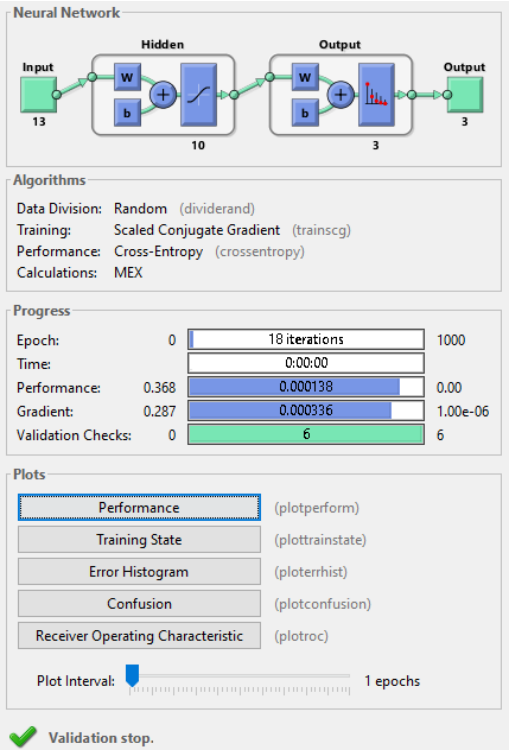


Figura 3.38. Fereastra *nntraintool* deschisă în urma antrenării

In urma antrenării se pot vedea rezultatele clasificării pentru cele 3 seturi de date.

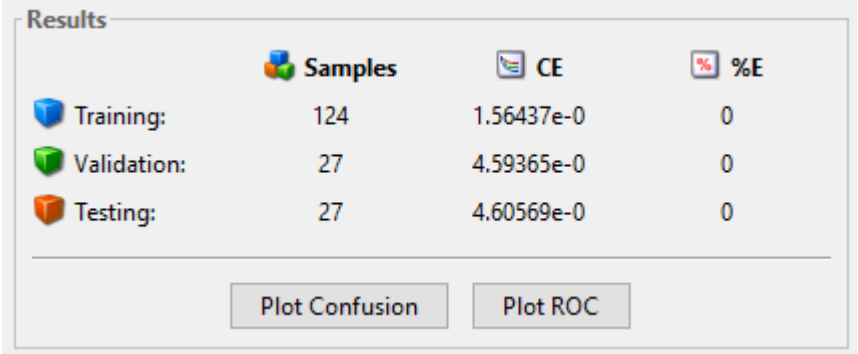


Figura 3.39. Rezultatele clasificării pentru cele 3 seturi de date (*antrenare, validare, test*)

Următorul pas oferă posibilitatea de a relua etapa de antrenare (*Train Again*), de a reconfigure rețeaua alegând un alt număr de neuroni pentru stratul ascuns (*Adjust Network Size*) de a folosi un alt set de date pentru antrenare (*Import Larger Data Set*) sau de a testa rețeaua deja antrenată pe un alt lot de test (*Optionally perform additional tests*) înainte de generarea fișierului **.m*.

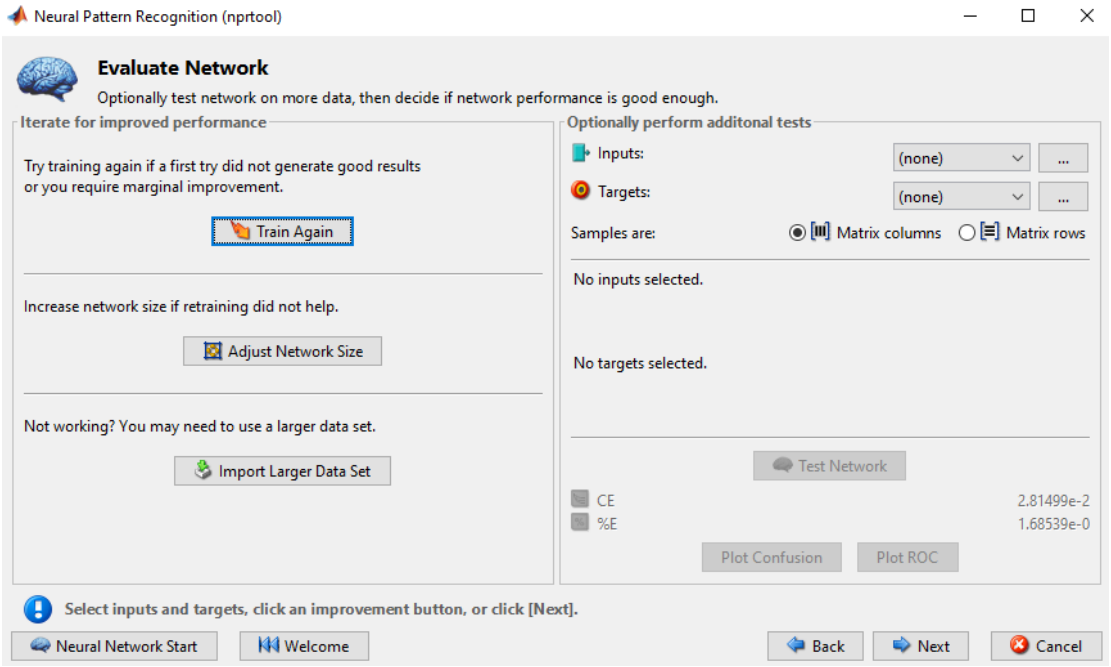


Figura 3.40. Evaluare rețea MLP

In fine, ultimul pas îl reprezintă salvarea rezultatelor și a rețelei în *Workspace*, pentru a fi folosită mai departe de scripturi MATLAB, pentru a fi salvată în fisiere **.mat*, etc. De asemenea, din acest ecran se pot genera fișiere **.m* care să antreneze rețeaua la fel cum o face aplicația, fără a mai fi nevoie de intervenția utilizatorului la fiecare pas.

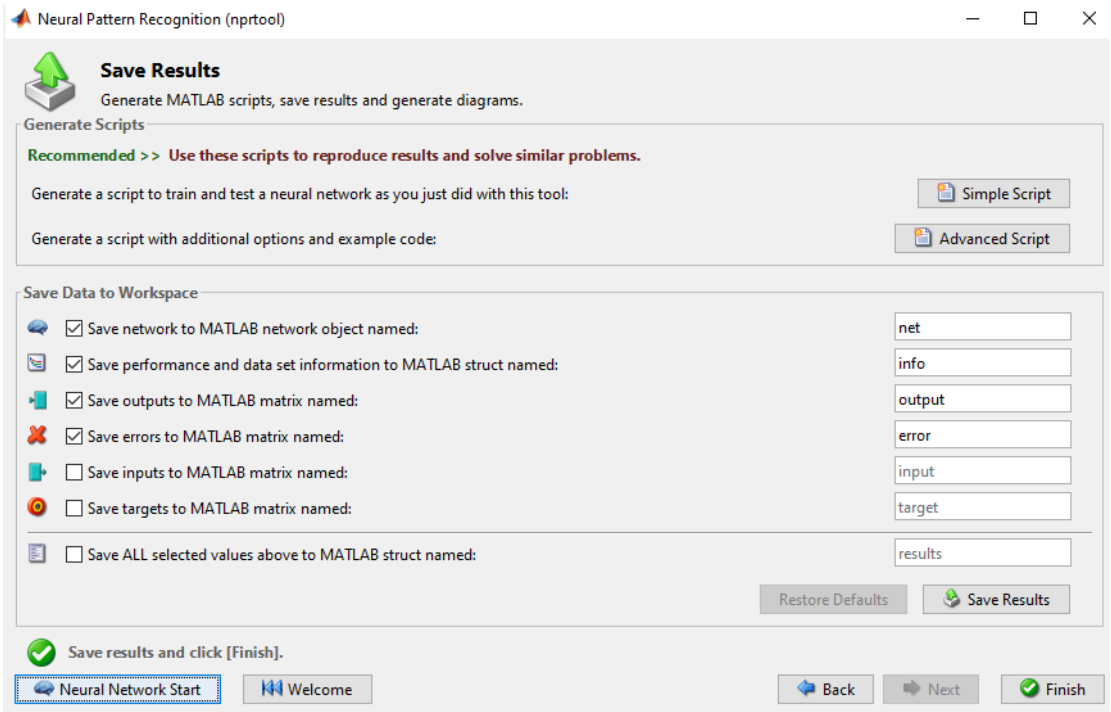


Figura 3.41. Generare cod sursă și salvare rezultate

Exemplu de script generat automat de interfață, folosind opțiunea *Simple Script*.

```
x = wineInputs; %    wineInputs - input data.
t = wineTargets; %   wineTargets - target data.

% Choose a Training Function
trainFcn = 'trainscg';  % Scaled conjugate gradient backpropagation.
% Create a Pattern Recognition Network
hiddenLayerSize = 10;
net = patternnet(hiddenLayerSize, trainFcn);
% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;
% Train the Network
[net,tr] = train(net,x,t);
% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y)
tind = vec2ind(t);
yind = vec2ind(y);
percentErrors = sum(tind ~= yind)/numel(tind);
% View the Network
view(net)
```

Capitolul 4

Metode statistice și neurale folosite pentru clustering

Clasificarea nesupervizată (clustering) atacă problema realizării împărțirii unei mulțimi în clase, astfel încât în interiorul unei clase să existe similaritate cât mai mare între membri, iar între clase să existe diferențe cât mai mari. În cazul clasificării nesupervizate, nu este nevoie de antrenare, algoritmul se auto-organizează fără intervenție externă.

4.1. Algoritmul *K-means* clustering

4.1.1. Introducere teoretică

Pentru o mulţime de N vectori p-dimensionali ce trebuie împărţiţi în K clase (grupuri, clustere), algorimul poate fi descris astfel:

a) *Iniţializare :* se atribuie în mod aleator cei N vectori în K clase şi se calculează mediile claselor (μ_k); o altă variantă este iniţializarea aleatoare a centrelor claselor;
b) pentru fiecare dintre cei N vectori se calculează distanţa faţă de centrele claselor şi se atribuie vectorul X_i clasei k pentru care $d_k = \|X_i - \mu_k\|^2$ este minimă;
c) după parcurgerea completă a setului de vectori se recalculează mediile claselor
d) dacă se îndeplinește una dintre *condițiile de stop*, atunci algoritmul se încheie; dacă nu, se reia algoritmul de la *pasul b*.

Condiții de stop:

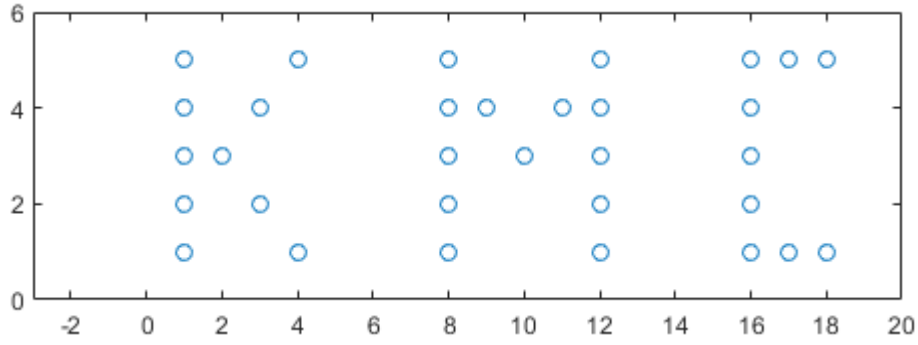
- dacă la sfârșitul unei noi iterații niciun vector nu a fost mutat în altă clasă (sau media niciunei clase nu s-a modificat).
- dacă funcția obiectiv J ce trebuie minimizată este mai mică decât un prag prestabilit.

$$J = \sum_{j=1}^{K} J_j = \sum_{j=1}^{K} \sum_{X_i \in \omega_j} \|X_i - \mu_j\|^2 \tag{4.1}$$

Observații:

- Numărul de iteraţii este dependent de alegerea partiției iniţiale a vectorilor, precum şi de organizarea intrinsecă a acestora.
- Pentru o mai bună clasificare, algoritmul poate fi rulat de mai multe ori și se alege împărțirea în clase pentru care s-a obținut funcția obiectiv J minimă.

Exemplu 4.1. Fie cei 32 de vectori 2D reprezentați grafic în figura de mai jos.



Se dorește împărțirea acestor vectori în 3 clase cât mai bine separate. În urma rulării algoritmului *k-means clustering* se obține:

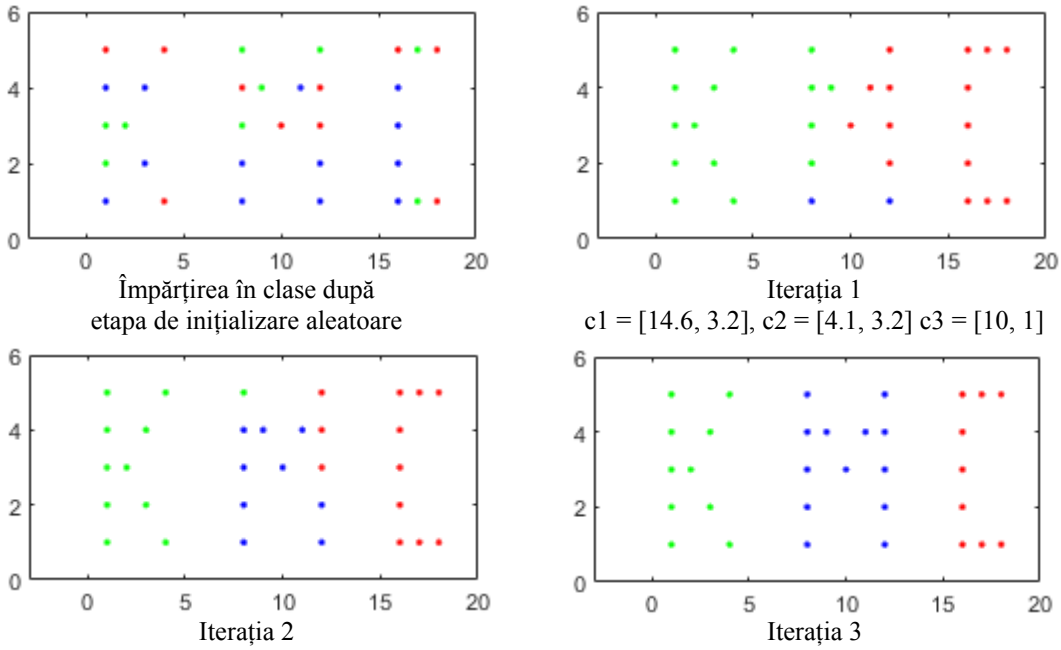


Figure 4.1. Clustering cu k-means

4.1.2. Utilizarea metodei *k-means clustering* în MATLAB

Pentru a realiza clasificarea în MATLAB folosind metoda nesupervizată *k-means clustering* se poate folosi funcția `kmeans`.

Sintaxă: **[**`Ind, centre`**]** = `kmeans(X, K, 'nr_rulari',val_nr)` unde:

- `X` = matricea vectorilor ce se dorește a fi clasificați; fiecare vector este salvat pe câte o linie din matricea `X;`
- `K` = numărul de clase în care se face clustering;
- `nr_rulari` = de câte ori să se repete algoritmul;
- `Ind` = indicii vectorilor după clustering; `Ind(3) = 1`, înseamnă că al 3-lea vector din `X` face parte din *clasa 1* ;
- `centre` = matricea centrelor claselor după clustering; centrele claselor sunt salvate pe linii

Obs: pentru sintaxa completă `>> help kmeans`

Aplicația 4.1. Fie vectorii:

$A = \begin{bmatrix}1\\5\end{bmatrix}$, $B = \begin{bmatrix}5\\5\end{bmatrix}$, $C = \begin{bmatrix}1\\1\end{bmatrix}$, $D = \begin{bmatrix}5\\1\end{bmatrix}$, $E = \begin{bmatrix}4\\2\end{bmatrix}$, $F = \begin{bmatrix}8\\2\end{bmatrix}$, $G = \begin{bmatrix}4\\-2\end{bmatrix}$, $H = \begin{bmatrix}8\\-2\end{bmatrix}$

Să se separe vectorii de mai sus în $K = 2$ clase folosind algoritmul *k-means*. În urma clasificării să se reprezinte distinct vectorii din fiecare clasă.

1. Clasificare folosind *k-means*

```
A = [1, 5]'; B = [5, 5]'; C = [1, 1]'; D = [5, 1]';
E = [4, 2]'; F = [8, 2]'; G = [4, -2]'; H = [8, -2]';
X = [A, B, C, D, E, F, G, H]';
K = 2; % K = numar de clase

% clasificare folosind k-means
Ind = kmeans(X, K, 'Replicates',5);
disp(['Vectorii au fost clasificati in 2 clase astfel: '])
disp(['Vectorul A in clasa: ', num2str(Ind(1))])
disp(['Vectorul B in clasa: ', num2str(Ind(2))])
disp(['Vectorul C in clasa: ', num2str(Ind(3))])
disp(['Vectorul D in clasa: ', num2str(Ind(4))])
disp(['Vectorul E in clasa: ', num2str(Ind(5))])
disp(['Vectorul F in clasa: ', num2str(Ind(6))])
disp(['Vectorul G in clasa: ', num2str(Ind(7))])
disp(['Vectorul H in clasa: ', num2str(Ind(8))])
```

```
Vectorii au fost clasificati in 2 clase astfel:
Vectorul A in clasa: 1
Vectorul B in clasa: 1
Vectorul C in clasa: 1
Vectorul D in clasa: 2
Vectorul E in clasa: 1
Vectorul F in clasa: 2
Vectorul G in clasa: 2
Vectorul H in clasa: 2
```

2. Reprezentare grafică a claselor după clustering

```
% reprezentare grafica a punctelor clasificate
figure(1)
hold on
for i = 1:length(Ind)
    if(Ind(i)==1)
        plot(X(i,1), X(i,2), 'ro')
    else
        plot(X(i,1), X(i,2), 'bo')
    end
end
hold off
axis([-2 10 -4 6])
```

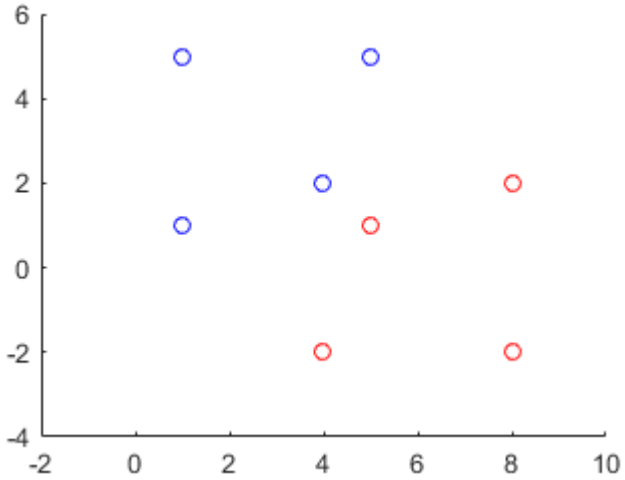


Figura 4.2. Clasificarea punctelor cu k-means clustering

Algoritmul *k-means* a fost repetat de 5 ori și s-a luat cea mai bună împărțire a datelor dintre cele 5 realizate. În urma clasificării, 4 dintre vectori au fost repartizați în clasa *albastră* iar 4 dintre vectori în clasa *roșie*. La o altă rulare, cele două clase se pot schimba între ele: vectorii care acum sunt în clasa *roșie* să ajungă toți în clasa *albastră* și invers; fără alte informații apriori, algoritmul nu are de unde să știe ce etichetă (culoare) să asocieze fiecărei clase.

Centrele claselor în urma etapei de clustering sunt:

```
>> centre
centre =
    6.2500   -0.2500
    2.7500    3.2500
```

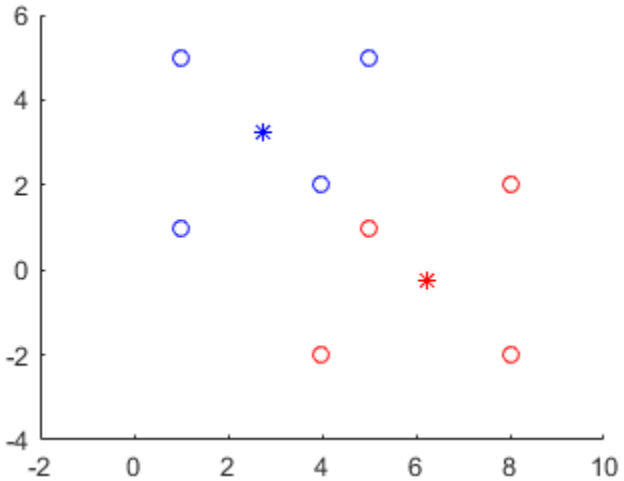


Figura 4.3. Reprezentarea claselor și a centrelor claselor

4.2. Algoritmul *Fuzzy C-means* clustering

4.2.1. Introducere teoretică

În analiza *non-fuzzy* sau *hard clustering*, informaţia este divizată în clustere crisp unde fiecare punct (obiect) aparţine unui cluster. În fuzzy clustering, obiectele pot aparţine mai multor clustere. Asocierea unui obiect cu un cluster se face prin gradele de apartenenţă. Algoritmul *Fuzzy C-means* clustering (FCM), cunoscut şi ca Fuzzy ISODATA, este bazat pe minimizarea funcţiei obiectiv:

$$J(\mathbf{U},\mathbf{V}) = \sum_{i=1}^{N}\sum_{j=1}^{C}(u_{ij})^m \left\|X_i - V_j\right\|^2 \tag{4.2}$$

- $\mathbf{X} = \{X_1, X_2, ..., X_N\}$ este o mulţime de N vectori;
- $\mathbf{V} = \{V_1, V_2, ..., V_c\}$ este setul de vectori reprezentând centrele clusterelor, unde C reprezintă numărul de clustere;
- u_{ij} este gradul de apartenenţă al elementului $\mathbf{X}_i$ la clusterul j ;

  u_{ij} trebuie să satisfacă următoarele condiţii:

$$u_{ij} \in [0,1]\ \forall i = 1,...,N, \forall j = 1,...,C \quad \text{şi} \quad \sum_{j=1}^{C} u_{ij} = 1,\ \forall i = 1,...,N \tag{4.3}$$

- m este gradul de *fuzzy*-ficare; este folosit pentru a controla micile diferenţe dintre gradele de apartenenţă. Valoarea lui m ar trebui să fie cuprinsă în intervalul $m \in [1,\infty]$. Nu există nicio bază teoretică pentru alegerea optimă a valorii lui m, dar de obicei se alege valoarea $m = 2$.

