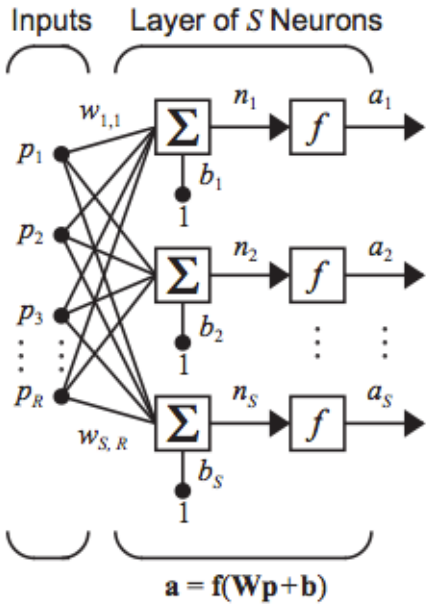
Laborator 9

Rețelele cu un singur perceptron nu pot rezolva probleme de clasificare liniar neseparabile. Spre exemplu, o rețea cu un singur perceptron nu poate rezolva problema învățării funcției booleene XOR (vedeți laboratorul 5). În cele ce urmează vom prezenta rețele de perceptroni cu un strat sau mai multe straturi care pot rezolva asemenea probleme de clasificare liniar neseparabile precum și probleme de regresie neliniară.

Rețele feedforward de perceptroni cu un singur strat

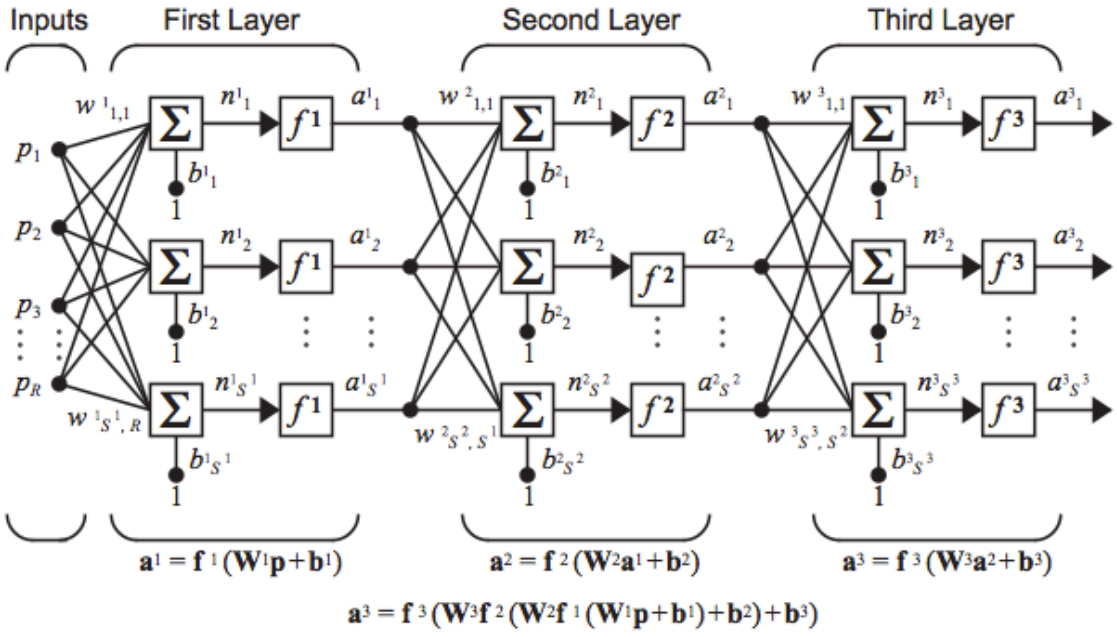


Rețelele neuronale feedforward sunt rețele de perceptroni grupați pe straturi, în care propagarea informației se realizează numai dinspre intrare spre ieșire (de la stânga la dreapta). În figura de alături este reprezentată grafic o *reţea feedforward cu un singur strat* (layer) de S perceptroni (neuroni) cu R intrări. Fiecare perceptron *i* are asociate ponderile $w_{i,1}$, $w_{i,2}$,…, $w_{i,R}$, bias-ul b_i, funcția de transfer f (spre exemplu f poate fi hardlim, logsig, tansig, purelin, etc.) și ieșirea a_i. Toate aceste sunt reținute în matricea de ponderi **W**, vectorul de bias-uri **b** și vectorul rezultant **a**.