Etapele algoritmului *Fuzzy C-means*

a) Iniţializarea centrelor claselor $\mathbf{V} = \{V_1, V_2, ..., V_C\}$, sau iniţializarea gradelor de apartenenţă u_{ij} cu o valoare arbitrară, care să satisfacă condiţia (4.3), apoi calcularea centrelor claselor;

b) Calcularea gradelor de apartenenţă u_{ij} folosind formula:

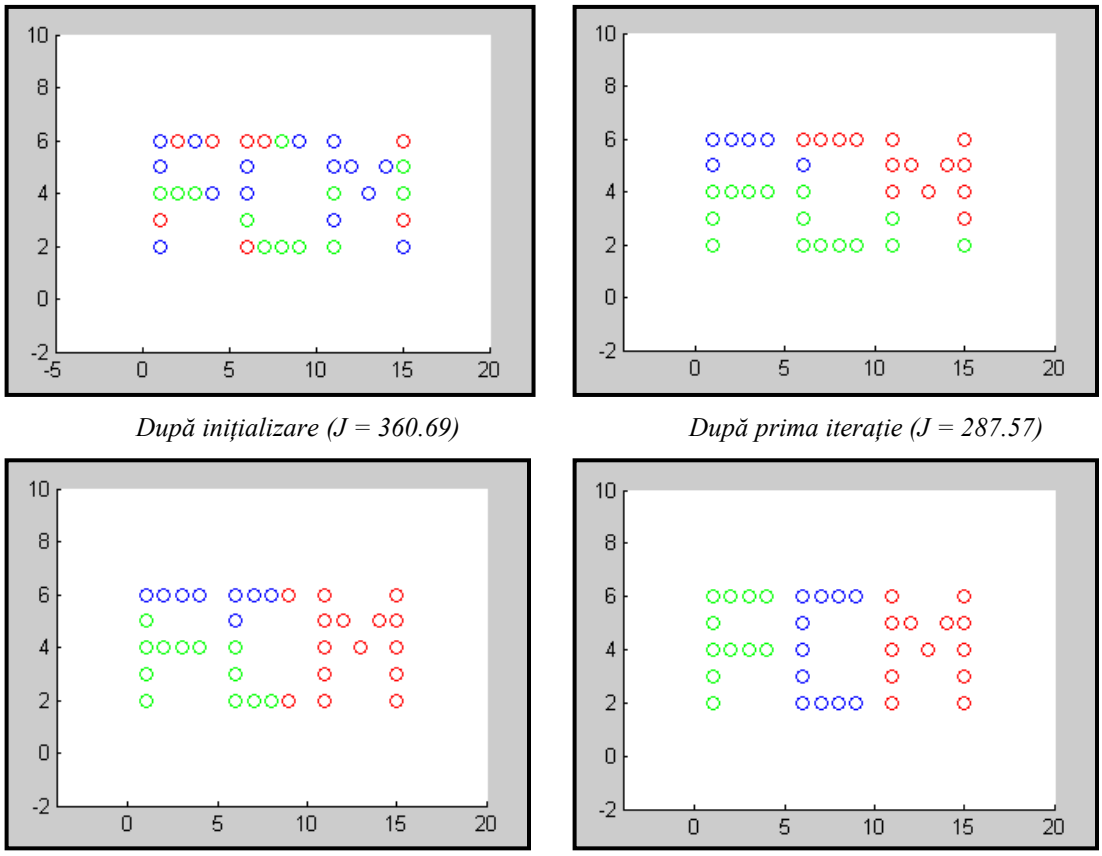
$$u_{ij} = \frac{1}{\sum_{k=1}^{C}\left(\frac{d_{ij}}{d_{ik}}\right)^{\frac{2}{m-1}}}, \quad \text{unde } d_{ij} = \left\|X_i - V_j\right\|, \forall i = 1,...,N, \forall j = 1,...,C; \tag{4.4}$$

c) Calcularea centrelor V_j folosind formul $V_j = \dfrac{\sum_{i=1}^{N}(u_{ij})^m X_i}{\sum_{i=1}^{N}(u_{ij})^m}, \forall j = 1,..,C$ (4.5)

d) Repetarea paşilor *b* şi *c* până când se ajunge la minimul valorii lui J .

Examplu 4.2. Fie cei 35 de vectori 2-dimensionali reprezentați grafic în figura de mai jos. Se dorește împărțirea acestor vectori în 3 clase cât mai bine separate.

În urma rulării algoritmului *fcm clustering* se obține:



După inițializare (J = 360.69) *După prima iterație (J = 287.57)*

După a 5-a iterație (J = 201.25) *După a 10-a iterație (J = 121.15)*

Figure 4.4. Clustering cu FCM

După prima iterație, gradele de apartenență sunt:

U	X_1	X_2	X_3	…	X_{35}
Clasa1	0.3808	0.3789	0.3701	…	0.2974
Clasa2	0.3631	0.3650	0.3660	…	0.3013
Clasa3	0.2561	0.2542	0.2551	…	0.4013

După a 10-a iterație, gradele de apartenență sunt:

U	X_1	X_2	X_3	…	X_{35}
Clasa1	0.8147	0.9008	0.9576	…	0.0375
Clasa2	0.1471	0.0796	0.0341	…	0.0989
Clasa3	0.0382	0.0196	0.0083	…	0.8636

4.2.2. Utilizarea metodei *Fuzzy C-means* clustering în MATLAB

Pentru a realiza clasificarea în MATLAB folosind metoda *fuzzy c-means clustering* se poate folosi funcția `fcm`.

Sintaxă: `[centre,U,obj_fcn] = fcm(X, C)` unde:

- `X` = matricea vectorilor ce se dorește a fi clasificați, așezați pe linii
- `C` = numărul de clase în care se face împărțirea vectorilor
- `centre` = centrele celor 2 clase după terminarea clasificării
- `U` = gradele de apartenență la cele 2 clase după terminarea clasificării
- `obj_fcn` = funcțiile obiectiv după fiecare iterație

Obs: pentru sintaxa completă `>> help fcm`

Aplicația 4.2. Fie următorul set de date bidimensionale:

$$X = \begin{bmatrix} 1 & 5 & 1 & 5 & 4 & 8 & 4 & 8 & 7 & 11 & 7 & 11 \\ 5 & 5 & 1 & 1 & 2 & 2 & -2 & -1 & 5 & 5 & 1 & 1 \end{bmatrix}$$

Se dorește împărțirea automata în 3 clase a setului de date *X*.

Reprezentarea în plan a punctelor este următoarea:

```
X = [1,5,1,5,4,8,4,8,7,11,7,11;
     5,5,1,1,2,2,-2,-2,5,5,1,1]';
C = 3; % C = numar de clase
figure(1)
plot(X(:,1), X(:,2),'ok')
axis([-1 13 -4 7])
```

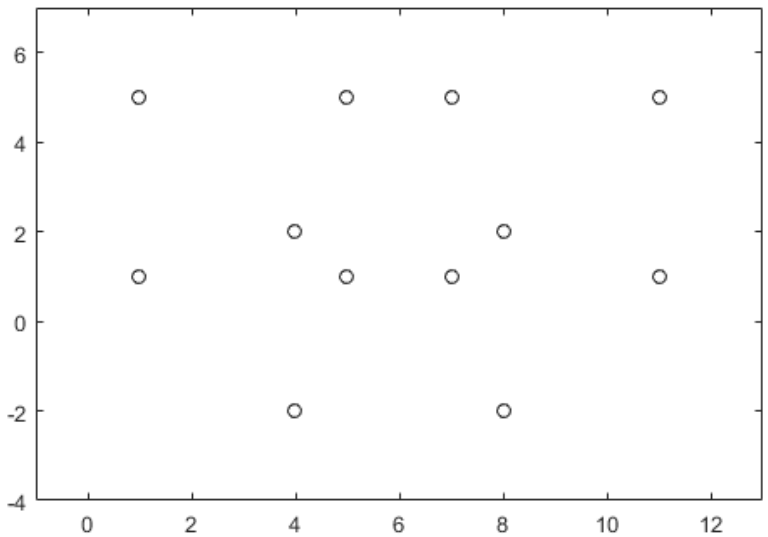


Figura 4.5. Reprezentarea punctelor în plan

```
[centre,U,obj_fcn] = fcm(X,C);
[val, indici_clase] = max(U,[],1);

figure(2)
hold on
for i = 1:length(indici_clase)
    if(indici_clase(i)==1)
        plot(X(i,1), X(i,2), 'ro')
        plot(centre(1,1),centre(1,2),'r*')
    elseif(indici_clase(i)==2)
        plot(X(i,1), X(i,2), 'bo')
        plot(centre(2,1),centre(2,2),'b*')
    else
        plot(X(i,1), X(i,2), 'go')
        plot(centre(3,1),centre(3,2),'g*')
    end
end
hold off
axis([-1 13 -4 7])
```

În urma rulării algoritmului *fuzzy c-means* de mai sus s-au obținut:

- mediile celor 3 clase:
```
centre =
    2.4788    3.1588
    6.0086    0.2030
    9.5215    3.1737
```
- variabila `U` având 3 linii și 12 coloane; pe colana `i`, linia 1, este gradul de apartenență al vectorului de pe linia `i` din `X` la clasa 1; pe colana `i`, linia 2, este gradul de apartenență al vectorului de pe linia `i` din `X` la clasa 2 etc; maximul de pe coloana `i` va fi salvat în variabila `val(i)`, iar în `indici_clase(i)` se va salva linia din coloana `U(i)` pe care s-a găsit `val(i)`, adică clasa în care va fi clasificat vectorul de pe linia `i`.

Matricea `U` arată astfel:

0.840	0.550	0.738	0.123	0.617	0.071	0.210	0.105	0.224	0.061	0.053	0.065
0.097	0.223	0.196	0.822	0.311	0.315	0.683	0.686	0.223	0.096	0.826	0.198
0.061	0.225	0.0653	0.0540	0.070	0.613	0.106	0.208	0.552	0.841	0.120	0.735

Vectorul `val` este:

0.840	0.550	0.738	0.822	0.617	0.613	0.683	0.686	0.552	0.841	0.826	0.735

Vectorul `indici_clase` este:

1	1	1	2	1	3	2	2	3	3	2	3

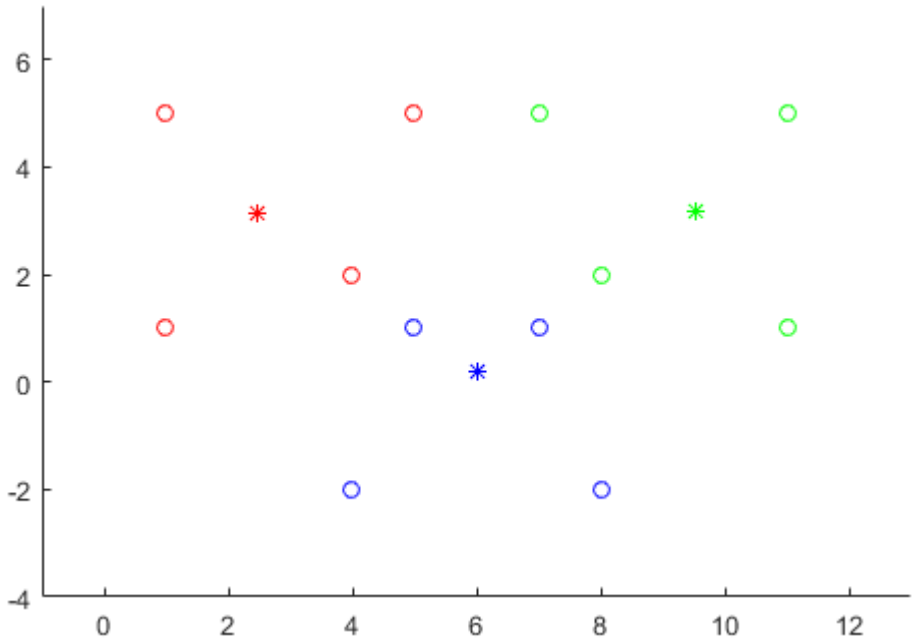


Figura 4.6. Clasificarea punctelor cu *fuzzy c-means* clustering

Algoritmul a rulat 75 de iterații, valoarea finală a funcției obiectiv fiind 49.54.

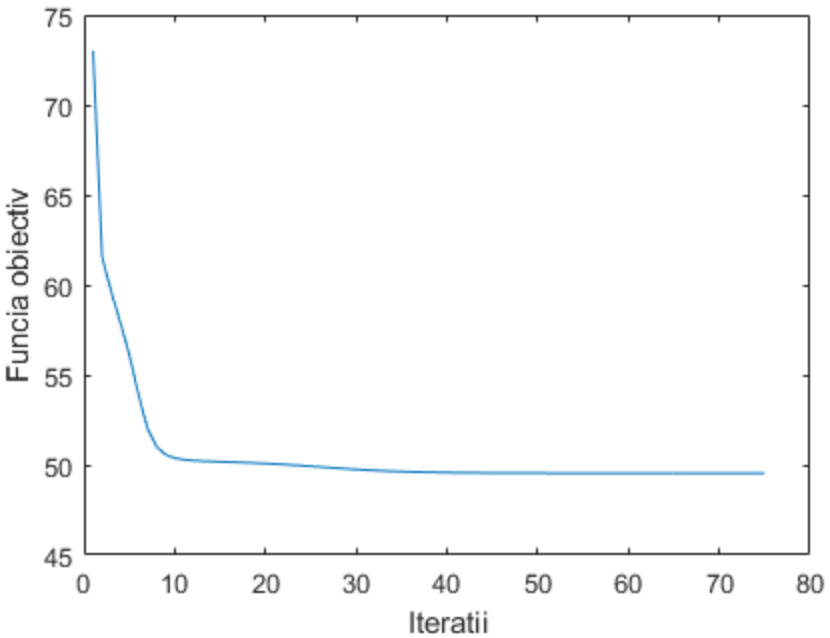


Figura 4.7. Evoluția funcției obiectiv în timpul algoritmului de clustering

4.3. Rețele cu auto-organizare (en. *Self-Organizing Maps* - SOM)

Rețelele cu auto-organizare (en. *Self-Organizing Maps*) au fost dezvoltate de Teuvo Kohonen [Kohonen1982]. Învățarea cu aceste rețele constă în modificarea adaptivă a ponderilor unei rețele de neuroni care interacționează local, ca răspuns la excitațiile de intrare, conform unei anumite reguli de învățare, până când rețeaua converge spre o anumită configurație.

Ideea de bază a învățării prin auto-organizare constă în introducerea interacțiilor locale între neuronii rețelei, în sensul că modificarea comportamentului unuia dintre neuroni afectează direct comportamentul neuronilor din imediata vecinătate. Această interacțiune locală conduce în timpul procesului de învățare la o ordonare globală a rețelei, având ca efect un comportament coerent [Neagoe1998].

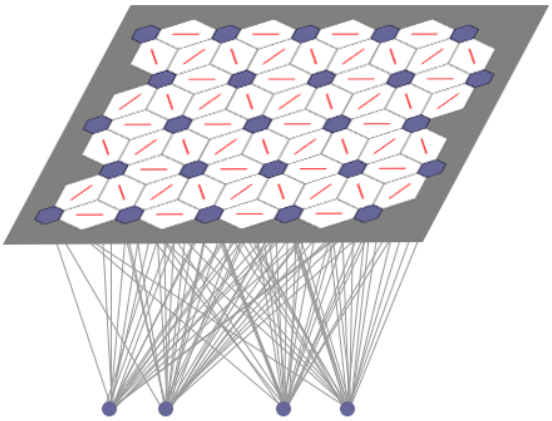


Figura 4.8. Arhitectura rețelei SOM bidimensionale

4.3.1. Structură rețea SOM

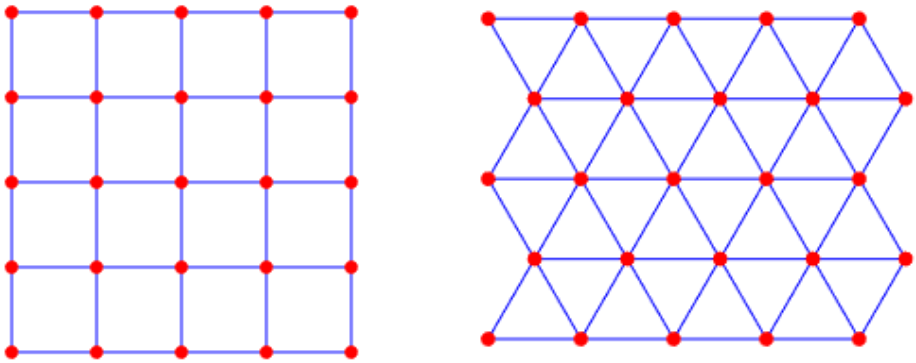
Rețeaua SOM este formată din două straturi:

- **un strat de noduri de intrare** având un număr de noduri egal cu dimensiunea vectorilor aplicați la intrarea rețelei. Pentru prelucrări de imagini, de cele mai multe ori vectorul de intrare este compus din pixelii imaginii. De exemplu, dacă se antrenează rețeaua cu imagini grayscale având dimensiunea de 20 x 30 pixeli, atunci vor fi 600 de noduri în stratul de intrare, fiecare nod reprezentând câte un pixel din imagine.
- **un strat de neuroni de ieșire** ce conține un număr de neuroni mai mare sau egal cu numărul de clase în care se dorește să se facă clasificarea vectorilor de intrare. De exemplu, dacă se dorește să se antreneze rețeaua SOM pentru recunoașterea cifrelor, atunci în stratul de ieșire trebuie să fie cel puțin 10 neuroni.

- există **conexiuni** de la fiecare nod de intrare la fiecare neuron de ieșire, și fiecare conexiune este determinată de o pondere care se rafinează în timpul instruirii.

Convenție de notare a ponderilor
$w_{j,i}$ = reprezintă ponderea conexiunii neuronului j din stratul de ieșire cu nodul i din stratul de intrare.

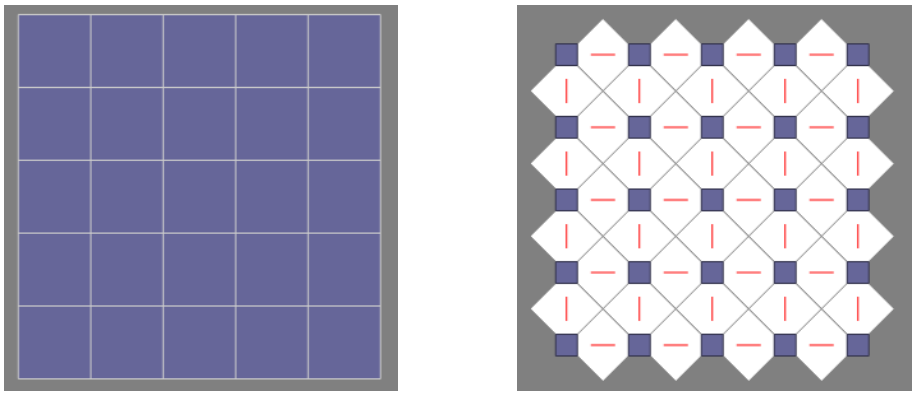
Neuronii din stratul de ieșire sunt interconectați printr-o relație de vecinătate. Această relație dictează topologia hărții. În mod uzual se folosesc două tipuri de topologii: **rectangulară** și **hexagonală**.



Topologie rectangulară de 5 x 5 Topologie hexagonală de 5 x 5

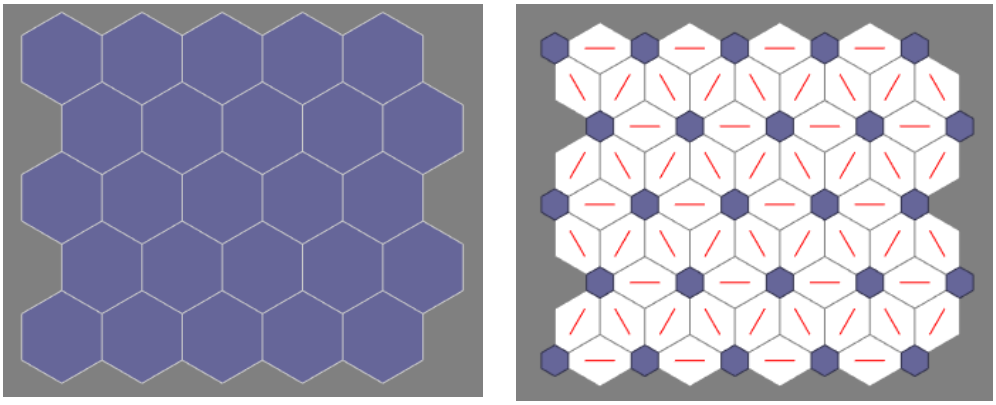
Figura 4.9. Diverse moduri în care sunt interconectați neuronii din stratul de ieșire al rețelei SOM

O altă modalitate de reprezentare a topologiilor mai sus amintite este următoarea:



Stratul neuronilor de ieșire Stratul neuronilor de ieșire cu afișarea legăturilor dintre neuroni

Figura 4.10. Topologie rectangulară 5 x 5



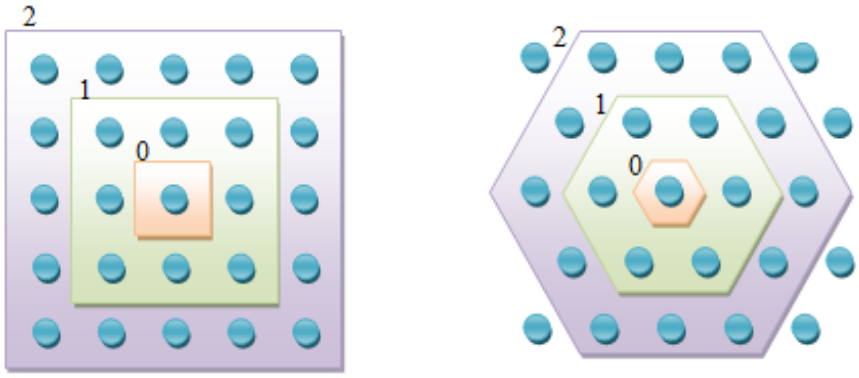
Stratul neuronilor de ieşire

Stratul neuronilor de ieşire cu afişarea legăturilor dintre neuroni

Figura 4.11. Topologie hexagonală de 5 x 5

Principiul vecinătăţii

Între neuronii din stratul de ieşire există o relaţie de distanţă; principiul vecinătăţii foloseşte aceste distanţe pentru a determina vecinătăţile pentru care se aplică rafinarea ponderilor. În *Figura 4.12* sunt 2 exemple de vecinătăţi, pentru topologia rectangulară, respectiv pentru topologia hexagonală.

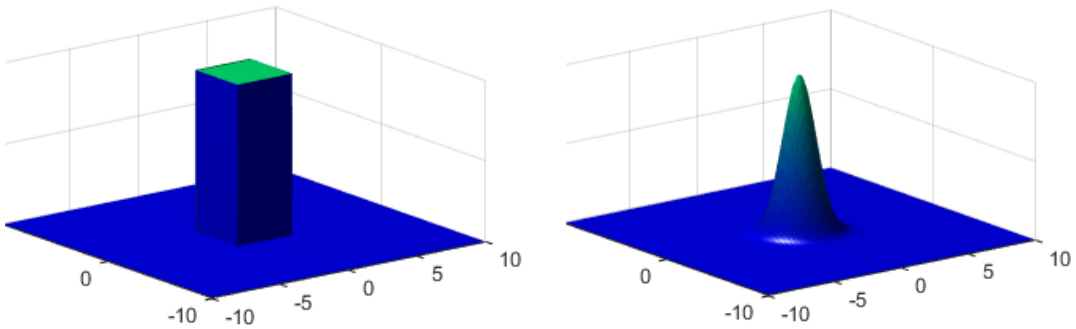


(a) topologie rectangulară plană (b) topologie hexagonală plană

Figura 4.12. Posibile tipuri de vecinătăţi pentru reţelele Kohonen
zona 0 – neuron câştigător, **zona 1** – vecinătate de ordin 1, **zona 2** – vecinătate de ordin 2

Funcţia de vecinătate determină cât de puternic sunt conectaţi neuronii între ei. Cele mai uzuale funcţii de vecinătate sunt:

- funcţia *bubble*, pentru care neuronii din întrega vecinătate a neuronului câştigător sunt la fel de puternic conectaţi şi neuronii din afara vecinătăţii nu sunt deloc conectaţi;
- funcţia *gaussiană*, pentru care neuronii din imediata apropiere a neuronului câştigător sunt mai puternic conectaţi decât cei mai depărtaţi.



Funcția de vecinătate *bubble* Funcția de vecinătate *gaussiană*

Figura 4.13. Exemple de funcții de vecinătate

4.3.2. Algoritmul clasic de instruire al rețelei SOM

Pas 1. Inițializare:

- se inițializează ponderile celor *M* neuroni de ieșire cu valori aleatoare mici;
- se aleg parametrii de control și condiția de oprire;
- se alege tipul de distanță folosită; de obicei se folosește *distanța Euclidiană*, dar se pot folosi si alte metrici, depinzând de particularitățile algoritmului și ale rețelei;
- se inițializează contoarele;

Pas 2. Se prezintă rețelei un nou vector de instruire.

Pas 3. Se calculează distanța dintre vectorul de intrare și fiecare nod de ieșire al rețelei conform relației:

$$D_j = \sum_{i=0}^{N-1} \left[x_i(t) - w_{ji}(t)\right]^2, \text{unde} \qquad (4.6)$$

- $x_i(t)$ este componenta *i* a vectorului de intrare, la momentul de timp *t*;
- $w_{ji}(t)$ este ponderea conexiunii dintre neuronul *j* din stratul de ieșire și nodul *i* din stratul de intrare la momentul de timp *t*;
- *N* este dimensiunea vectorului de intrare; *i = 0...N-1*;
- D_j este distanța dintre vectorul de intrare și setul de ponderi asociat neuronului *j* din stratul de ieșire; *j = 0 ... M-1;*

Pas 4. Se selectează neuronul câștigător *j**, aflat la distanța minimă față de vectorul de intrare.

Pas 5. Se actualizează ponderile neuronului *j** și ale tuturor neuronilor din vecinătatea neuronului *j** la momentul respectiv. Regula de ajustare este dependentă de distanța folosită, de tipul de vecinătate adoptat și de alți parametri de control cum ar fi *rata de învățare* $\eta(t) \in [0..1]$, care controlează cât de mult sunt afectați neuronii din vecinătate. Rata de învățare descrește în timp.

Un exemplu de actualizare a ponderilor pentru vecinătăți de tip *bubble* este:

- $w_{ji}(t+1) = w_{ji}(t) + \eta(t)[x_i(t) - w_{ji}(t)]$, pentru neuroni din interiorul vecinătății;
- $w_{ji}(t+1) = w_{ji}(t)$, pentru neuroni din afara vecinătății.

Pas 6. Se verifică condițiile de stop. Dacă acestea se îndeplinesc, procesul de învățare se termină, altfel se merge la *Pas 2*.
Condiții de stop. Cele mai folosite condiții de stop pentru rafinarea ponderilor sunt:

- algoritmul a rulat un număr prestabilit de epoci;
- gradientul de modificare a ponderilor rețelei devine nesemnficativ;
  $|w_{ji}(t+1) - w_{ji}(t)| < \varepsilon$,unde ε reprezintă pragul de oprire a rafinării.

După ce învățarea rețelei neurale a luat sfârșit se trece la etapa de **calibrare a hărții** în scopul localizării diferitelor date de intrare. **Această etapă este supervizată.**

Calibrarea hărții se realizează astfel:

- se aduc la intrarea rețelei toți vectorii din lotul de antrenare.
- se calculează apoi distanța euclidiană dintre vectorii de antrenare și setul de ponderi asociat fiecărui neuron din stratul de ieșire.
- pentru fiecare vector de antrenare se declară câștigător neuronul pentru care se obține distanța euclidiană minimă.
- după ce se trec prin rețea toți vectorii de antrenare, în final va rezulta o hartă unde pentru fiecare neuron se specifică câți vectori din fiecare clasă îl au câștigător.
- fiecărui neuron i se va asocia apoi clasa din care fac parte cei mai mulți vectori.

Atenție! Varianta de SOM utilizată este nesupervizată atunci când se antrenează rețeaua dar este supervizată după procesul de antrenare când trebuie ca fiecărui neuron din stratul de ieșire să i se atribuie o clasă.

Exemplu 4.3. Antrenarea unei rețele SOM pentru clustering în 2 clase
Fie setul de vectori:

$$X = [X_1, \quad X_2, \quad X_3, \quad X_4, \quad X_5, \quad X_6] = \begin{bmatrix} 1 & 1 & 2 & 5 & 5 & 6 \\ 1 & 3 & 2 & 1 & 3 & 2 \end{bmatrix}$$

Se dorește să se afle din ce clasă face parte vectorul $X_{test} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$.

Se cunoaște că vectorii $X_1,\ X_2,\ X_3,\ X_4,\ X_5,\ X_6$ sunt împărțiți în două clase C_1 și C_2 astfel: clasa $C_1 = [X_1,\ X_2,\ X_3]$ și clasa $C_2 = [X_4,\ X_5,\ X_6]$.

Soluție

Se va folosi o rețea SOM cu următorii parametri:

- 2 x 2 neuroni în stratul de ieșire
- 2 noduri în stratul de intrare
- vecinătate: *bubble* de dimensiune 1
- rata de învățare $\eta = 0.1$.

După fiecare epocă η se înjumătățește

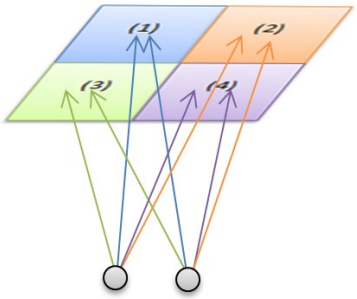


Figura 4.14. Arhitectură rețea SOM folosită pentru rezolvarea problemei

Rezolvarea problemei presupune:

Inițializare. Se inițializează aleator ponderile rețelei:

$$W^{(1)} = \begin{bmatrix} 0.1 & 0.2 \\ 0.2 & -0.1 \\ 0.1 & 0.1 \\ -0.3 & 0.1 \end{bmatrix} \begin{matrix} \Longrightarrow & W_1^T \\ \Longrightarrow & W_2^T \\ \Longrightarrow & W_3^T \\ \Longrightarrow & W_4^T \end{matrix}$$

Distanța folosită va fi distanța Euclidiană.

Pas 1. Se aduce la intrarea rețelei primul vector din lotul de antrenare $X_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

Se calculează distanțele dintre X_1 și cei 4 neuroni din stratul de ieșire.

$$d(X_1, W_1) = \sqrt{(1-0.1)^2 + (1-0.2)^2} = 1.20$$
$$d(X_1, W_2) = \sqrt{(1-0.2)^2 + (1+0.1)^2} = 1.36$$
$$d(X_1, W_3) = \sqrt{(1-0.1)^2 + (1-0.1)^2} = 1.27$$
$$d(X_1, W_4) = \sqrt{(1+0.3)^2 + (1-0.1)^2} = 1.58$$

- Se constată că distanța minimă este față de neuronul numărul 1.
- Neuronul cu numărul 1 este declarat câștigător și doar lui i se va actualiza setul de ponderi.

$$W_1^{(2)} = W_1^{(1)} + \eta \cdot [X_1 - W_1^{(1)}]$$

$$W_1^{(2)} = \begin{bmatrix}0.1\\0.2\end{bmatrix} + 0.1 \cdot \begin{bmatrix}1-0.1\\1-0.2\end{bmatrix} = \begin{bmatrix}0.1\\0.2\end{bmatrix} + 0.1 \cdot \begin{bmatrix}0.9\\0.8\end{bmatrix} = \begin{bmatrix}0.1\\0.2\end{bmatrix} + \begin{bmatrix}0.09\\0.08\end{bmatrix} = \begin{bmatrix}0.19\\0.28\end{bmatrix}$$

- Setul de ponderi devine:

$$W^{(2)} = \begin{bmatrix}0.19 & 0.28\\0.2 & -0.1\\0.1 & 0.1\\-0.3 & 0.1\end{bmatrix} \begin{matrix}\Longrightarrow & W_1^T\\\Longrightarrow & W_2^T\\\Longrightarrow & W_3^T\\\Longrightarrow & W_4^T\end{matrix}$$

Pas 2. Se aduce la intrarea rețelei al 2-lea vector din lotul de antrenare $X_2 = \begin{bmatrix}1\\3\end{bmatrix}$

$$d(X_2, W_1) = 2.84$$
$$d(X_2, W_2) = 3.2$$
$$d(X_2, W_3) = 3.03$$
$$d(X_2, W_4) = 3.17$$

- Se constată că distanța minimă este față de neuronul numărul 1.
- Neuronul cu numărul 1 este declarat câștigător și doar lui i se va actualiza setul de ponderi.

$$W_1^{(3)} = W_1^{(2)} + \eta \cdot [X_2 - W_1^{(2)}]$$

$$W_1^{(3)} = \begin{bmatrix}0.19\\0.28\end{bmatrix} + 0.1 \cdot \begin{bmatrix}1-0.19\\3-0.28\end{bmatrix} = \begin{bmatrix}0.27\\0.55\end{bmatrix}, \quad W^{(3)} = \begin{bmatrix}0.27 & 0.55\\0.2 & -0.1\\0.1 & 0.1\\-0.3 & 0.1\end{bmatrix} \begin{matrix}\Longrightarrow & W_1^T\\\Longrightarrow & W_2^T\\\Longrightarrow & W_3^T\\\Longrightarrow & W_4^T\end{matrix}$$

Pas 3. Se aduce la intrarea rețelei al 3-lea vector din lotul de antrenare etc.

După ce se aduc la intrarea rețelei toți vectorii din lotul de antrenare (la finalul primei epoci), rata de învățare se modifică și devine:

$$\eta = \frac{\eta}{2} = 0.05$$

După ce se ajunge la condiția de stop, setul de ponderi devine:

$$W = \begin{bmatrix}5.5 & 1.5\\1.5 & 1.5\\3 & 3\\1 & 3\end{bmatrix} \begin{matrix}\Longrightarrow & W_1^T\\\Longrightarrow & W_2^T\\\Longrightarrow & W_3^T\\\Longrightarrow & W_4^T\end{matrix}$$

Revenim acum la întrebarea inițială: din ce clasă face parte vectorul $X_{test} = \begin{bmatrix}2\\1\end{bmatrix}$?

Pentru a putea spune din ce clasă face parte vectorul X_{test}, este nevoie de **calibrarea hărții** în scopul localizării diferitelor date de intrare.

Calibrarea rețelei SOM pentru problema dată

Etapa 1. Se aduc pe rând la intrarea rețelei toți vectorii din clasa C_1.

- Se aduce la intrarea rețelei **vectorul $\boldsymbol{X_1}$**.

Atenție! Ponderile rețelei sunt cele calculate în urma antrenării.

$$d(X_1, W_1) = \sqrt{(1-5.5)^2 + (1-1.5)^2} = 4.52$$
$$d(X_1, W_2) = \sqrt{(1-1.5)^2 + (1-1.5)^2} = 0.7$$
$$d(X_1, W_3) = \sqrt{(1-3)^2 + (1-3)^2} = 2.82$$
$$d(X_1, W_4) = \sqrt{(1-1)^2 + (1-3)^2} = 2$$

Se constată că distanța minimă este față de **neuronul numărul 2**.

- Se aduce la intrarea rețelei **vectorul $\boldsymbol{X_2}$**.

$$d(X_2, W_1) = \sqrt{(1-5.5)^2 + (3-1.5)^2} = 4.74$$
$$d(X_2, W_2) = \sqrt{(1-1.5)^2 + (3-1.5)^2} = 1.58$$
$$d(X_2, W_3) = \sqrt{(1-3)^2 + (3-3)^2} = 2$$
$$d(X_2, W_4) = \sqrt{(1-1)^2 + (3-3)^2} = 0$$

Se constată că distanța minimă este față de **neuronul numărul 4.**

- Se aduce la intrarea rețelei **vectorul $\boldsymbol{X_3}$**.

$$d(X_3, W_1) = \sqrt{(2-5.5)^2 + (2-1.5)^2} = 3.53$$
$$d(X_3, W_2) = \sqrt{(2-1.5)^2 + (2-1.5)^2} = 0.70$$
$$d(X_3, W_3) = \sqrt{(2-3)^2 + (2-3)^2} = 1.41$$
$$d(X_3, W_4) = \sqrt{(2-1)^2 + (2-3)^2} = 1.41$$

Se constată că distanța minimă este față de **neuronul numărul 2.**

Etapa 2. Se aduc pe rând la intrarea rețelei toți vectorii din clasa C_2.

- Se aduce la intrarea rețelei **vectorul $\boldsymbol{X_4}$**.

$$d(X_4, W_1) = \sqrt{(5-5.5)^2 + (1-1.5)^2} = 5.65$$
$$d(X_4, W_2) = \sqrt{(5-1.5)^2 + (1-1.5)^2} = 3.53$$
$$d(X_4, W_3) = \sqrt{(5-3)^2 + (1-3)^2} = 2.82$$
$$d(X_4, W_4) = \sqrt{(5-1)^2 + (1-3)^2} = 4.47$$

Se constată că distanța minimă este față de **neuronul numărul 3**.

- Se aduce la intrarea rețelei **vectorul $\boldsymbol{X_5}$**.

$$d(X_5, W_1) = \sqrt{(5-5.5)^2 + (3-1.5)^2} = 0.7$$
$$d(X_5, W_2) = \sqrt{(5-1.5)^2 + (3-1.5)^2} = 3.53$$
$$d(X_5, W_3) = \sqrt{(5-3)^2 + (3-3)^2} = 2.82$$
$$d(X_5, W_4) = \sqrt{(5-1)^2 + (3-3)^2} = 4.47$$

Se constată că distanța minimă este față de **neuronul numărul 1**.

- Se aduce la intrarea rețelei **vectorul X_6**.

$$d(X_6, W_1) = \sqrt{(6-5.5)^2 + (2-1.5)^2} = 0.70$$
$$d(X_6, W_2) = \sqrt{(6-1.5)^2 + (2-1.5)^2} = 4.52$$
$$d(X_6, W_3) = \sqrt{(6-3)^2 + (2-3)^2} = 3.16$$
$$d(X_6, W_4) = \sqrt{(6-1)^2 + (2-3)^2} = 5.09$$

Se constată că distanța minimă este față de **neuronul numărul 1**.

În urma etapei de calibrare s-a obținut:

0	**2**
0	**1**

Harta neuronilor câștigători pentru clasa C_1

2	**0**
1	**0**

Harta neuronilor câștigători pentru clasa C_2

Fiecărui neuron i se va asocia apoi clasa din care fac parte cei mai mulți vectori. De exemplu, în cazul de mai sus:

- neuronului 1 i se va asocia clasa C_2,
- neuronului 2 i se va asocia clasa C_1,
- neuronului 3 i se va asocia clasa C_2,
- neuronului 4 i se va asocia clasa C_1.

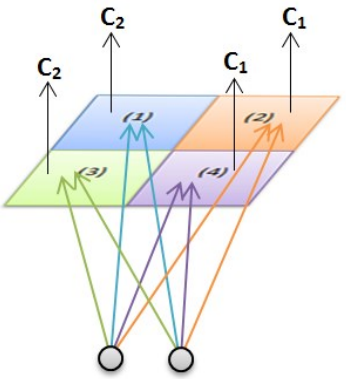


Figura 4.15. Rețeaua SOM în urma calibrării

Revenind acum la vectorul de test $X_{test} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$, se vor calcula distanțele dintre X_{test} și toți neuronii din stratul de ieșire al rețelei SOM. Clasa de apartenență a neuronului pentru care se va obține distanța minimă va fi clasa de apartenență a vectorului X_{test}.

$$(X_{test}, W_1) = \sqrt{(2-5.5)^2 + (1-1.5)^2} = 3.53$$
$$d(X_{test}, W_2) = \sqrt{(2-1.5)^2 + (1-1.5)^2} = 0.70$$
$$d(X_{test}, W_3) = \sqrt{(2-3)^2 + (1-3)^2} = 2.23$$
$$d(X_{test}, W_4) = \sqrt{(2-1)^2 + (1-3)^2} = 2.23$$

Se constată că distanța minimă este față de neuronul numărul 2 care apaține clasei C_1, deci vectorul X_{test} aparține clasei C_1.