Rețele feedforward de perceptroni multistrat

În general, reţelele feedforward sunt multistrat, conținând mai multe straturi de perceptroni. Perceptronii de pe primul strat sunt singurii care primesc semnale din exterior (semnalele de intrare). Perceptronii dintr-un strat sunt direct conectați cu perceptronii din stratul următor; nu există conexiuni între perceptronii aceluiași strat. Ultimul strat se numeşte ***strat de ieşire*** (*output layer*) iar celelalte straturi se numesc ***straturi ascunse*** (*hidden layers*). Figura de pe pagina următoare ilustrează o rețea cu 3 straturi. Fiecărui strat îi corespunde o matrice de ponderi **W**, un vector bias **b**, şi un vector rezultant **a**. Primului strat îi corespunde matricea de ponderi provenită de la intrări, notată în figură cu $\mathbf{W^1}$ (în Matlab este data de câmpul IW - *Input Weights*) iar apoi fiecărui strat *i* se asociază o matrice de ponderi $\mathbf{W^i}$ provenită din ieşirile stratului anterior. În Matlab, ponderile care conectează straturile se numesc *Layer Weights* (LW). Pentru a distinge matricele de ponderi pentru

fiecare strat în parte se notează pentru fiecare variabilă în parte ca superscript indicele nivelului din care face parte. Indicii arată care este stratul destinaţie (primul indice) şi stratul sursă (al doilea indice). Astfel, matricea de ponderi $LW^{2,1}$ conectează stratul 1 cu stratul 2.

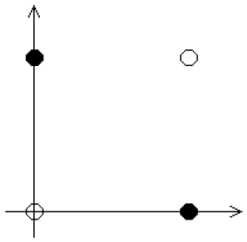


În Matlab definim o rețea feedforward folosind funcția ***newff*** astfel:

net = newff (PR, [k_1,...,k_m], {f_1,..., f_m}, BTF, BLF, PF), unde
PR – [min_1 max_1;...; min_n max_n], unde min_i și max_i reprezintă valoarea minimă, respectiv valoarea maximă pentru semnalul de intrare pentru perceptronul i.
k_i - numărul de perceptroni aflaţi pe stratul i;
f_i - funcţia de transfer a stratului i. Implicit este 'tansig' pentru nivelul ascuns şi 'purelin' pentru nivelul de ieşire (de obicei se folosesc funcții diferențiabile astfel încât rețeaua să se poată antrena);
BTF – funcţia de antrenare a reţelei folosind algoritmul backpropagation, implicit = 'trainlm' (metoda Levenberg-Marquart);
BLF – funcţia de învăţare a ponderilor/bias-ului, implicit = 'learngdm' (regula gradientului descendent cu moment);
PF - funcţia de performanţă, implicit = 'mse' (media pătratelor erorilor).

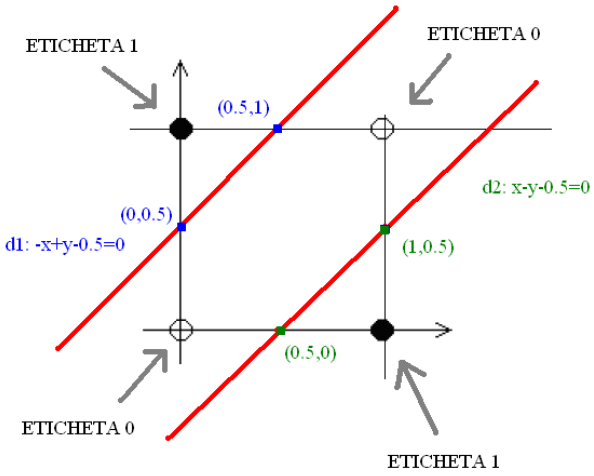
Rezolvarea problemei XOR folosind o rețea multistrat

Figura de mai jos conține reprezentarea grafică a funcției XOR, în care punctele (0,0), (1,1) au eticheta 0 (culoare albă), iar punctele (1,0), (0,1) au eticheta 1 (culoare neagră).