Exemplu 4.4. Antrenarea și calibrarea unei rețele SOM pentru învățarea cifrelor 1 și 2 folosind baza de date MNIST

Acesta se dorește a fi un exemplu didactic, de aceea rețeaua SOM a fost antrenată numai cu imagini din două clase, astfel:

- 563 de imagini conținând *cifra 1*
- 488 de imagini conținând *cifra 2*
- dimensiunea unei imagini este de 28 x 28 pixeli

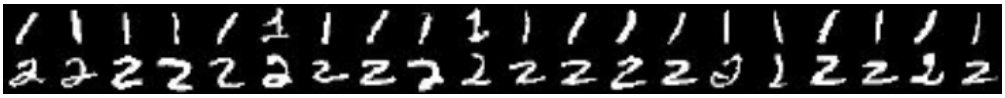


Figura 4.16. Exemple de imagini din baza de date MNIST utilizate la antrenarea rețelei SOM

Arhitectură rețea

- Se antrenează rețeaua cu imagini de n x n pixeli. Imaginea este transformată într-un vector linie cu n^2 elemente (în cazul în care era imagine color RGB erau $3 \cdot n^2$ elemente). Deci pentru exemplul nostru sunt 784 neuroni în stratul de intrare (28 x 28 = 784).
- Rețeaua se auto-organizează (nesupervizat) în funcție de vectorii din lotul de antrenare. În urma antrenării, fiecare neuron din stratul de ieşire are asociat un vector de ponderi de dimensiune n^2.
- Se folosește o rețea rectangulară de dimensiune 5 x 5, deci vor fi 25 de neuroni în stratul de ieşire.

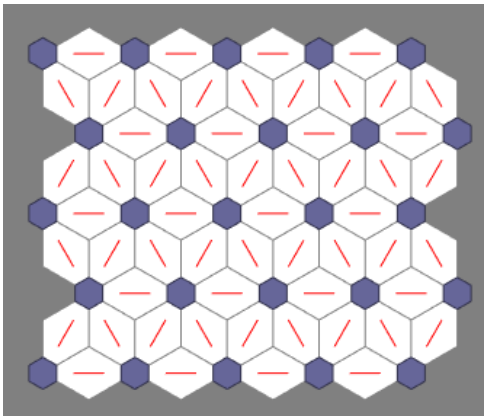


Figura 4.17. Conexiunile neuronilor din stratul de ieşire

În urma antrenării se poate determina corespondența dintre fiecare vector din lotul de antrenare și neuronii din stratul de ieşire (pe baza distanței minime euclidiene).

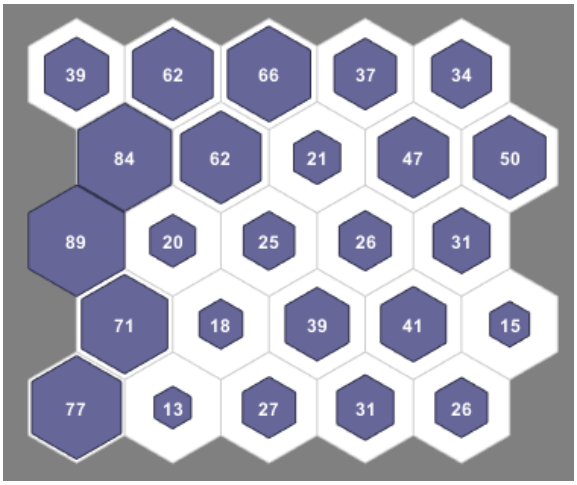


Figura 4.18. Neuronii câștigători

După etapa nesupervizată de antrenare, urmează apoi etapa supervizată de atribuire a unei clase fiecărui neuron din stratul de ieșire astfel:

- pentru fiecare clasă în parte, se aduc la intrarea rețelei vectorii din lotul de antrenare.
- se calculează apoi distanța euclidiană dintre vectorii de antrenare și setul de ponderi asociat fiecărui neuron din stratul de ieșire.
- se declară câștigător neuronul pentru care se obține distanța euclidiană minimă.
- după ce se trec prin rețea toți vectorii din lotul de antrenare dintr-o clasă, în final va rezulta o hartă unde în fiecare celulă se specifică câți vectori din lotul de antrenare au un anumit neuron câștigător.
- fiecărui neuron i se va asocia apoi clasa din care fac parte cei mai mulți vectori.

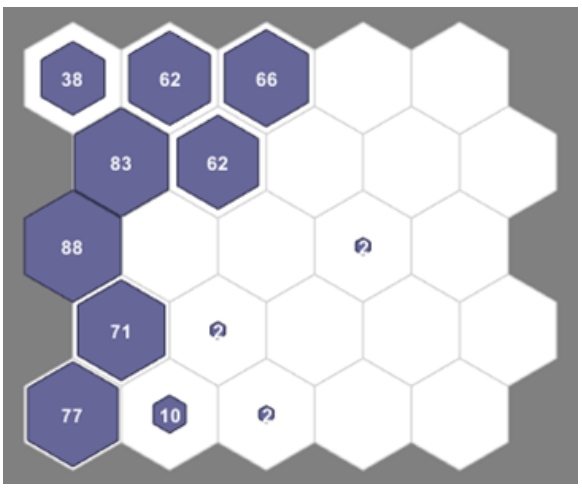


Figura 4.19. Mapare rețea pentru *Cifra 1*

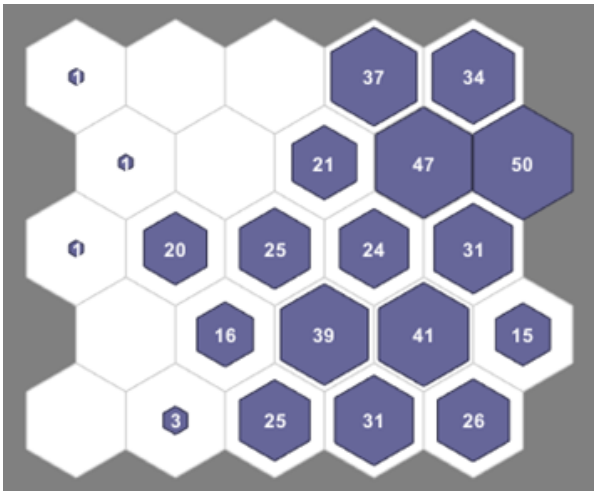


Figura 4.20. Mapare rețea pentru *Cifra 2*

Pentru exemplul de mai sus, unde am considerat 2 clase (*Cifra1* și *Cifra2*):

- neuronul de coordonate (0,0) este câștigător pentru 77 vectori din clasa *Cifra*1 și pentru 0 vectori din clasa *Cifra2*;
- neuronul de coordonate (2,0) este câștigător pentru 2 vectori din clasa *Cifra*1 și pentru 25 vectori din clasa *Cifra2*;
- neuronul de coordonate (4,0) este câștigător pentru 26 vectori din clasa *Cifra*2 și pentru 0 vectori din clasa *Cifra1*;

Fiecărui neuron i se va asocia apoi clasa din care fac parte cei mai mulți vectori. De exemplu, în cazul de mai sus:

- neuronului de coordonate (0,0) i se va asocia clasa *Cifra*1,
- neuronului de coordonate (2,0) i se va asocia clasa *Cifra2,*
- neuronului de coordonate (5,0) i se va asocia clasa *Cifra2 etc*

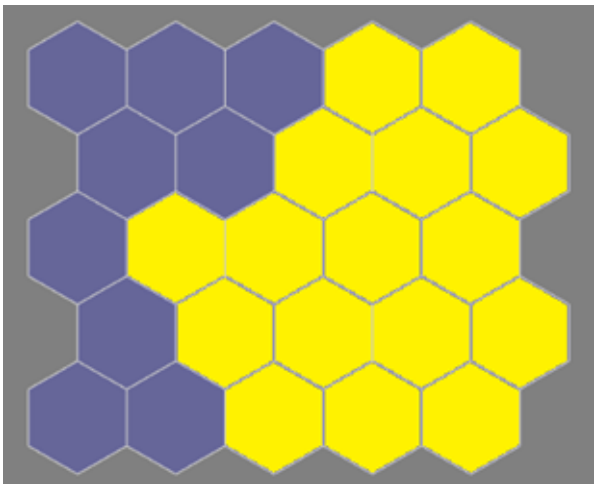


Figura 4.21. Calibrarea rețelei SOM în 2 clase (mov: cifra 1, galben: cifra 2)

Așa cum am menționat anterior, pentru a putea antrena rețeaua SOM cu imagini (reprezentate ca matrice bidimensionale cu M linii și N coloane pentru imagini grayscale), acestea sunt transformate în vectori (linie sau coloană) având $M \cdot N$ elemente, prin urmare și setul de ponderi asociat unui neuron din stratul de ieșire este tot un vector cu $M \cdot N$ elemente. În urma antrenării rețelei, ponderile au fost rafinate astfel încât să se asemene cât mai bine cu imaginile din lotul de antrnare.

Pentru a vizualiza setul de ponderi asociate unui neuron din stratul de ieșire, nu trebuie decât să transformăm vectorul de ponderi de dimensiune $M \cdot N$ într-o matrice bidimensională cu M linii și N coloane, care poate fi afișată ca imagine.
De exemplu, setul de ponderi asociat neuronului de pe lina 5, coloana 3, din stratul de ieșire arată astfel:

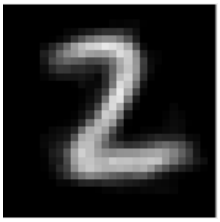


Figura 4.22. Setul de ponderi asociat neuronului de pe lina 5, coloana 3

Ponderile asociate tuturor neuronilor din stratul de ieșire arată astfel:

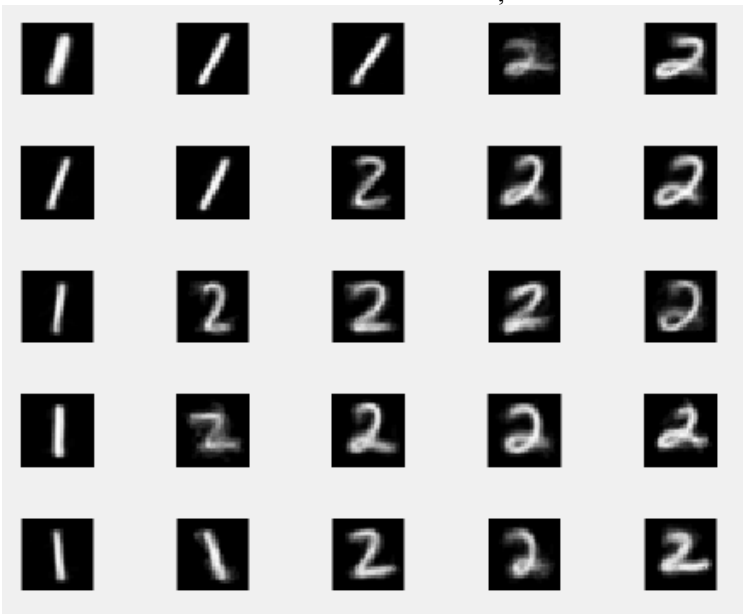


Figura 4.23. Reprezentarea grafică a ponderilor asociate neuronilor din stratul de ieșire

4.3.3. Implementarea rețelei SOM în MATLAB

1. Pentru definirea rețelei SOM se poate folosi funcția `selforgmap`:

Sintaxă: `SOMnet = selforgmap(M,N)` unde:

- `M, N` reprezintă dimensiunile rețelei SOM

Dacă nu se specifică, implicit rețeaua este de 8 x 8, cu topologie hexagonală, vecinătate inițială de 3 neuroni și rulează 200 de epoci.

2. Antrenarea rețelei se face cu funcția `train`.

Sintaxă: `SOMnet = train(SOMnet,X)` unde:

- `SOMnet =` rețeaua SOM creată anterior cu funcția `selforgmap`
- `X =` matricea vectorilor de antrenare, așezați pe coloane

3. Pentru a vizualiza rețeaua se poate folosi funcția `view`.

Sintaxă: `view(SOMnet)`

4. Pentru a simula/testa rețeaua se poate folosi funcția `sim`.

Sintaxă: `O = sim(SOMnet,Y)` unde:

- `SOMnet =` rețeaua SOM creată anterior cu funcția `selforgmap`
- `Y` = lotul de test
- `O` = ieșirile obținute în urma clasificării

Același lucru se obține și folosind sintaxa `O = SOMnet(Y)`

5. Pentru a vedea câți vectori sunt alocați fiecărui neuron din stratul de ieșire se poate folosi funcția `plotsomhits.`

Sintaxă: `plotsomhits(SOMnet, Y)` unde:

- `SOMnet =` rețeaua SOM creată anterior cu funcția `selforgmap`
- `Y =` matricea vectorilor pentru care se dorește să se vadă câți vectori sunt alocați fiecărui neuron din stratul de ieșire

6. Pentru a vizualiza ponderile rețelei poate fi folosită comanda:
   `ponderi = SOMnet.IW{1,1}` unde:

- ponderi = ponderile rețelei
- `SOMnet =` rețeaua SOM creată anterior

Obs: pentru sintaxa completă a funcțiilor de mai sus consultați *help*-ul din MATLAB

Aplicația 4.3. Exemplu de antrenare a unei rețele SOM
Fie vectorii:

$A = \begin{bmatrix}1\\5\end{bmatrix}$, $B = \begin{bmatrix}5\\5\end{bmatrix}$, $C = \begin{bmatrix}1\\1\end{bmatrix}$, $D = \begin{bmatrix}5\\1\end{bmatrix}$, $E = \begin{bmatrix}4\\2\end{bmatrix}$, $F = \begin{bmatrix}8\\2\end{bmatrix}$, $G = \begin{bmatrix}4\\-2\end{bmatrix}$, $H = \begin{bmatrix}8\\-2\end{bmatrix}$

Să se antreneze o rețea SOM cu vectorii de mai sus. Rețeaua are 2 x 2 neuroni pe stratul de ieșire.

1. Configurare, antrenare și afișare rețea SOM

```
A = [1, 5]'; B = [5, 5]'; C = [1, 1]'; D = [5, 1]';
E = [4, 2]'; F = [8, 2]'; G = [4, -2]'; H = [8, -2]';
X = [A, B, C, D, E, F, G, H];
% configurare retea SOM
SOMnet = selforgmap([2 2]);
% antrenarea retea SOM
SOMnet = train(SOMnet,X);
% vizualizare retea SOM
view(SOMnet)
```

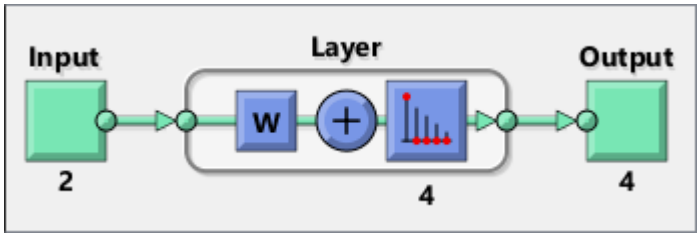


Figura 4.24. Arhitectura rețelei SOM

2. Afișarea stratului de ieșire: topologie și fiecare neuron pentru câți vectori a fost declarat câștigător.

```
% afisare cati vectori sunt alocati fiecarui neuron din stratul de iesire
plotsomhits(SOMnet, X)
```

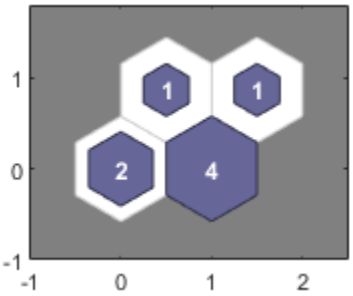


Figura 4.25. Câți vectori din X sunt alocați fiecărui neuron din stratul de ieșire

3. Afișarea ponderilor rețelei în urma antrenării.

```
% ponderile retelei
disp('ponderile retelei sunt:')
ponderi = SOMnet.IW{1,1}
```

```
ponderi =
    3.0000    5.0000
    3.5000    0.5000
    8.0000    2.0000
    8.0000   -2.0000
```

Semnificația ponderilor de mai sus este următoarea:

- Neuronul din stratul de ieșire de coordonate (0, 0), adică cel căruia i-au fost alocați 2 vectori din lotul `X`, are asociat setul de ponderi [3, 5]
- Neuronul din stratul de ieșire de coordonate (1, 0), adică cel căruia i-au fost alocați 4 vectori din lotul `X`, are asociat setul de ponderi [3.5, 0.5] etc

4. Simularea rețelei folosind lotul de antrenare `X`

```
% simulare retea SOM
disp('iesirile retelei sunt:')
O = sim(SOMnet, X)
```

```
O =
     1     1     0     0     0     0     0     0
     0     0     1     1     1     0     1     0
     0     0     0     0     0     1     0     0
     0     0     0     0     0     0     0     1
```

Semnificația ieșirilor de mai sus este următoare:

- Primul vector din matricea O este $[1, 0, 0, 0]^T$, adică primul vector din `X` ($A = [1, 5]^T$) este asociat primului neuron din stratul de ieșire, cel de coordonate `(0, 0)`
- Al doilea vector din matricea O este $[1, 0, 0, 0]^T$, adică al doilea vector din `X` ($B = [5, 5]^T$) este asociat primului neuron din stratul de ieșire, cel de coordonate `(0, 0)`
- Al treilea vector din matricea O este $[0, 1, 0, 0]^T$, adică al treilea vector din `X` ($C = [1, 1]^T$) este asociat celui de-al doilea neuron din stratul de ieșire, cel de coordonate `(1, 0)` etc
- Ultimul vector din matricea O este $[0, 0, 0, 1]^T$, adică ultimul vector din `X` ($H = [8, -2]^T$) este asociat celui de-al patrulea neuron din stratul de ieșire, cel de coordonate `(1, 1)` etc

Aplicația 4.4. Antrenarea și calibrarea unei rețele SOM pentru clasificarea în două clase a unei baze de date sintetice (generată pseudorandom)
Fie o bază de date conținând două clase de vectori bidimensionali, $clasa\ X$ și $clasa\ Y$, fiecare clasă conținând câte 100 de vectori cu valori generate pseudorandom și având distribuție normală astfel:

- pentru $clasa\ X$, media este $m_1 = [-5, -5]$ și dispersia este 2
- pentru $clasa\ Y$, media este $m_2 = [5, -10]$ și dispersia este 2

Să se antreneze o rețea SOM având topologie rectangulară de 5 x 5, astfel încât să separe cele două clase. Să se calibreze apoi rețeaua (să se determine corespondența dintre fiecare neuron din stratul de ieșire și clasa corespunzătoare).