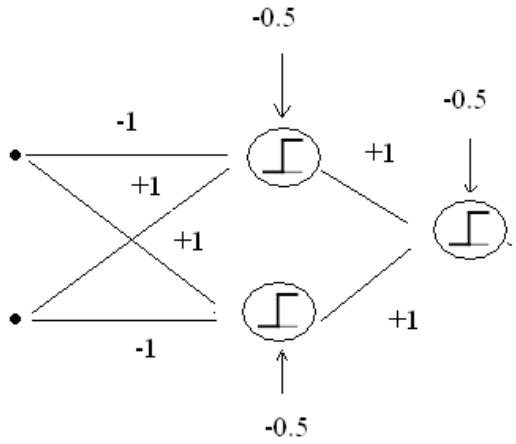
XOR

Pentru problema XOR, limitarea perceptronului poate fi depăşită prin crearea unei reţele de perceptroni cu 2 straturi cu următoarea structură:
- primul strat conține doi perceptroni, fiecare perceptron implementează ecuația unei drepte (culoare roșie în figura de mai jos) care separă un punct negru de celelalte două puncte albe. Fiecare perceptron stabilește dacă punctul (x,y) se află deasupra sau dedesubtul celor două drepte.
- al doilea strat conține un perceptron care stabilește dacă punctul (x,y) se află între cele două drepte, ieşirea reţelei fiind astfel 0 sau 1, în caz contrar.
Există o infinitate de drepte roșii care separă un punct negru de celelalte două puncte albe. Particularizând dreptele de culoare roşie ca în figura următoare:



obţinem dreptele de separare d_1: -x + y -0.5 = 0 şi d_2: x -y - 0.5 = 0.
Punctele (x,y) deasupra dreptei d_1 au valoarea ieşirii perceptronului de pe primul strat corespunzător 1, iar cele de dedesubtul dreptei d_1 au valoarea 0. Punctele deasupra dreptei d_2 au valoarea ieşirii perceptronului de pe primul strat corespunzător 0, iar cele de dedesubtul dreptei d_2 au valoarea 1. În

final, punctul (x,y) este etichetat cu 0 dacă se află între cele două drepte și cu 1 altfel. Reţeaua pe care o putem construi poate fi reprezentată grafic astfel:

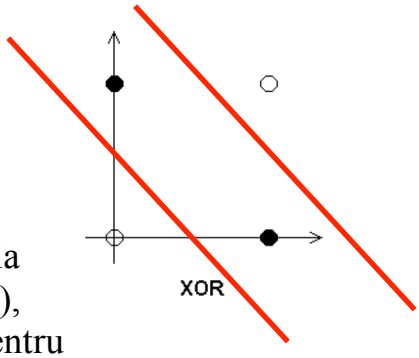


Codul Matlab pentru construcția acestei rețele este următorul:

```
X = [0 1 0 1; 0 0 1 1]; %input-urile
t = [0 1 1 0]; % target-urile
net=newff(minmax(X),[2 1],{'hardlim','hardlim'});%definim reteaua
net.IW{1,1} = [-1 1; 1 -1]; %matricea de ponderi de pe primul strat
net.LW{2,1}=[1 1]; % matricea de ponderi de pe al doilea strat
net.b{1} = [-0.5;- 0.5];
net.b{2} = -0.5; %bias-urile
a=sim(net,X)
isequal(a,t)
```

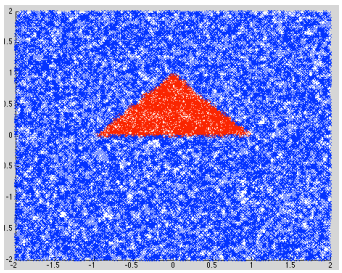
Exerciții

1. Rezolvați problema XOR folosind o rețea care implementează soluția din figura alăturată (punctele cuprinse între cele două drepte primesc valorea 0, cele din afara dreptelor priemsc valoarea 1).