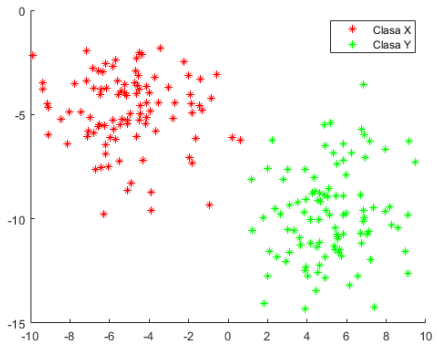
Implementarea în MATLAB este următoarea:

1. Se generează pseudorandom baza de date.

```
nrElementeClasa = 100;
m1 = [-5,-5]; m2 = [5, -10]; dispersie = 2;
X = [normrnd(m1(1), dispersie,[1,nrElementeClasa]);
     normrnd(m1(2), dispersie,[1,nrElementeClasa])];
Y = [normrnd(m2(1), dispersie,[1,nrElementeClasa]);
     normrnd(m2(2), dispersie,[1,nrElementeClasa])];
database = [X, Y];
```

2. Se reprezintă în spațiu 2-*D* punctele din clasele *X* și *Y*.

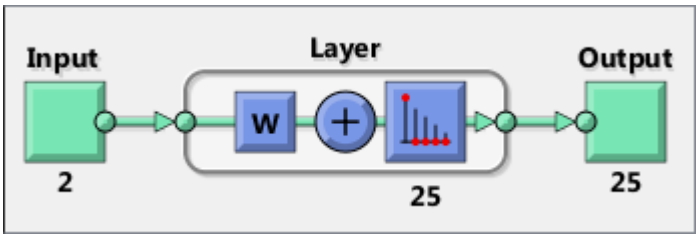
```
figure(1)
hold on
plot(X(1,:),X(2,:),'*r')
plot(Y(1,:),Y(2,:),'*g')
hold off
legend('Clasa X', 'Clasa Y')
```



3. Se antrenează nesupervizat rețeaua (clustering).

Se implementează o rețea rectangulară plană de dimensiune 5 x 5 și vecinătate 3.

```
dim_SOM = 5;
net = selforgmap([dim_SOM, dim_SOM],100,3,'gridtop');
% se antreneaza reteaua SOM
net = train(net,database);
% vizualizare arhitectura retea SOM utilizata.
figure(2)
view(net)
```



Se afișează stratul de ieșire: topologie și fiecare neuron pentru câți vectori a fost declarat câștigător.

```
% afisare cati vectori sunt alocati fiecarui neuron din stratul de iesire
plotsomhits(net, database)
```

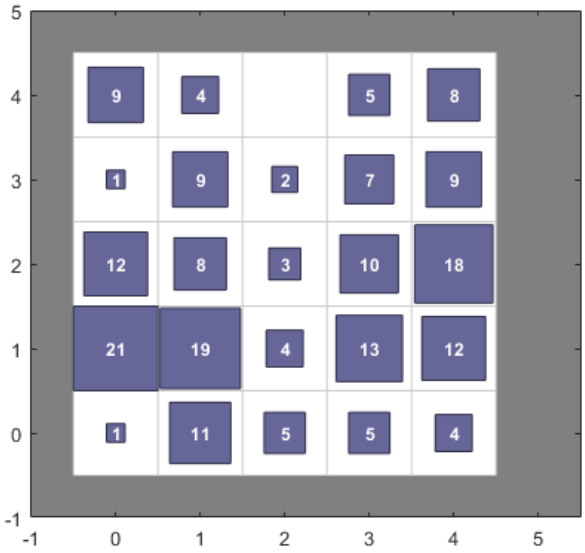
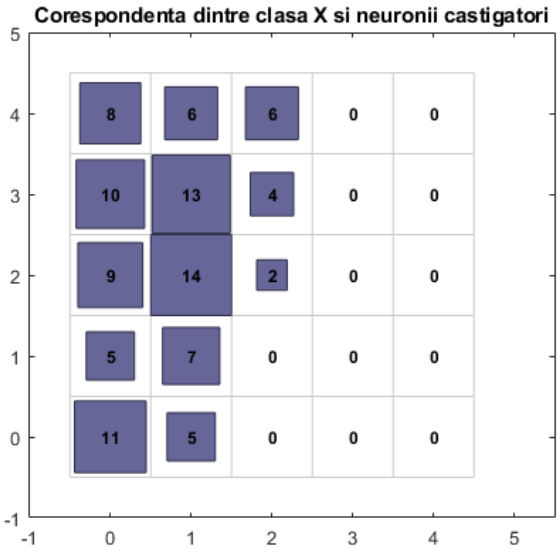


Figura 4.26. Câți vectori din `database` sunt alocați fiecărui neuron din stratul de ieșire

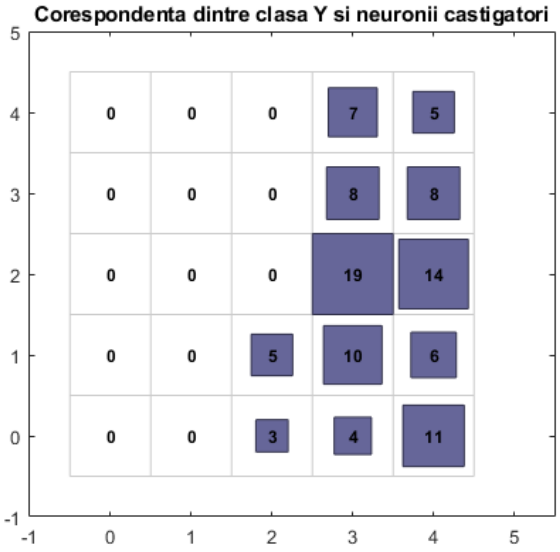
Se afișează corespondența dintre fiecare neuron din stratul de ieșire și numărul de vectori din *clasa X* pentru care neuronul respectiv a fost declarat câștigător.

```
plotsomhits(net,X)
title('Corespondenta dintre clasa X si neuronii castigatori')
```



Se afișează corespondența dintre fiecare neuron din stratul de ieșire și numărul de vectori din *clasa Y* pentru care neuronul respectiv a fost declarat câștigător.

```
plotsomhits(net,Y)
title('Corespondenta dintre clasa Y si neuronii castigatori')
```



4. Se calibrează rețeaua SOM.

Etapa de calibrare are ca scop corespondența dintre fiecare neuron din stratul de ieșire și clasele *X* și *Y*.

```
testX = sim(net,X);
% Stabilirea corespondentei dintre fiecare neuron din stratul de iesire si clasa X:
% se calculeaza distanta euclidiana dintre fiecare vector din clasa X si fiecare nod
din stratul de iesire al retelei SOM.
% matricea testX va contine valoarea 1 pentru distanta minima si 0 in rest
% matricea testX va avea dim_SOM x dim_SOM linii(25 linii) si nrElementeClasa coloane
% se transforma matricea testX intr-o matrice test_reshX cu dim_SOM linii, dim_SOM
coloane si nrElementeClasa straturi
test_reshX = reshape(testX,dim_SOM,dim_SOM,size(X,2));
% test_reshX  va contine cate un strat pentru fiecare vector din clasa X;
% stratul N va contine distantele dintre al N-lea vector din clasa X si toti neuronii
din stratul de iesire al retelei SOM.
% Pentru distanta minima va fi valoarea 1 iar in rest 0
% practic, fiecare strat din test_reshX  va fi o matrice cu 5 linii si 5 coloane, ce
va contine valoarea 1 pentru neuronul al carui set de ponderi se apropie cel mai mult
cu vectorul din clasa X si 0 in rest
% se insumeaza pe straturi matricea test_reshX
SOMhitsX = sum(test_reshX,3);
% fiecare neuron din stratul de iesire va arata numarul de vectori din clasa X pentru
care este castigator
% SOMhitsX este de fapt o matrice ce contine elementele reprezentate in Figura 3
% se calculeaza procentual de cate ori este castigator fiecare neuron pentru vectorii
din clasa X
SOMhits_percX = SOMhitsX./sum(sum(SOMhitsX))*100;
% se calculeaza distanta euclidiana dintre fiecare vector din clasa Y si fiecare nod
din stratul de iesire al retelei SOM
% in continuare se procedeaza ca si in cazul clasei X
testY = sim(net,Y);
test_reshY = reshape(testY,dim_SOM,dim_SOM,size(Y,2));
% SOMhitsX este de fapt o matrice ce contine elementele reprezentate in Figura 4
SOMhitsY = sum(test_reshY,3); SOMhits_percY = SOMhitsY./sum(sum(SOMhitsY))*100;
% folosind logica majoritara se decide apartenenta fiecarui neuron la o clasa
SOMhits(:,:,1) = SOMhits_percX; SOMhits(:,:,2) = SOMhits_percY;
[val, hartaSOM] = max(SOMhits,[],3);
% afisare harta SOM calibrata
table(rot90(hartaSOM))
```

```
Harta SOM calibrata
___________________

1    1    1    2    2
1    1    1    2    2
1    1    1    2    2
1    1    2    2    2
1    1    2    2    2
```

Unde 1 reprezintă *clasa X* și 2 reprezintă *clasa Y*.

Aplicația 4.5. Antrenarea unei rețele SOM pentru învățarea a două cifre din baza de date MNIST

Să se antreneze o rețea SOM cu topologie hexagonală, pentru clasificarea cifrelor 1 și 2 din baza de date MNIST (pentru o execuție mai rapidă a programului se vor folosi doar primele 500 de imagini cu cifra 1 și primele 500 de imagini cu cifra 2 din baza de date MNIST). Se vor afișa:

- arhitectura rețelei
- corespondența dintre clasa *cifrei 1* și neuronii câștigători
- corespondența dintre clasa *cifrei 2* și neuronii câștigători
- rețeaua SOM calibrată
- ponderile finale obținute în urma antrenării (reprezentarea va fi sub formă de imagini)

De asemenea, să se calculeze:

- procentul de imagini clasificate corect cu toate imaginile conținând *clasa 1* (nu doar cele 500 folosite la antrenare)
- procentul de imagini clasificate corect cu toate imaginile conținând *clasa 2* (nu doar cele 500 folosite la antrenare)
- matricea de confuzie

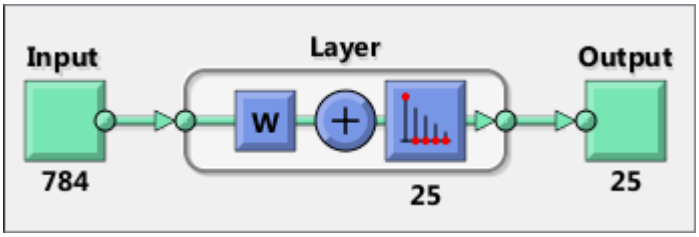
Implementarea în MATLAB este următoarea:

1. Citirea bazei de date din fișierul *excel*.

```
database_MNIST = xlsread('mnist_train_10k.xlsx');
% Baza de date folosita este una restransa, folosind doar primele 10000 de imagini.
% Dintre acestea se vor folosi doar primele 500 de imagini cu cifra 1 si primele 500
de imagini cu cifra 2.
cifreInput = [1, 2];
NrCifre = length(cifreInput);
NrImaginiPerCifra = 500;
% se cauta in baza de date pozitiile din fisierul excel pe care se afla cifra 1
cifra_index1 = find((database_MNIST(:,1)==cifreInput(1)));
% se cauta in baza de date pozitiile din fisierul excel pe care se afla cifra 2
cifra_index2 = find((database_MNIST(:,1)==cifreInput(2)));
% se formeaza vectorul cifre_index ce contine pozitiile din fisierul excel pe care se
afla cifra 1 si cifra 2
cifre_index = [cifra_index1(1:NrImaginiPerCifra); cifra_index2(1:NrImaginiPerCifra)];
% se salveaza in matricea cifre toate imaginile cu cifrele 1 si 2;
% fiecare imagine este salvata pe cate o linie din matricea cifre;
% pe prima pozitie este cifra, iar pe urmatoarele M x N coloane (784) sunt valorile
pixelilor
% baza de date ce va fi folosita pentru antrenare nu foloseste si prima coloana
database_train = database_MNIST(cifre_index,2:end)';
```

2. Implementarea și antrenarea rețelei SOM pentru clasificarea cifrelor 1 și 2

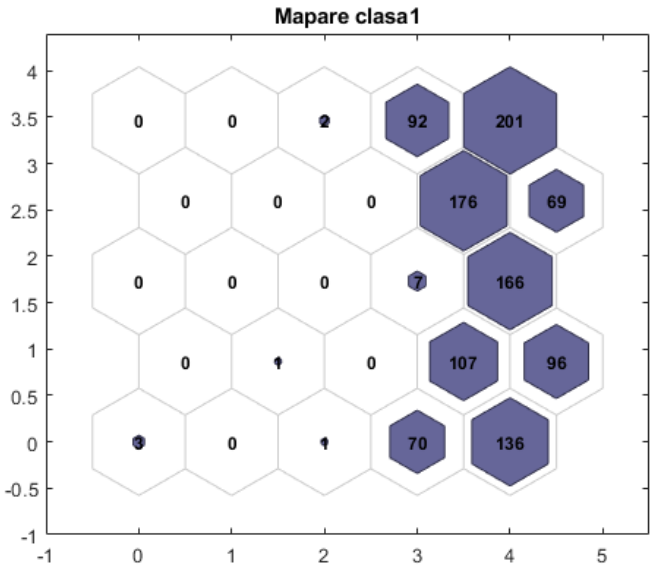
```
dim_SOM = 5;
% se creaza o retea SOM hexagonala de dimensiune dim_SOM x dim_SOM
 net = selforgmap([dim_SOM, dim_SOM]);
% se antreneaza reteaua SOM
 net = train(net,database_train);
% vizualizare arhitectura retea SOM utilizata.
figure(), view(net)
```

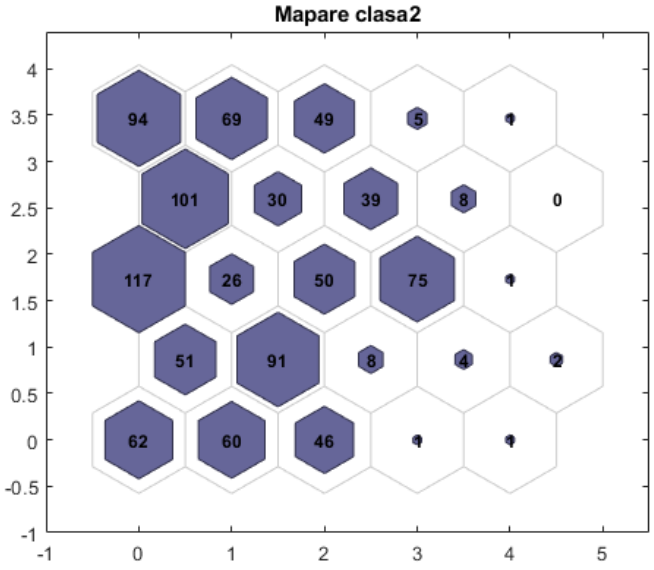


3. Calibrarea rețelei SOM

```
 clasa1 = database_MNIST(cifra_index1,2:end)';
 % mapare retea pentru elementele din prima clasa
 test_clasa1 = sim(net,clasa1);
 test_clasa1_reshape = reshape(test_clasa1,dim_SOM,dim_SOM,size(test_clasa1,2));
 SOMhits_clasa1 = sum(test_clasa1_reshape,3);
 SOMhits_clasa1_proc = SOMhits_clasa1/sum(sum(SOMhits_clasa1));
 % se afiseaza de cate ori a fost declarat castigator un neuron din stratul de iesire
pentru imaginile din clasa 1
 figure(), plotsomhits(net,clasa1), title('Mapare clasa1')

 clasa2 = database_MNIST(cifra_index2,2:end)';
 % mapare retea pentru elementele din a doua clasa
 test_clasa2 = sim(net,clasa2);
 test_clasa2_reshape = reshape(test_clasa2,dim_SOM,dim_SOM,size(test_clasa2,2));
 SOMhits_clasa2 = sum(test_clasa2_reshape,3);
 SOMhits_clasa2_proc = SOMhits_clasa2/sum(sum(SOMhits_clasa2));
 % se afiseaza de cate ori a fost declarat castigator un neuron din stratul de iesire
pentru imaginile din clasa 2
 figure(), plotsomhits(net,clasa2), title('Mapare clasa2')
 % calcul procentual
 SOMhits_all_proc(:,:,1) = SOMhits_clasa1_proc;
 SOMhits_all_proc(:,:,2) = SOMhits_clasa2_proc;
 % folosind logica majoritara se decide fiecare neuron carei clase ii este
reprezentativ
[val, hartaSOM] = max(SOMhits_all_proc,[],3);
% afisare harta SOM calibrata
table(rot90(hartaSOM))
```

```
Harta SOM
_____________________

2     2     2     1     1
2     2     2     1     1
2     2     2     2     1
2     2     2     1     1
2     2     2     1     1
```





4. Afișarea ponderilor sub formă de imagini.

```
ponderi = net.IW{1,1};
% variabila ponderi are dim_SOM x dim_SOM linii si M x N coloane (unde M si N
reprezinta dimensiunile imaginii, in cazul de fata sunt 784 de coloane)
% pe fiecare coloana se salveaza setul de ponderi asociat unui neuron din stratul de
iesire
figure()
for  i = 1:dim_SOM
    for j = 1:dim_SOM
        pondere_vector = ponderi((i-1)*dim_SOM+j,:);
        % se transforma vectorul de ponderi in imagine
        pondere_imag = reshape(pondere_vector,28,28)';
        % se afiseaza ca imagine setul de ponderi asociat fiecarui neuron din stratul
de iesire
        subplot(dim_SOM, dim_SOM,(dim_SOM-i)*dim_SOM+j)
        imshow(pondere_imag/255)
    end
end
```

5. Calcularea scorului de clasificare corectă pentru toate cifrele din *clasa 1* existente în toată baza de date de 10000 de imagini.

```
clasa1 = database_MNIST(cifra_index1,2:end)';
for i = 1:size(clasa1,2)
    sim_vectorTest = sim(net, clasa1(:,i));
    sim_reshape = reshape(sim_vectorTest,dim_SOM, dim_SOM);
    poz = find(sim_reshape==1);
    etichete_clasa1_prezise(i) = hartaSOM(poz);
end
scorClasificareClasa1 =
length(find(etichete_clasa1_prezise==cifreInput(1)))/size(clasa1,2)*100;
disp(['Scor clasificare pentri clasa 1 = ', num2str(scorClasificareClasa1),'%'])
```

```
Scor clasificare pentru clasa 1 = 98.7578%
```

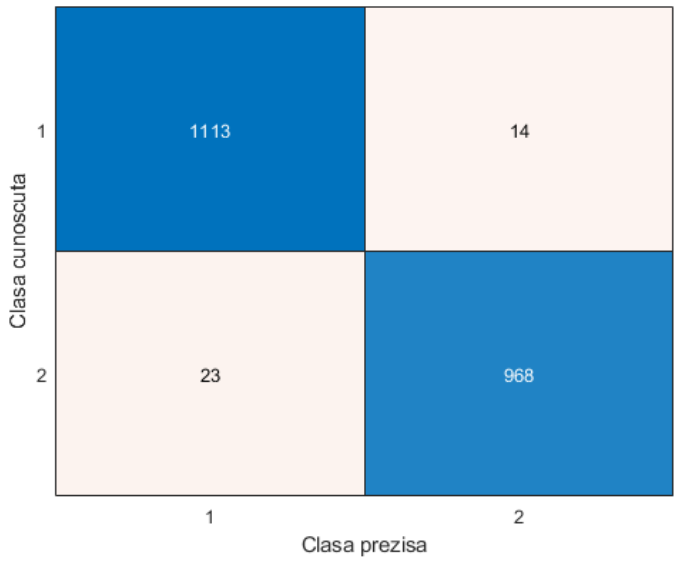
6. Calcularea scorului de clasificare corectă pentru toate cifrele din clasa 2 existente în toată baza de date de 10000 de imagini.

```
clasa2 = database_MNIST(cifra_index2,2:end)';
for i = 1:size(clasa2,2)
    sim_vectorTest = sim(net, clasa2(:,i));
    sim_reshape = reshape(sim_vectorTest,dim_SOM, dim_SOM);
    poz = find(sim_reshape==1);
    etichete_clasa2_prezise(i) = hartaSOM(poz);
end
scorClasificareClasa2 =
length(find(etichete_clasa2_prezise==cifreInput(2)))/size(clasa2,2)*100;
disp(['Scor clasificare pentri clasa 2 = ', num2str(scorClasificareClasa2),'%'])
```

```
Scor clasificare pentru clasa 2 = 97.6791%
```

7. Afișarea matricei de confuzie.

```
etichete_cunoscute_clasa1 = cifreInput(1) *ones(1, size(clasa1,2));
etichete_cunoscute_clasa2 = cifreInput(2) *ones(1, size(clasa2,2));
etichete_cunoscute = [etichete_cunoscute_clasa1, etichete_cunoscute_clasa2];
etichete_prezise = [etichete_clasa1_prezise, etichete_clasa2_prezise];
matriceConfuzie = confusionmat(etichete_cunoscute,etichete_prezise);
figure(), confusionchart(matriceConfuzie), xlabel('Clasa prezisa'), ylabel('Clasa
cunoscuta')
```



4.3.4. Antrenarea rețelei SOM cu ajutorul *toolbox*-urilor MATLAB

Ca și în cazul rețelelor MLP prezentate anterior, antrenarea rețelei SOM se poate realiza folosind *toolbox*-urilor MATLAB-*ului.* Pentru exemplificare vom folosi același exemplu ca cel din *Capitolul 3, Aplicația 3*, și anume antrenarea unei rețele SOM pentru învățarea a două cifre din baza de date MNIST.

Pentru versiunea de MATLAB *2019a*, se accesează *tab*-ul *Apps*, categoria *Machine Learning and Deep Learning, toolbox*-ul *Neural Net Clustering.*

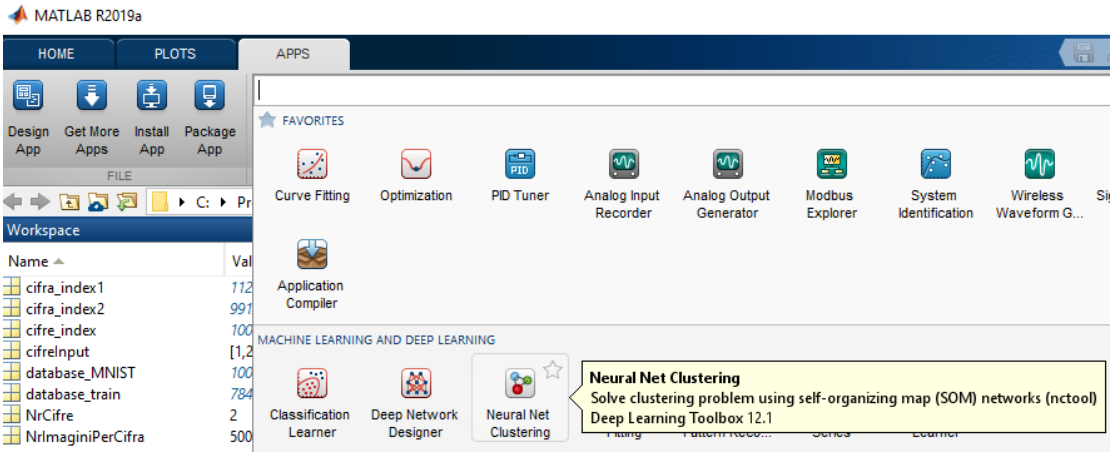


Figura 4.27. *Toolbox*-ul de *Neural Net Clustering*

Toolbox-ul *Neural Net Clustering* va deschide următoarea interfață grafică.

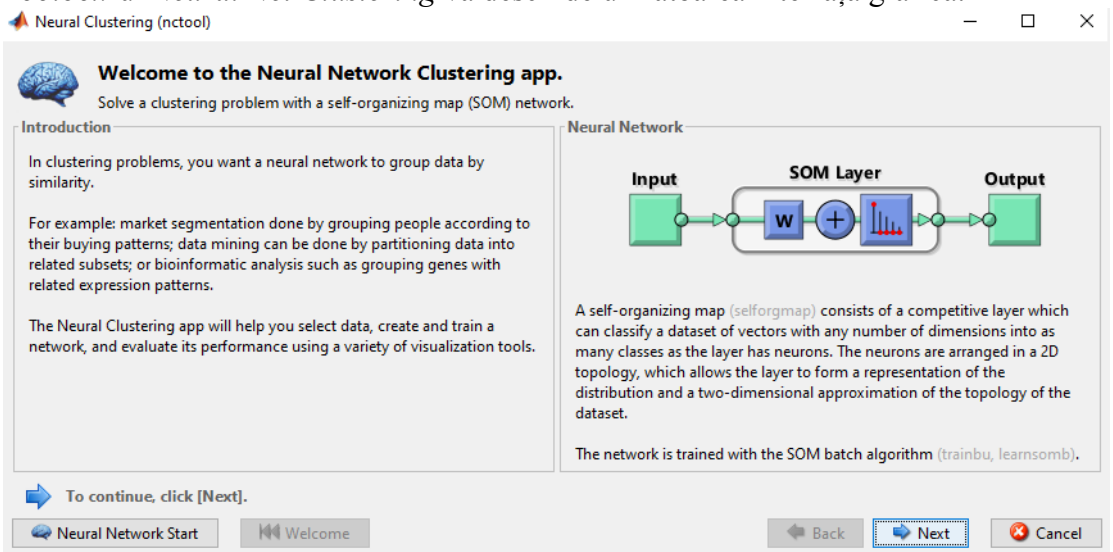


Figura 4.28. *Toolbox*-ul *Neural Net Clustering (fereastra nctool)*

În continuare, pentru a încărca baza de date pentru care se dorește antrenarea rețelei se va alege opțiunea *Next*. Se va deschide următoarea fereastră:

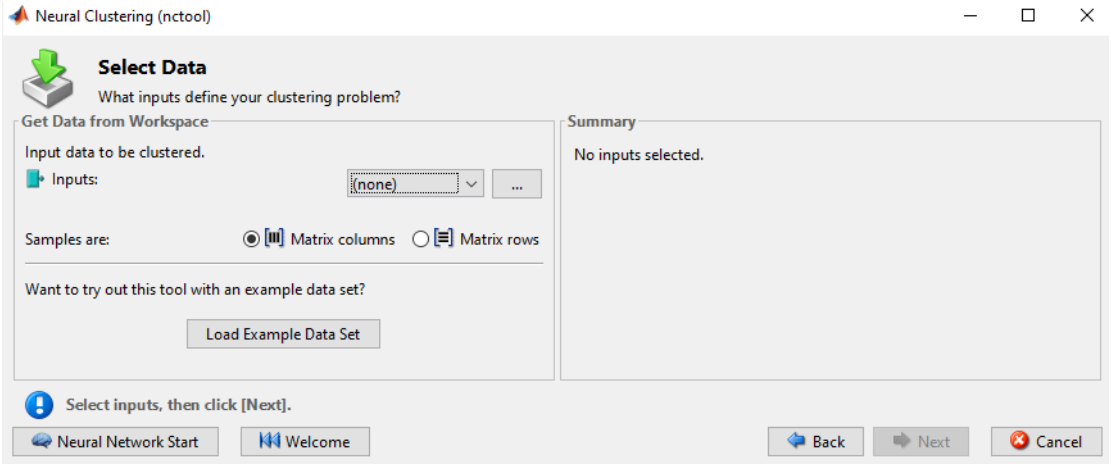


Figura 4.29. Încărcarea bazei de date

Spre deosebire de rețeaua MLP care este supervizată și are nevoie și de ieșirile dorite (*Targets*), rețeaua SOM nu are nevoie pentru antrenare și de *Targets*, fiind o metodă nesupervizată de învățare.

Incărcarea datelor se poate face din fereastra *Workspace*, dacă acestea există acolo. Dacă nu există, trebuie mai întâi încărcate. Pentru exemplul nostru, datele sunt deja în *Workspace* (*database_train* din *Anexa 2*); pentru parametrul *Inputs* se va selecta variabila *database_train.* Deoarece fiecare coloană din *database_train* conține câte o imagine vectorizată, se va alege opțunea *Matrix columns* pentru aranjarea datelor.

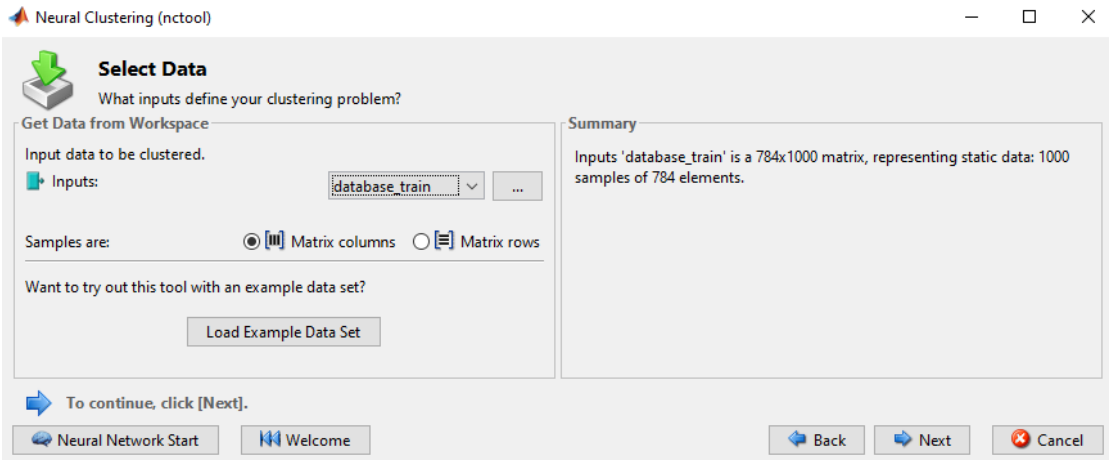


Figura 4.30. Încărcarea bazei de date - selectarea intrărilor

Următorul pas îl constituie configurarea rețelei SOM, singurul parametru care poate fi modificat fiind dimensiunea stratului de ieșire.

Folosind *toolbox-ul* pentru învățarea rețelei SOM, topologia impusă este cea rectangulară iar neuronii din stratul de ieșire sunt aranjați într-o rețea de N x N, unde N este parametrul *Size of two-dimensional Map* din interfață. Se va alege $N = 5$.

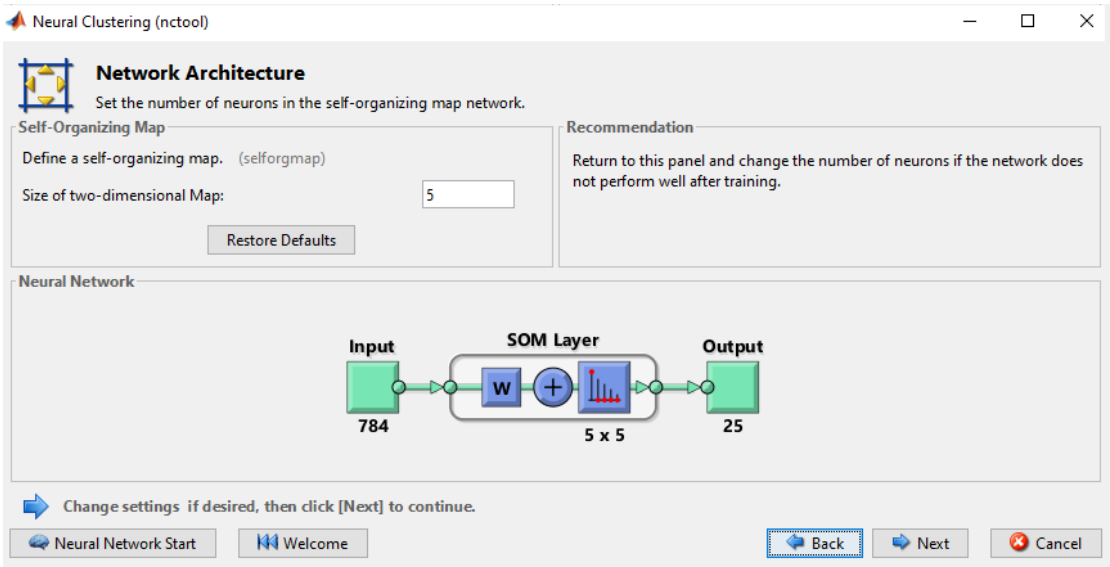


Figura 4.31. Arhitectura rețelei SOM

Următorul pas este cel de antrenare a rețelei.

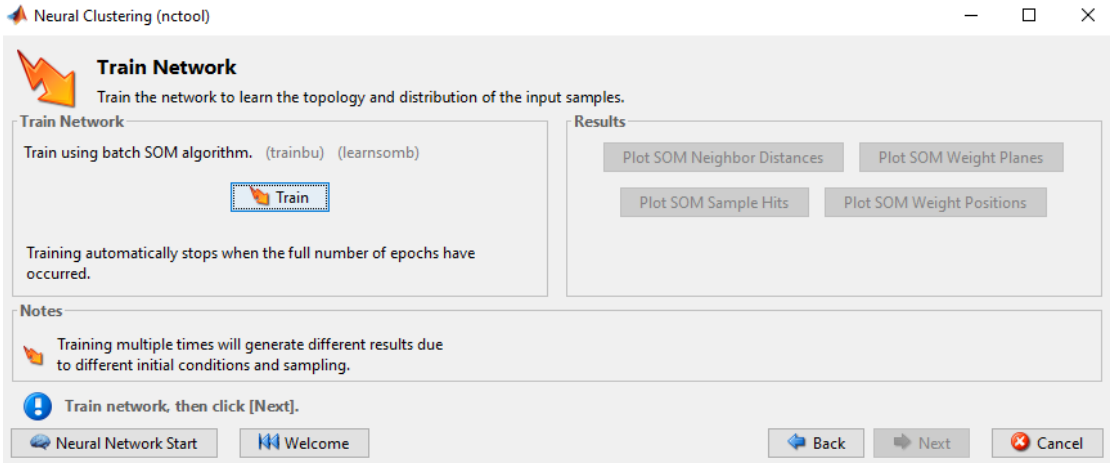


Figura 4.32. Antrenarea rețelei SOM

Prin apăsarea butonului de *Train* este deschisă fereastra *nntraintool* cu parametrii *default* (parametrii de învățare și de oprire a învățării) setați de interfață.

Următorul pas oferă posibilitatea de a relua etapa de antrenare (*Train Again*) sau de a afișa grafic performanțele antrenării.

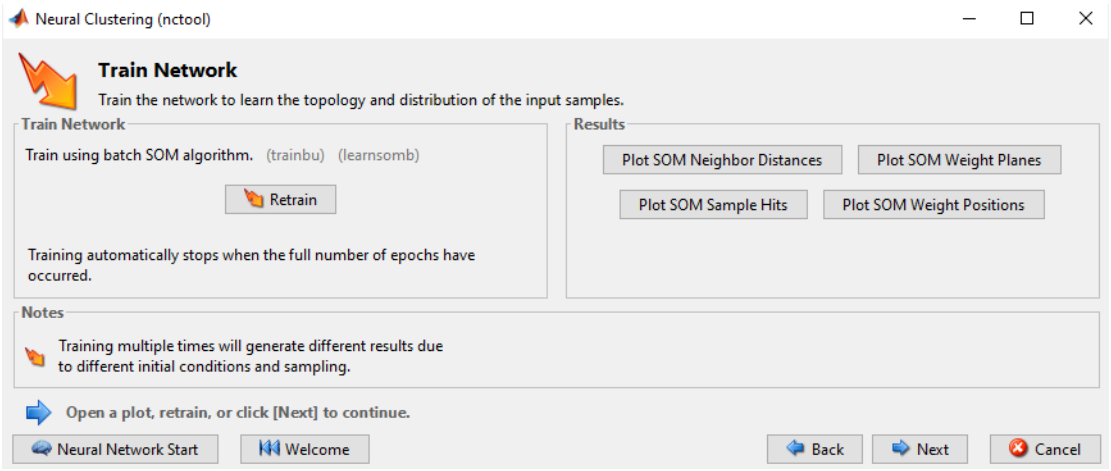


Figura 4.33. Rezultatele învățării

Pentru a vedea harta obținută în urma învățării (de câte ori a fost declarat câstigator fiecare neuron din stratul de iesire) se alege opțiunea *Plot SOM Sample Hits*.

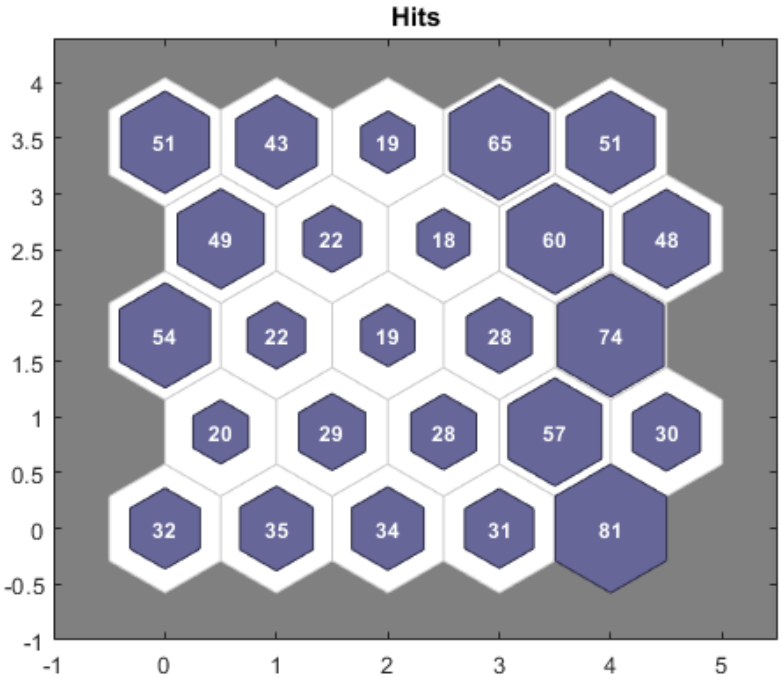


Figura 4.34. Harta obținută în urma învățării nesupervizate

Următorul pas oferă posibilitatea de a relua etapa de antrenare (*Train Again*), de a reconfigura rețeaua alegând un alt număr de neuroni pentru stratul de ieșire (*Adjust Network Size*) de a folosi un alt set de date pentru învățare (*Import Larger Data Set*) sau de a testa rețeaua deja antrenată pe un alt set de date (*Optionally perform additional tests*) înainte de generarea fișierului **.m*.

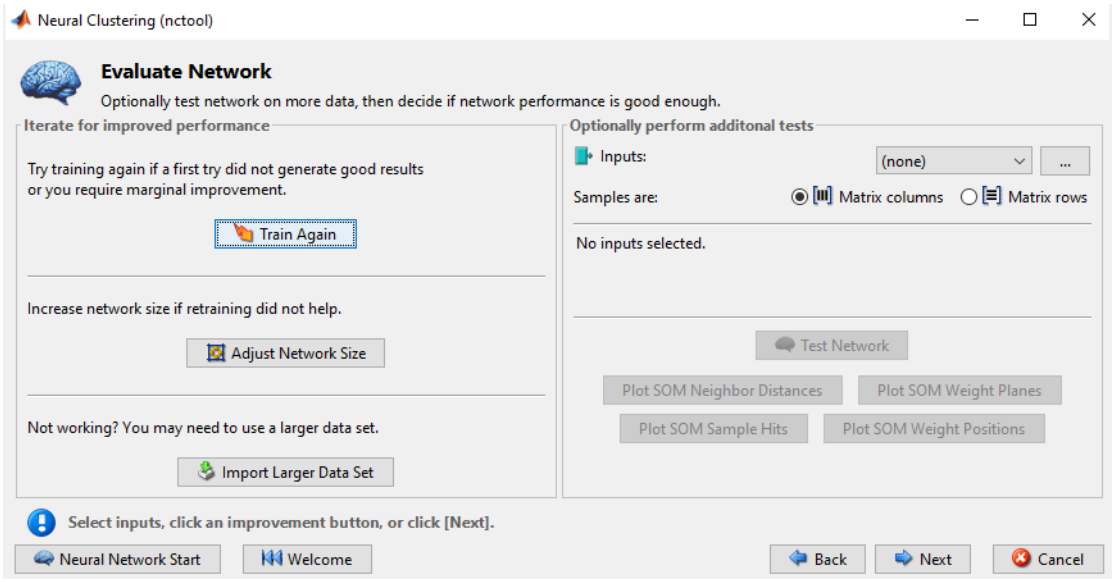


Figura 4.35. Evaluare rețea SOM

Ultimul pas îl reprezintă salvarea rezultatelor și a rețelei în *Workspace*, pentru a fi folosită mai departe de scripturi MATLAB, pentru a fi salvată în fisiere **.mat*, etc. De asemenea, din acest ecran se pot genera fișiere **.m* care să antreneze rețeaua la fel cum o face aplicația, fără a mai fi nevoie de intervenția utilizatorului la fiecare pas.

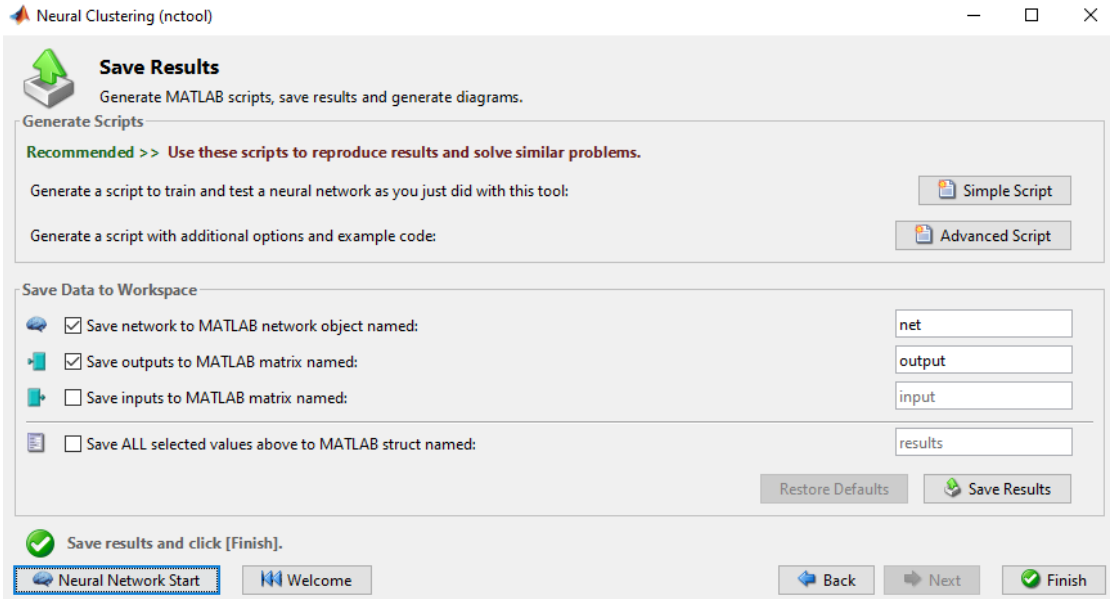


Figura 4.36. Generare cod sursă și salvare rezultate

```
% Solve a Clustering Problem with a Self-Organizing Map
% Script generated by Neural Clustering app
% Created 03-Nov-2019 11:49:34
%
% This script assumes these variables are defined:
%
%   database_train - input data.

x = database_train;

% Create a Self-Organizing Map
dimension1 = 5;
dimension2 = 5;
net = selforgmap([dimension1 dimension2]);

% Train the Network
[net,tr] = train(net,x);

% Test the Network
y = net(x);

% View the Network
view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotsomtop(net)
%figure, plotsomnc(net)
%figure, plotsomnd(net)
%figure, plotsomplanes(net)
%figure, plotsomhits(net,x)
%figure, plotsompos(net,x)
```

În urma antrenării rețelei folosind *toolbox*-ul *Neural Net Clustering* din MATLAB se obține doar harta necalibrată (nu se cunoaște corelația dintre neuronii din stratul de ieșire și clase). Pentru a realiza calibrarea rețelei se completează codul de mai sus la fel cum s-a procedat în *Capitolul 4, Aplicația 2*, începând cu *punctul 4*.

Capitolul 5

Metode de segmentare

Segmentarea se referă la partiționarea unei imagini în seturi de pixeli, în general pentru delimitarea obiectelor de interes. Deși din punct de vedere spațial segmentarea nu pare legată de clasificare, trecerea în spațiul culorilor reduce problema la o clasificare (adesea nesupervizată), cu câteva constrângeri suplimentare legate de poziționarea spațială relativă a pixelilor unii față de alții. De exemplu, în majoritatea prelucrărilor de imagini care implică segmentare există și operații de pre- sau post-procesare care să filtreze sau elimine pixeli izolați.