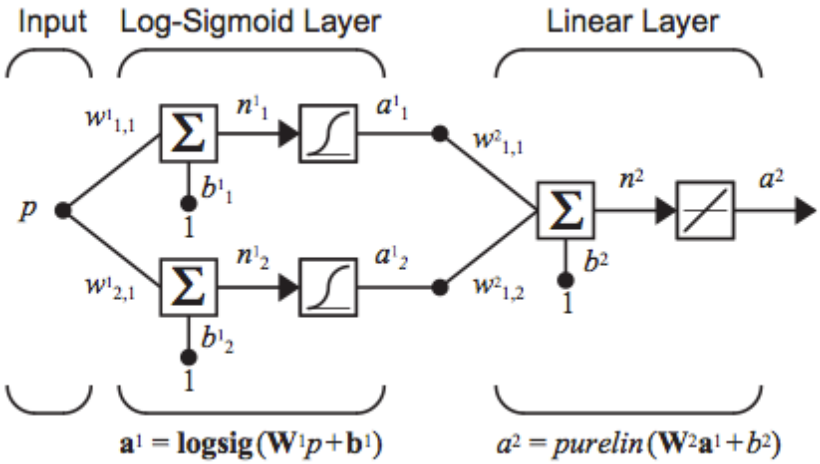


2. a) Construiţi o reţea care să implementeze funcţia indicator a triunghiului ABC de vârfuri A(-1,0), B(0,1), C(1,0). (funcţia indicator ia valoarea 1 pentru punctele din interiorul triunghiului şi de pe frontieră şi 0 în rest).
Indicaţie: reţeaua va avea pe primul strat 3 perceptroni, câte unul pentru dreptele AB, AC şi BC şi pe ultimul strat un singur perceptron.

b) Construiţi matricea **puncteTest** de dimensiune 2 × 20000 care să conţină 20000 de puncte generate uniform în pătratul [-2 2] × [-2 2]. Simulaţi reţeaua de la punctul a pe datele **puncteTest** şi plotaţi punctele colorându-le diferit în funcţie de răspunsul reţelei. (culoarea roşie pentru punctele din triunghi şi culoarea albastră pentru punctele din afara triunghiului).



În exemplele anterioare, am folosit rețelele feedforward multistrat la probleme neliniare de clasificare binară (obținem clasele 0 sau 1 ca output-uri). Putem folosi rețelele și pentru probleme de regresie. Figura de mai jos conține reprezentarea grafică a unei rețele cu 2 perceptroni pe stratul ascuns cu funcția de transfer logsig (logsig(x) = $1/(1+e^{-x})$) și un perceptron pe stratul de ieșire cu funcția de transfer liniară – purelin (purelin(x) = x).



Programul demonstrativ **nnd11nf** exemplifică această rețea. Studiați cum se modifică ieșirea a^2 a rețelei în funcție de intrarea p modificând cei 7 parametri ai rețelei.

Exemplul de mai sus ilustrează flexibilitatea unei rețele cu doar două straturi în reprezentarea unor funcții cu diverse grafice. De altfel, se demonstrează că rețelele mutistrat cu un număr mare de perceptroni pe straturile ascunse pot aproxima aproape orice funcție. Numărul de straturi și numărul de perceptroni se determină experimental.

Antrenarea rețelelor feedforward multistrat folosind backpropagation

Rețelele feedforward multistrat se pot antrena folosind algoritmul backpropagation (cursul 10) astfel încât ieșirea **y** a rețelei să fie apropiată de eticheta **d** dorită pentru intrarea **x**. În algoritmul de backpropagation, pentru mulțimea de antrenare S = $\{(\mathbf{x}^1,\mathbf{d}^1), (\mathbf{x}^2,\mathbf{d}^2), \ldots, (\mathbf{x}^m,\mathbf{d}^m)\}$, index-ul de performanță (net.performFcn) prin care se măsoară diferența dintre ieșirea y^i și eticheta d^i este funcția medie a pătratelor erorii (mse – mean square error):

$$E(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{d}^i - \boldsymbol{y}^i)^2,$$

unde **w** este vectorul de ponderi conținând toți parametri rețelei. Pentru exemplul rețelei anterioare **w** are 7 parametri, w = ($w^1_{1,1}$, $w^1_{2,1}$, b^1_1, b^1_2, $w^2_{1,1}$, $w^2_{2,1}$, b^2). Pentru ușurința calculului analitic (la derivare) se înlocuiește în formula de mai sus m cu 2, funcția rezultantă diferind doar cu o constantă de cea inițială.
Antrenarea rețelei înseamnă actualizarea vectorului de ponderi **w** în vederea micșorării erorii E(**w**). În algoritmul de backpropagation (propagarea înapoi a erorii), actualizarea vectorului de ponderi **w** se realizează prin metoda gradientul descendent, prin calcularea derivatelor parțiale ale funcției E în raport cu toți parametri din **w**. Ecuațiile pentru fiecare strat se găsesc în curs, ele sunt implementate de Matlab.

Programul demonstrativ **nnd11bc** arată cum se actualizează vectorul de ponderi w în cazul rețelei anterioare pornind de la w^0 = (-0.27, -0.41, -0.48, -0.13, 0.09, -0.17, 0.48) iar funcția care încearcă să fie aproximată, este f(x) = 1+sin(x*π/4).

Antrenarea unei rețele feedforward pentru problema XOR

Rețelele care rezolvă problema XOR de mai înainte au fost create manual. Ele au funcțiile de transfer 'hardlim' care sunt funcții nediferențiabile. Algoritmul backpropagation are la bază idea de actualizare a ponderilor pe baza gradienului. De aceea, rețelele care sunt antrenate cu backpropagation necesită funcții de transfer diferențiabile. Înlocuim pentru exemplul anterior cu XOR funcția hardlim cu funcția logsig (ea aproximează asimptotic funcția hardlim). Codul Matlab care antrenează rețeaua de perceptroni corespunzătoare este următorul:

```
X = [0 1 0 1; 0 0 1 1]; %input-urile
t = [0 1 1 0]; % target-urile
net=newff(minmax(X),[2 1],{'logsig','logsig'});%creare retea
view(net);%vizualizare retea
[net, info] = train(net,X,t);
a=sim(net,X)
```

Este posibil ca soluția obținută să nu fie cea dorită. Acest lucru se poate întâmpla dacă algoritmul de backpropagation (varianta implicită este cea de antrenare pe baza gradientului cu moment – 'trainlm', algoritmul Levenberg-Marqardt; acest algoritm împreună cu alți algoritmi vor fi abordați laboratorul următor) converge la un minim local. În acest caz putem observa că eroarea pătratică medie pentru soluția curentă este mare (valoarea erorii este dată de valoarea *info.perf(end)*). Putem evita convergența la un minim local cu performanță scăzută prin antrenarea rețelei de mai multe ori, de fiecare data ea fiind inițializată cu un alt set de ponderi. Rețeaua cea mai bine antrenată este cea pentru care valoarea *info.perf(end)* este cea mai mică. Evident, trebuie evitată supraînvățarea (overfitting-ul), pe cazul general păstrând o mulțime de validare. După antrenare, întrucât problema este de clasificare binară, putem înlocui funcția de transfer de pe ultimul strat cu funcția hardlim.

Exercițiu
Antrenați rețeaua corespunzătoare exercițiului 2 anterior care implementează funcția indicator a triunghiului ABC.

Programul demonstrativ **nnd11fa** ilustrează capacitatea unei rețele cu un strat ascuns și cu număr variabil de perceptroni pe acest strat de a aproxima o funcție reală sinusoidală. Rulați acest program demonstrativ și observați cum pentru a aproxima o funcție cu număr mare de puncte de inflexiune (index de dificultate mai mare) avem nevoie de un număr mai mare de perceptroni.

Rețelele multistrat sunt antrenate pe baza unor mulțimi de antrenare finite. Mulțimea de antrenare poate fi reprezentativă pentru mai multe clase de funcții. Este important ca rețeaua să generalizeze pe baza exemplelor de

antrenare. Programul demonstrativ **nnd11gn** ilustrează acest lucru. Putem antrena o rețea cu 2 sau 9 perceptroni pentru a aproxima o funcție dată prin mulțimea de antrenare. Pentru funcții simple (difficulty index = 1) numai rețele cu număr mic de perceptroni au capacitate de generalizare. Numărul de parametri ai rețelei cu 2 perceptroni este 7, în timp ce numărul de parametri ai rețelei cu 9 perceptroni este 28. Pentru a putea generaliza, o rețea trebuie să aibă mai puțin parametri decât numărul exemplelor din mulțimea de antrenare (ea conține 11 puncte pentru funcția exemplificată în program cu indexul de dificultate = 1).