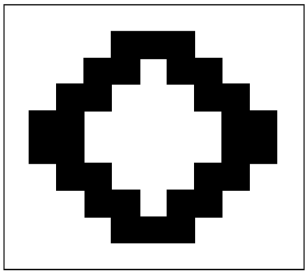
5.1. Scurtă introducere asupra reprezentării imaginilor în MATLAB

Indiferent de conținutul imaginii (că e vorba de imagine grayscale, RGB, imagine satelitară etc) în MATLAB imaginile sunt salvate sub formă matriceală cu unul sau mai multe straturi.

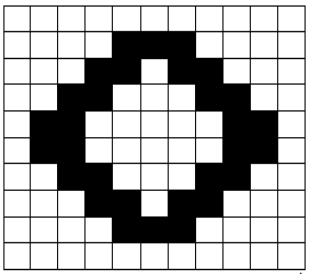
În legatură cu tipul de date al unei imagini, este important de menționat următorul aspect: dacă imaginea este în format `double`, trebuie avut grijă ca elementele matricei să fie în intervalul [0, 1]; dacă imaginea este în format `uint8`, trebuie avut grijă ca elementele matricei să fie în intervalul [0, 255]. Pentru a evita multe probleme care pot apărea datorită inadvertenței dintre tipul de date și valorile pixelilor, recomandat ar fi ca imaginea să fie convertită de la început în format `double` cu valori în intervalul [0, 1]

În continuare vom prezenta pe scurt cele mai uzuale tipuri de imagini și reprezentarea acestora în MATLAB.

Imagine binară. O imagine binară (*alb – negru*) poate fi reprezentată folosind o matrice ce conține numai valorile 0 și 1 (unde 0 reprezintă *negru* iar 1 reprezintă *alb*).



(a)



(b)

1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	0	0	0	1	1	1	1
1	1	1	0	0	1	0	0	1	1	1
1	1	0	0	1	1	1	0	0	1	1
1	0	0	1	1	1	1	1	0	0	1
1	0	0	1	1	1	1	1	0	0	1
1	1	0	0	1	1	1	0	0	1	1
1	1	1	0	0	1	0	0	1	1	1
1	1	1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1

(c)

Figura 5.1. (a) imagine alb-negru (b) reprezentarea în pixeli a imaginii
(c) reprezentarea matriceală a imagini

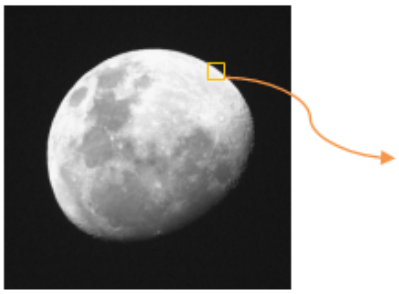
De obicei, în MATLAB se folosesc imaginile binare în urma segmentării unei imagini în 2 clase. Tipul de date asociat unei astfel de imagini este de obicei tipul `logical`.

Dacă imaginea se numește `imag`, atunci pentru a accesa elementul din colțul stânga-sus al imaginii se folosește comanda:

```
>> pixel = imag(1,1)
```

Unde `pixel` va fi un scalar ce va avea fie valoarea 1 fie valoarea 0.

Imagine grayscale. Imaginile cu niveluri de gri (imagini *grayscale*) pot fi reprezentate ca matrice, fiecare element al matricei reprezentând intensitatea pixelului respectiv. Valorile intensităților se exprimă în mod uzual pe 8 biți, cu alte cuvinte sunt disponibile 256 de niveluri de gri pentru intensitatea fiecărui pixel.



26	27	24	25	25	25	20	24	24	24
29	27	26	25	28	24	24	25	25	25
94	34	28	26	27	23	25	24	22	24
237	171	59	29	27	25	25	25	25	24
246	245	220	131	41	28	26	26	26	27
239	243	247	243	187	67	29	27	26	24
239	241	245	246	246	217	98	31	29	26
240	239	239	241	244	246	233	141	39	27
236	238	240	236	240	241	246	237	168	47
235	233	235	232	234	235	240	245	241	182

Figura 5.2. Reprezentarea matriceală a unei imagini grayscale

Conținutul unei imagini grayscale digitale este o matrice cu aceeași dimensiune cu cea a imaginii (numărul de linii din matrice coincide cu numărul de pixeli pe verticală din imagine și numărul de coloane din matrice corespunde cu numărul de pixeli pe orizontală din imagine). De exemplu, o imagine *grayscale* având dimensiunea de 200 x 300 pixeli nu este altceva decât o matrice cu 200 de linii și 300 de coloane.

Imagine color. Conținutul unei imagini digitale color în format RGB reprezintă o matrice cu trei straturi (stratul de roșu-*Red*, stratul de verde-*Green* și stratul de albastru-*Blue*) fiecare strat fiind o matrice cu aceeași dimensiune cu cea a imaginii.

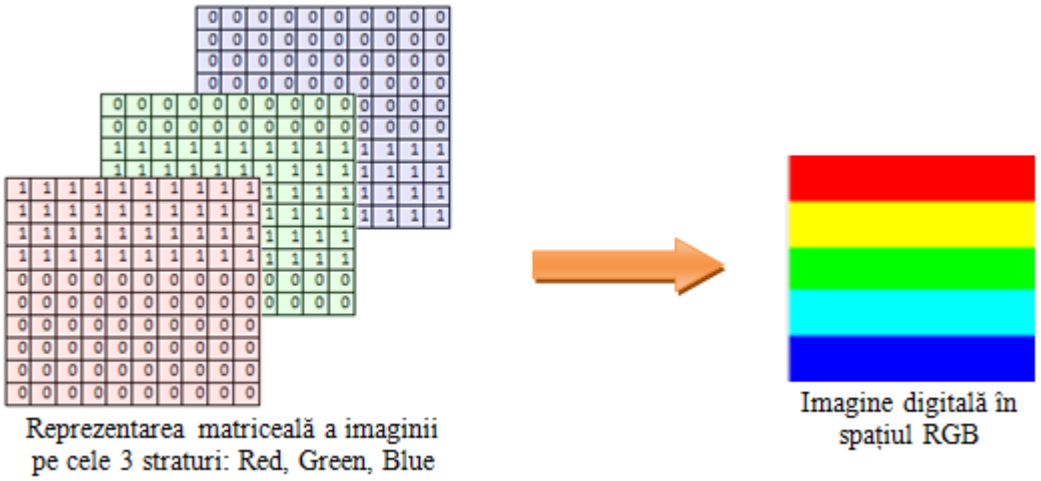


Figura 5.3. Reprezentarea matriceală a unei imagini digitale color

Dacă imaginea se numește `imag`, atunci pentru a accesa elementul din colțul stânga-sus al imaginii se folosește comanda:

```
>> pixel = imag(1,1,:)
```

Unde `pixel` va fi un vector cu 3 elemente: `pixel(1)` = componenta de roșu, `pixel(2)` = componenta de verde, `pixel(3)` = componenta de albastru.

Citirea imaginilor folosind funcția `imread`

Funcția `imread` permite citirea imaginilor binare, grayscale și color.

Sintaxă: `imag = imread('nume_imagine', 'ext')`, unde:

- parametrul `nume_imagine` reprezintă numele imaginii
- parametrul `ext` reprezintă extensia imaginii (`jpg`, `png`, `bmp` etc)

Observație. Dacă lipsește parametrul `ext`, atunci numele imaginii trebuie să conțină și extensia (ex: `imread('imagine.jpg')`). Dacă imaginea nu se află în folderul curent, la numele imaginii trebuie specificată întreaga cale.

Dacă imaginea este grayscale atunci matricea `imag` va avea dimensiunea `M` x `N`, unde `M` reprezintă numărul de linii și `N` numărul de coloane.

Dacă imaginea este color, atunci matricea `imag` va avea dimensiunea `M` x `N` x 3, cele 3 straturi reprezentând planurile RGB (*Observație*: pentru imaginile `tiff` pot exista mai multe straturi).

Afișarea unei imagini folosind funcția `imshow`

Funcția `imshow` permite afișarea unei imagini binare, grayscale sau color.

Sintaxă*:* `imshow(imag)`

Observații:

- Pentru o imagine color, dacă valorile imaginii sunt de tip `double`, atunci un pixel având valorile `[0,0,0]` reprezintă un pixel negru iar un pixel având valorile `[1,1,1]` reprezintă un pixel alb.
- Pentru o imagine color, dacă valorile imaginii sunt de tip `uint8` sau `uint16`, atunci un pixel având valorile `[0,0,0]` reprezintă un pixel negru iar un pixel având valorile `[255,255,255]` reprezintă un pixel alb.

Aplicație. Se citește și se afișează o imagine.

```
>>  imag = imread('corabie.jpg');
>>  figure(1), imshow(imag)
```

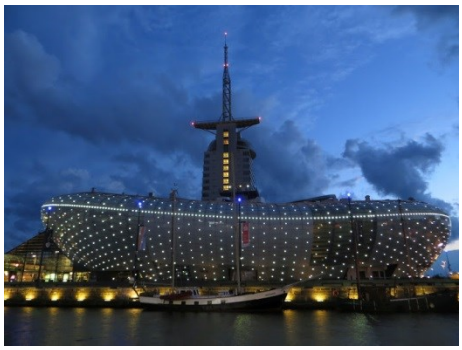


Figura 5.4. Exemplu de citire și afișare imagine în MATLAB

5.2. Operația de prăguire (en. *Thresholding*)

Introducere teoretică

Pentru o imagine grayscale, operația de prăguire constă în aplicarea unei valori de prag (T) față de care valorile de gri din imagine vor lua două valori {0, 1}.

$$imagSeg(x,y) = \begin{cases} 1, daca\ imag(x,y) \geq T \\ o, daca\ imag(x,y) < T \end{cases} \quad (5.1)$$

Pentru valorile de gri mai mari decât pragul T nivelurile de gri devin **1**, iar pentru valorile mai mici decât pragul T nivelurile de gri vor deveni **0**.

Implementare în MATLAB

Pentru implementarea operației de prăguire în MATLAB se poate folosi funcția `imbinarize`, care realizează segmentarea în diverse moduri, depinzând de parametrii de intrare. Pentru operația de prăguire cu prag impus T, sintaxa este următoarea:

Sintaxă: `BW = imbinarize(I,pragT)` unde:

- `I` = imaginea grayscale ce se dorește a fi binarizată
- `pragT` = pragul de comparație; `pragT` este un scalar în intervalul [0, 1].
- `BW` = imaginea binarizată (tipul de date `logical`)

Obs: pentru sintaxa completă `>> help imbinarize`

Aplicația 5.1. Să se binarizeze imaginea de mai jos. Pragul de segmentare va fi determinat din histograma imaginii.

Figura 5.5. Imagine originală

```
imag = imread('sah.jpg');
% conversie imagine color in imagine grayscale
imagGray = rgb2gray(imag);
% histograma imaginii pentru determinarea pragului
histograma = imhist(imagGray);
figure(1), bar([0:255], histograma)
xlabel('niveluri gri'), ylabel('numar aparitii niveluri gri')
```

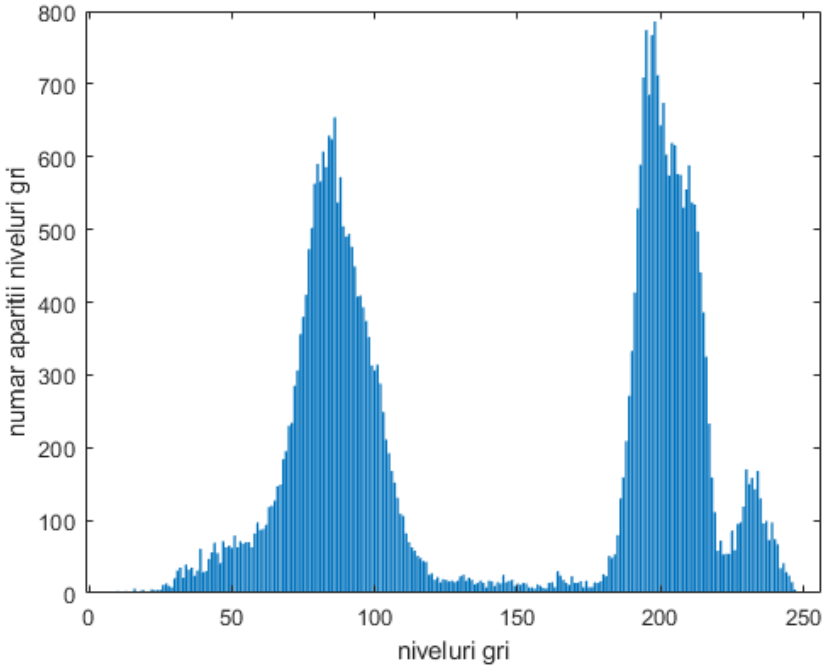


Figura 5.6. Histograma imaginii

Analizând histograma se deduce că pragul care separă cel mai bine clasele este aproximativ 160. Folosind acest prag (care va trebui mai întâi să fie normat, adică adus în intervalul [0, 1]) se va segmenta apoi imaginea.

```
pragT = 160/255;
% binarizarea imaginii folosind pragul determinat din analiza histogramei
BW = imbinarize(imagGray, pragT);
figure(2), imshow(BW)
```

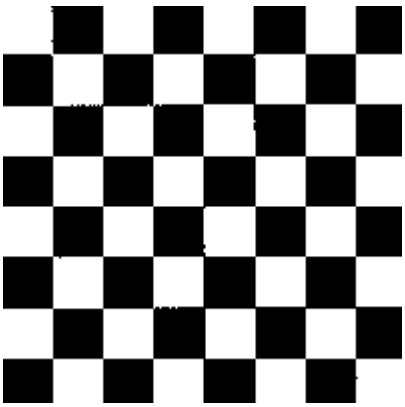


Figura 5.7. Imagine binarizată folosind operația de prăguire

5.3. Segmentarea cu prag optim – *metoda Otsu*

Introducere teoretică

Dezavantajul metodei implementate anterior este acela că utilizatorul trebuie să determine pentru fiecare imagine în parte care este pragul de segmentare. Dacă se dorește ca pragul să fie determinat automat, și nu introdus de utilizator, se poate folosi *metoda Otsu*; această metodă alege valoarea de prag ce maximizează varianța între două clase. Pragul va fi ales astfel încât cele două grupuri să fie cât mai "strânse", minimizând astfel suprapunerea. O măsură a omogenității grupului este varianța. Un grup cu omogenitate mare are o varianță mică, un grup cu omogenitate mică are varianță mare.

Fie o imagine cu niveluri de gri între 1 și L, având n_i pixeli cu intensitatea i, și histograma normată $p_i = n_i / N$ (unde N reprezintă numărul de pixeli din imagine). Pentru o valoare de prag k, varianța între clase este:

$$\sigma_B^2(k) = \frac{[\mu_T \omega(k) - \mu(k)]^2}{\omega(k)[1 - \omega(k)]} \tag{5.2}$$

unde: $\mu_T = \sum_{i=1}^{L} ip_i$, $\omega(k) = \sum_{i=1}^{k} p_i$, $\mu_k = \sum_{i=1}^{k} ip_i$

Valoarea optimă de prag k^* este aleasă astfel:

$$\sigma_B^2(k^*) = \underbrace{max}_{1<k<L} \sigma_B^2(k) \tag{5.3}$$

Implementare în MATLAB

Pentru a realiza binarizarea unei imagini grayscale în MATLAB folosind metoda *Otsu*, se poate folosi tot funcția `imbinarize` descrisă și folosită anterior, cu singura diferență că funcția nu va mai primi ca parametru de intrare pragul pentru segmentare, acesta fiind determinat automat conform algoritmului descris mai sus.

Sintaxă: `BW = imbinarize(I)` unde:

- `I` = imaginea grayscale ce se dorește a fi binarizată
- `BW` = imaginea binarizată (tipul de date `logical`)

Obs: funcția `imbinarize` face două lucruri:

- determină pragul optim de segmentare;
- cu pragul determinat, realizează binarizarea

Dacă se dorește să se afle pragul determinat de metoda *Otsu* cu care se face apoi binarizarea, se poate folosi funcția `graythresh`.

Sintaxă: `pragOtsu = graythresh(I)` unde:

- `I` = imaginea grayscale ce se dorește a fi binarizată
- `pragOtsu` = pragul de binarizare determinat cu metoda Otsu; pragul are valori în intervalul [0, 1]

Aplicația 5.2. Se va binariza imaginea folosită în aplicația anterioară (imaginea din *Figura 5.5*) folosind metoda Otsu.

```
imag = imread('sah.jpg');
% conversie imagine color in imagine grayscale
imagGray = rgb2gray(imag);
% binarizarea imaginii folosind metoda Otsu
BW = imbinarize(imagGray);
figure(1), imshow(BW)
```

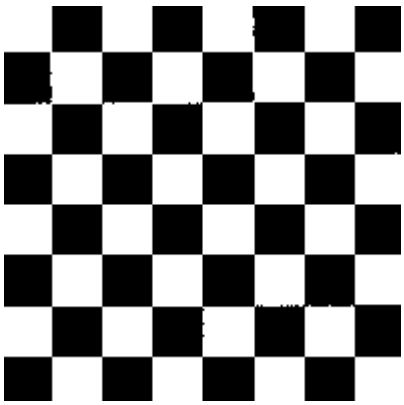


Figura 5.8. Imagine binarizată folosind metoda Otsu

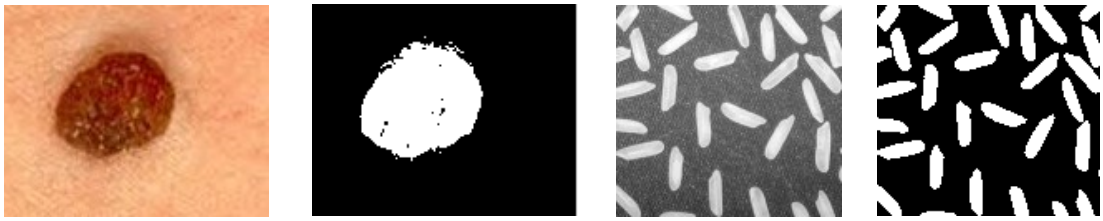
Se determină pragul de binarizare.

```
% determinarea pragului de segmentare
pragOtsu = graythresh(imagGray);
disp(['Valoarea pragului de binarizare = ',num2str(pragOtsu)])
```

```
Valoarea pragului de binarizare = 0.56471
```

Valoarea pragului de binarizare 0.56471 corespunde nivelului de gri 144.

Alte exemple de imagini binarizate folosind metoda Otsu:



Imagine inițială | Imagine binarizată | Imagine inițială | Imagine binarizată

Prag determinat cu Otsu = 130 | Prag determinat cu Otsu = 130

Figura 5.9. Exemple de imagini binarizate folosind metoda Otsu

5.4. Creșterea regiunilor

Introducere teoretică

Ideea de bază a metodei de segmentare bazată pe creșterea regiunilor este următoarea: se pornește, inițial, de la un pixel sau un grup de pixeli, denumiți *pixeli semințe* (*seeds*) și se examinează, pe rând, toți pixelii vecini ai acestora. Dacă unul dintre aceștia îndeplinește un anumit criteriu, dat de relația de vecinătate, atunci pixelul respectiv este adăugat grupului de pixeli deja format. Acest proces este continuat pentru toți pixelii noi adăugați, până când nu se mai respectă criteriul relației de vecinătate. Algoritmul depinde de masca *pixelilor semințe* inițiale, însă cel mai important este definirea criteriului de omogenitate pe baza căruia se adaugă noii vecini. În plus, prin natura sa, acest algoritm este unul recursiv ceea ce necesită resurse din partea sistemului de calcul.

Algoritm

Pas 1. Se pornește de la un punct de start, căruia i se cunosc coordonatele inițiale. Acesta este pixelul sămânță și este primul adăugat regiunii.

Pas 2. Se parcurge toată imaginea pixel cu pixel. Dacă pixelul curent nu face parte din regiune, este adiacent regiunii și nu diferă foarte mult de media regiunii, atunci:

- pixelul curent se adaugă regiunii
- se recalculează media regiunii

Pas 3. Se verifică *condiția de stop*. Dacă nu este îndeplinită atunci se reia de la *Pasul 2,* altfel algoritmul se încheie.

Condiție stop: Numărul de pixeli ai regiunii dintre două parcurgeri succesive ale imaginii nu se mai schimbă.

Observații:

1. Pentru a verifica similitudinea dintre pixelul curent (I_{pixel}) și media regiunii (m) folosim condiția:

$$\left|I_{pixel} - m\right| < T,$$ unde `T` este un prag impus.

2. Pentru a recalcula media unei regiuni formată din `N` pixeli de medie $\mathtt{m}_{\mathtt{prec}}$ căreia i se adaugă un nou pixel de intensitate `I`, se folosește formula:

$$m_{new} = \frac{m_{prec} \cdot N + I}{N + 1}$$

Aplicația 5.3. Fie imaginea cu tabla de șah din aplicațiile anteriore. Folosind algoritmul *region growing* care să pornească de la un *pixel sămânță* aflat într-un pătrățel al tablei de șah, să se găsească întreaga regiune a acelui pătrățel.

Pentru a seta *pixelul sămânță*, vom folosi funcția `ginput` din MATLAB, care permite selectarea cu mouse-ul al unui pixel din imagine și salvarea automată a coordonatelor sale (linie și coloană).

```
% incarcare imagine
imag = double(imread('sah.jpg'))/255;
figure(1)
imshow(imag)

% transformare imagine color originala in imagine grayscale
imag_gray = rgb2gray(imag);

% selectare punct samanta
[n_init, m_init] = ginput(1);
m_init = round(m_init); % m_init = linie pentru samanta
n_init = round(n_init); % n_init = coloana pentru samanta
```

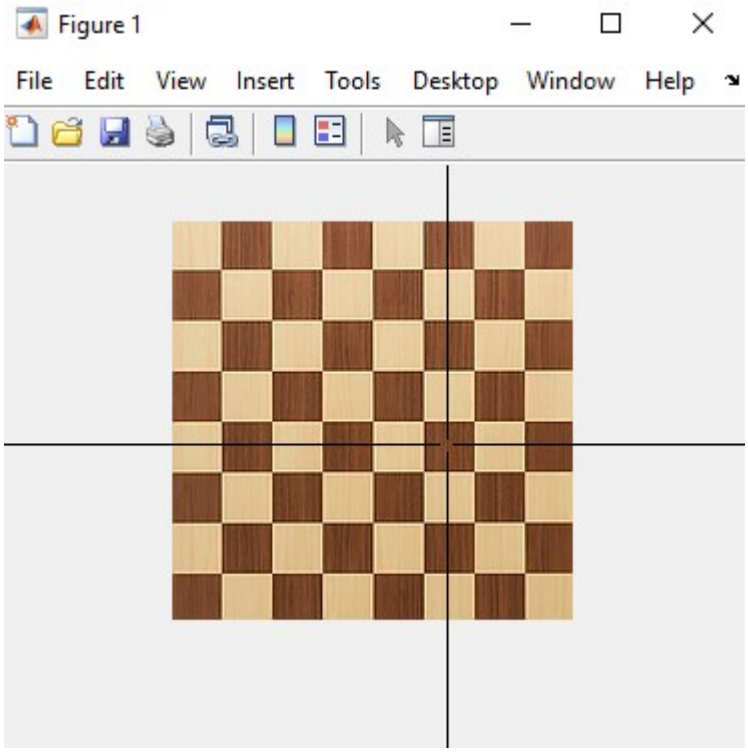


Figura 5.10. Selectarea *pixelului sămânță*

```
prag = 20/255;    % toleranta
medie = imag_gray(m_init,n_init); % media initiala
% marcare punct de start in imaginea originala
imag_seg = zeros(size(imag,1), size(imag,2));
imag_seg(m_init, n_init,1) = 1;
flag = 1;
N = 1;
while (flag == 1)
flag = 0;
 for  m = 2:size(imag,1)-1
    for n = 2:size(imag,2)-1
        I = imag_gray(m,n);
        if(imag_seg(m,n)==0)
            if((imag_seg(m-1,n)==1)||(imag_seg(m+1,n)==1)||(imag_seg(m,n-
1)==1)||(imag_seg(m,n+1)==1))
                if(abs(I-medie) < prag)
                    imag_seg(m,n) = 1;
                    medie = (medie*N + I)/(N+1);
                    N  = length(find(imag_seg==1));
                    flag = 1;
                end
            end
        end
    end
  end
end
figure(2)
imshow(imag_seg)
```

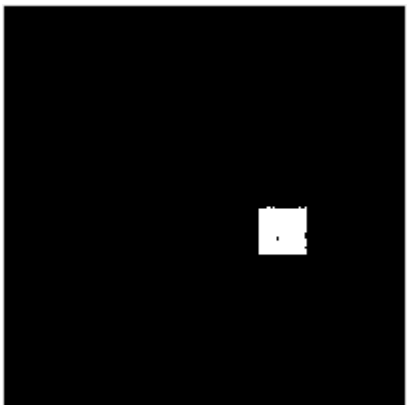


Figura 5.11. Regiunea găsită pornind de la *pixelul sămânță*

5.5. K-means clustering

Algoritmul este cel prezentat în *Capitolul 4.1*, doar că de data acesta îl vom aplica pentru segmentarea automată a imaginilor. Se va împărți automat imaginea de mai jos în două clase (*piele* și *aluniță*) folosind *k-means clustering*.

Figura 5.12. Imagine originală

Dimensiunea imaginii este de 112 x 140 x 3 (112 de linii, 140 de coloane, 3 straturi). In cazul de față, vectorii ce trebuie clasificați sunt pixelii care sunt 3-D (au 3 componente: roșu, verde, albastru). Pentru a putea folosi funcția `kmeans` va trebui ca fiecare pixel să fie salvat pe câte o linie a unei matrice ce va fi apoi folosită ca parametru de intrare pentru funcția `fcm`. Această reorganizare a pixelilor se poate face în MATLAB folosind funcția `reshape`. Folosind funcția `reshape` s-a obținut matricea `imagVector` cu 15680 de linii (112 x 140) și 3 coloane; pe fiecare linie se află câte un pixel din imagine.

```
imag = double(imread('alunita.jpg'))/255;
K = 2; % numarul de clase
[rows, cols, layers] = size(imag);
% pixelii din imagine vor fi salvati pe linii in matricea imagVector
imagVector = reshape(imag,[],layers);
% se realizeaza clustering cu k-means
% se repeta algoritmul de 5 ori si se ia cea mai buna clasificare
VectorKmeans = kmeans(imagVector, K, 'Replicates',5);
VectorKmeans(VectorKmeans==2) = 0;
VectorKmeans(VectorKmeans==1) = 1;
% in urma clasificarii se obtine un vector avand doar valorile 0 si 1
% vectorul obtinut se rearanjeaza ca imagine
ImagKmeans = reshape(VectorKmeans,rows,cols);
figure(1), imshow(ImagKmeans)
```

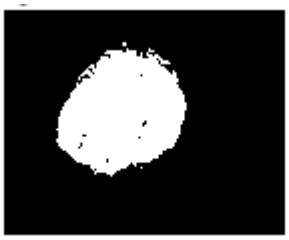


Figura 5.13. Imagine segmentată cu *k-means*

5.6. Fuzzy C-means clustering

Algoritmul este cel prezentat în *Capitolul 4.2*, doar că de data acesta îl vom aplica pentru segmentarea automata a imaginilor. Se va împărți automat imaginea de mai jos în două clase folosind *fuzzy C-means clustering*.

Sintaxă: `[centre,U,obj_fcn] = fcm(X, C)` unde:

- `X` = matricea vectorilor ce se dorește a fi clasificați, așezați pe linii
- `C` = numărul de clase în care se face împărțirea vectorilor
- `centre` = centrele celor 2 clase după terminarea clasificării
- `U` = gradele de apartenență la cele 2 clase după terminarea clasificării
- `obj_fcn` = funcțiile obiectiv după fiecare iterație

Obs: pentru sintaxa completă `>> help fcm`

Figura 5.14. Imagine originală

Dimensiunea imaginii este de 225 x 258 x 3 (225 de linii, 258 de coloane, 3 straturi). In cazul de față, vectorii ce trebuie clasificați sunt pixelii care sunt 3-D (au 3 componente: roșu, verde, albastru). Pentru a putea folosi funcția `fcm` va trebui ca fiecare pixel să fie salvat pe câte o linie a unei matrice ce va fi apoi folosită ca parametru de intrare pentru funcția `fcm`. Această reorganizare a pixelilor se poate face în MATLAB folosind funcția `reshape`.

```
C = 2; % C = numar de clase
cale = 'pasare.jpg';
imag = double(imread(cale))/255;
[rows, cols, layers] = size(imag);
imag_vector = reshape(imag,[],layers);
```

Folosind funcția `reshape` s-a obținut matricea `imag_vector` cu 58050 de linii (225 x 258) și 3 coloane; pe fiecare linie se află câte un pixel din imagine.

Matricea `imag_vector` a fost apoi folosită pentru clasificarea în `C = 2` clase.

```
[centre,U,obj_fcn] = fcm(imag_vector,C);
[val, pos] = max(U,[],1);
pos(pos==2) = 0;
pos(pos==1) = 1;
imag_seg = reshape(pos,rows,cols);
figure(1), image(imag), title('imagine originala')
figure(2), imshow(imag_seg), title('imagine segmentata cu Fuzzy C-means')
```

În urma algoritmului `fcm` s-au obținut:

- mediile celor 2 clase:

```
centre =
         0.9331    0.5008    0.4459
         0.3789    0.3817    0.2435
```

- variabila `U` având 2 linii și 58050 de coloane; pe colana `i,` linia 1, este gradul de apartenență al vectorului de pe linia `i` din `imag_vector` la clasa 1; pe colana `i,` linia 2, este gradul de apartenență al vectorului de pe linia `i` din `imag_vector` la clasa 2; se va alege maximul dintre cele 2.
- variabila `obj_fcn` conținând valorile funcțiilor obiectiv. Algoritmul a rulat 22 de iterații, în consecință sunt 22 de valori, ultima fiind:

```
Iteration count = 22, obj. fcn = 1776.370844
```

În urma clasificării folosind algoritmul *fuzzy C-means clustering*, imaginea clasificată arată astfel:

Figura 5.15. Imagine clasificată cu `fcm`

Anexe

Baze de date

Anexa 1. Baza de date *Wine*

Baza de date *Wine* conține un set de 178 de vinuri, pentru fiecare vin existând un set de 13 parametri fzico-chimici. Acești parametrii sunt:

1. Alcool
2. Acidul malic
3. Reziduu
4. Alcalinitatea rezidurilor
5. Magneziu
6. Fenoli
7. Flavanoide
8. Fenoli neflavanoizi
9. Proantocianine
10. Intensitatea culorii
11. Culoare/Nuanță
12. OD280 / OD315 de vinuri diluate
13. Proline

Pe baza acestor parametrii, vinurile au fost împărțite în 3 clase:

- Clasa 1
- Clasa 2
- Clasa 3

Baza de date *Wine* este disponibilă în biblioteca MATLAB-ului și poate fi încărcată astfel:

```
>> [wineInputs, targets] = wine_dataset;
```

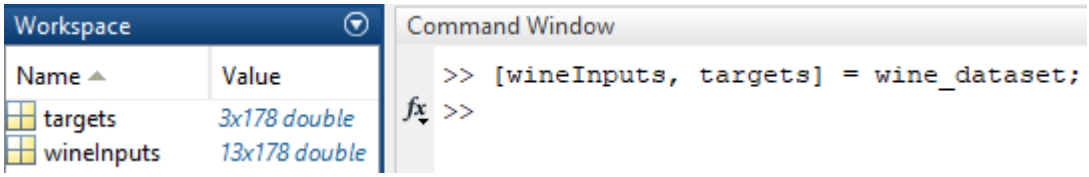


Figura A.1. Baza de date *Wine*

- `wineInputs` este o matrice cu 13 linii și 178 de coloane; în această matrice, pe fiecare coloană se regăsesc parametrii fizico-chimici ai unui vin.
- `targets` este o matrice cu 3 linii și 178 de coloane conținând împărțirea în clase a vectorilor din matricea `wineInputs`; dacă vectorul de pe coloana `i` din matricea `wineInputs` aparține *clasei 1*, atunci coloana `i` din `targets` va fi $[1, 0, 0]^T$; dacă aparține *clasei 2*, atunci coloana `i` din `targets` va fi $[0, 1, 0]^T$ iar dacă aparține *clasei 3*, atunci coloana `i` din `targets` va fi $[0, 0, 1]^T$.

Distribuția celor 178 de vectori în cele 3 clase este următoarea.

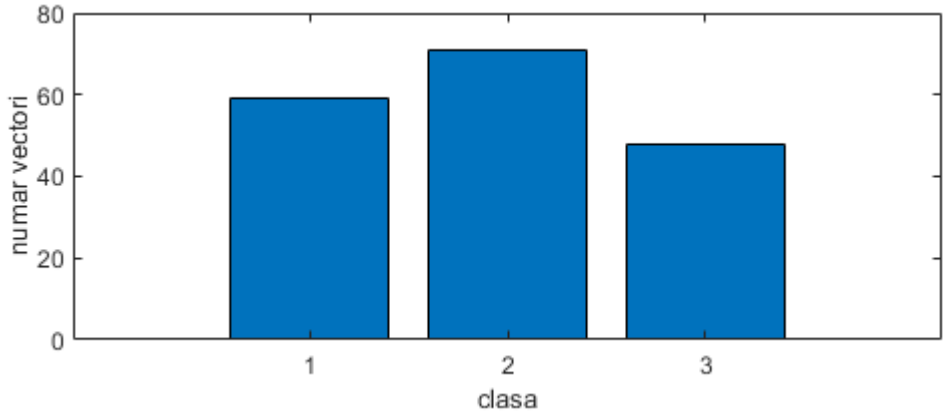


Figura A.2. Distribuția vectorilor în clase

Așa arată primul vector din baza de date, care aparține *clasei 1*.

```
>> wineInputs(:,1)
ans =
   1.0e+03 *
    0.0142
    0.0017
    0.0024
    0.0156
    0.1270
    0.0028
    0.0031
    0.0003
    0.0023
    0.0056
    0.0010
    0.0039
    1.0650

>> targets(:,1)
ans =
     1
     0
     0
```

Anexa 2. Baza de date *MNIST*

Baza de date MNIST (en . *Modified National Institute of Standards and Technology*) conține cifre scrise de mână și este folosită în mod obișnuit pentru instruirea și testarea diferitelor sisteme de procesare a imaginilor.

Baza de date conține 60.000 de imagini pentru antrenare și 10.000 de imagini pentru test. Dimensiunea unei imagini este de 28 x 28 pixeli.

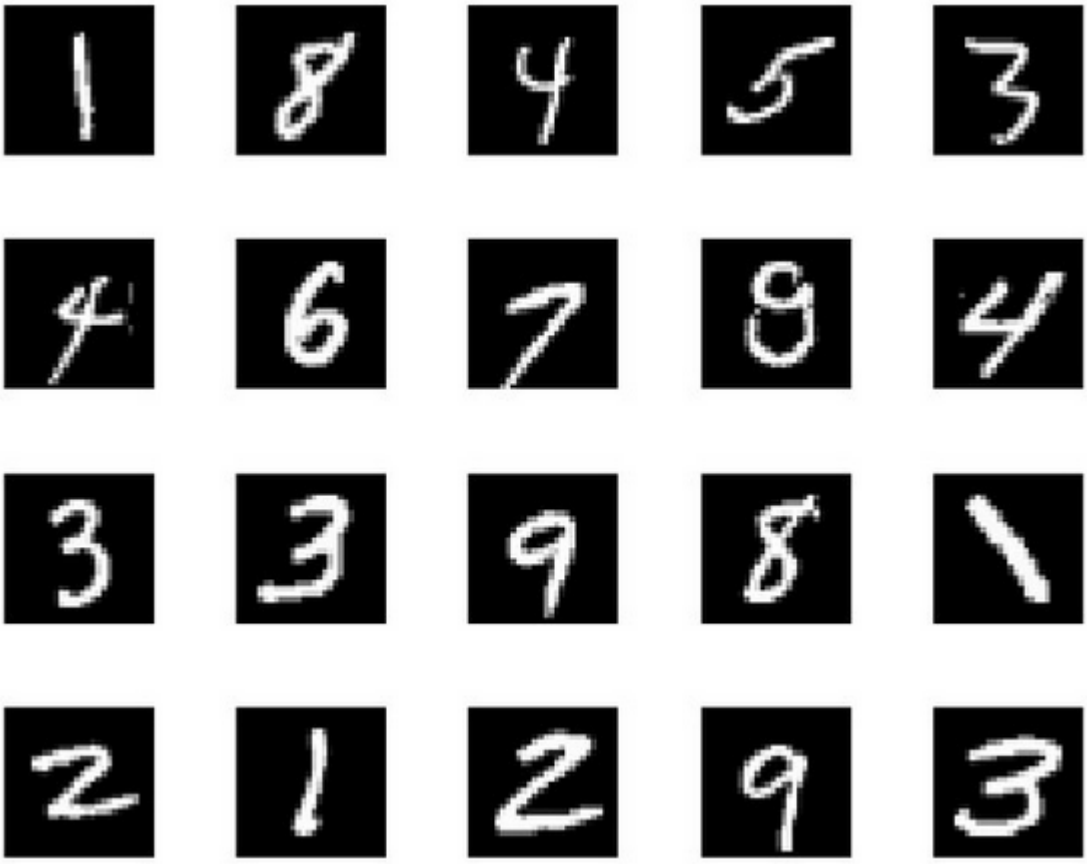


Figura A.3. Exemplu de imagini cu cifre din baza de date MNIST

Baza de date MNIST oficială este disponibilă online, accesul fiind gratuit [Yann LeCun]. În această lucrare, s-a folosit baza de date în format CSV, accesarea imaginilor fiind mult mai simplă:

- prima coloană din fișierul CSV conține cifra reprezentată în imagine;
- următoarele 784 de coloane din fișierul CSV conțin pixelii din imagine, în format `uint8` (între 0 și 255)

În continuare este un exemplu de încărcare a bazei de date MNIST, citire și afișare a unei cifre în MATLAB.

```
% incarcare baza de date
MNISTdatabase = xlsread('mnist_train.xlsx');
% se alege a cata imagine din baza de date sa fie citita
Nr_imag = 5;
% se cunoaste ca dimensiunea imaginilor este 28 x 28
dim_imag = 28;
% se citeste imaginea, care initial este un vector
imag_vector  = MNISTdatabase(Nr_imag ,2:end)/255;
% se transforme vectorul in imagine
imag_matrice = reshape(imag_vector,dim_imag,dim_imag)';
% se citeste clasa careia apartine imaginea
imag_clasa   = MNISTdatabase(Nr_imag , 1);
% se afiseaza imaginea
figure(1)
imshow(imag_matrice)
disp(['cifra afisata este cifra = ', num2str(imag_clasa)])
```

```
cifra afisata este cifra = 9
```

Bibliografie

[Bishop2006], M. Bishop, "*Pattern Recognition and Machine Learning"*, Springer, New York, 2006, ISBN-10: 0-387-31073-8, ISBN-13: 978-0387-31073-2

[Grossberg1969], S. Grossberg, *"Embedding fields: A theory of learning with physiological implications"*, Journal of Mathematical Psychology, 6, 209–239, 1969

[Hebb1949], Hebb, D. O. "*The organization of behavior; a neuropsychological theory. Wiley*", 1949

[Kohonen1982], Kohonen, Teuvo "*Self-Organized Formation of Topologically Correct Feature Maps*". Biological Cybernetics. 43 (1): 59–69. doi:10.1007/bf00337288, 1982

[Marquardt1963], Marquardt, D., "*An Algorithm for Least-Squares Estimation of Nonlinear Parameters*", *SIAM Journal on Applied Mathematics,* Vol. 11, No. 2, pp. 431–441, June 1963

[Mathwork 1], Mathwork documentation, https://uk.mathworks.com/help/deeplearning/ug/divide-data-for-optimal-neural-network-training.html

[Mathwork 2], Mathwork documentation, https://www.mathworks.com/help/nnet/ug/train-and-apply-multilayer-neural-networks.html

[Michie 1994], Donald Michie, David J. Spiegelhalter, C. C. Taylor, *Machine Learning, Neural and Statistical Classification*, Published 1994 DOI:10.1080/00401706.1995.10484383

[Mihu Z. Ioan], *Rețele neuronale*, suport curs

[Neagoe1998], V. Neagoe, O. Stănăşilă, "*Recunoaşterea formelor şi rețele neurale-algoritmi fundamentali*", Ed. Matrix Rom, Bucureşti, 1998.

[NeghinaC2012], Teză de doctorat, Autor: Ing. Elena – Cătălina GAIȚĂ (NEGHINĂ), Conducător de doctorat: Prof. dr. ing. Victor Emil NEAGOE *"Contribuții la dezvoltarea unor metode inteligente de inspirație naturală pentru recunoaşterea formelor"* UPB, 2012

[NeghinaC2016], Cătălina NEGHINĂ, Alina SULTANA, Mihai NEGHINĂ, *"MATLAB. Un prim pas spre cercetare"*, Editura Universității "Lucian Blaga", ISBN: ISBN 978-606-12-1213-2, Sibiu, 2016

[Rosenblatt1958], Rosenblatt, F, "*The perceptron: A probabilistic model for information storage and organization in the brain*", *Psychological Review, 65*(6), 386–408, 1958

[Rumelhart1986], Rumelhart, D. E., Hinton, G. E., & McClelland, J. L. "*A general framework for parallel distributed processing*". In D. E. Rumelhart, J. L. McClelland, & The PDP Research Group (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition. Volume 1: Foundations* (pp. 45–76). Cambridge, MA: MIT Press, 1986

[Widrow1960], B. Widrow et al. "*Adaptive Adaline neuron using chemical memistors*", Number Technical Report 1553-2. Stanford Electron. Labs., Stanford, CA, October 1960

[Widrow1962], Widrow B., Hoff M.E, "*Associative Storage and Retrieval of Digital Information in Networks of Adaptive Neurons*", 1962

[Yann LeCun], *The MNIST database of handwritten digits*, http://yann.lecun.com/exdb/mnist/

[Zurada1992], Jacek M. ZURADA, "*Introduction to Artificial Neural Systems"*, By West Publishing Company, 1992