Algoritmi metaeuristici.

Lab 6:

Optimizare multicriterială
Probleme de clasificare. Utilizare Weka pentru probleme de clasificare
Probleme de aproximare și predicție. Implementarea în Scilab a rețelelor neuronale feedforward cu un nivel ascuns și a rețelelor RBF

---

1. Optimizare multicriterială.

Optimizarea multicriterială are ca scop optimizarea simultană a mai multor criterii. Funcția obiectiv este de forma $F:R^n->R^r$, iar componentele sunt $F=(f_1,f_2,\ldots,f_r)$.

Criteriile de optimizat sunt de regulă conflictuale, astfel că nu există o soluție unică a problemei. In aceste condiții se caută soluții în sens Pareto, care se caracterizează prin faptul că nu există alte configurații care să fie mai bune în raport cu toate criteriile de optimizat (orice îmbunătățire în raport cu unul dintre criterii conduce la o înrăutățire în raport cu un alt criteriu). Astfel de soluții se numesc nedominate (în sensul că nu există alte soluții care să le domine în raport cu fiecare dintre criteriile de optimizat).

Există mai multe abordări ale unei astfel de probleme. Principalele categorii de metode sunt:

- *Metode bazate pe tehnica agregării:* se transformă problema inițială de optimizare multicriterială într-una de optimizare uni-criterială prin combinarea criteriilor. Noua funcție de optimizat este $f(x)=w_1f_1(x)+w_2f_2(x)+\ldots+w_rf_r(x)$ unde $w_1,w_2,\ldots,w_r$ sunt ponderi asociate criteriilor. Pentru fiecare set de ponderi se poate obține o altă soluție.
- *Aproximarea directă a mulțimii Pareto optimale:* se utilizează o populație de elemente care vor aproxima mulțimea Pareto optimală (multimea tuturor soluțiilor nedominate).

Exemple de funcții test utilizate pentru a analiza performanțele algoritmilor de optimizare multicriterială sunt disponibile la: http://en.wikipedia.org/wiki/Test_functions_for_optimization

Aplicație 1. Se consideră funcția $F:[0,4]->RxR$, $F(x)=((x-1)^2,(x-2)^2)$. Să se aproximeze mulțimea Pareto optimală și frontul Pareto corespunzător (frontul Pareto reprezintă mulțimea valorilor funcțiilor obiectiv corespunzătoare elementelor din mulțimea Pareto optimală).

Varianta 1. Se aplică tehnica agregării

a) se construiește funcția obiectiv:

```
function y=fw(x)
    w=0.1;
    y1=(x-1)*(x-1);
    y2=(x-2)*(x-2);
    y=w*y1+(1-w)*y2;
endfunction
```

b) se aplica succesiv o strategie evolutivă (de exemplu cea descrisă în SE.sci) pentru optimizarea functiei obiectiv pentru următoarele valori ale ponderii w: (0.1,0.2,0.3,…,0.9) și se stochează valoarea într-un vector x

c) se vizualizează valorile funcțiilor obiectiv pentru valorile stocate la pasul anterior (va reprezenta o aproximare a frontului Pareto):

```
function pareto(x)
    f1=(x-1).^2;
    f2=(x-2).^2;
    plot(f1,f2,'*');
endfunction
```

Varianta 2. Se folosește implementarea algoritmului NSGA-II (funcția optim_nsga2 în Scilab) sau a algoritmului MOGA (funcția optim_moga în Scilab).

Indicație: vezi exNSGA.sci

2. Probleme de clasificare

Problemele de clasificare necesită gruparea unor obiecte descrise prin caracteristicile lor (vectori de trăsături sau de atribute) în clase predefinite. Clasificatorii se construiesc pornind de la exemple de clasificare corectă printr-un proces de învățare (numită învățare supervizată spre deosebire de cazul învățării nesupervizate unde clasele nu sunt predefinite).

Problema clasificării intervine în numeroase domenii și numeroase probleme practice pot fi formulate ca probleme de clasificare. Exemple de probleme de clasificare:

- *Recunoașterea caracterelor*:
  - Datele de intrare conțin caracteristici ale caracterelor (obținute în urma preprocesării reprezentării grafice a caracterului)
  - Clasele corespund caracterelor care urmează a fi identificate (litere, cifre, alte simboluri)

- *Identificarea mesajelor de tip spam*:
  - Datele de intrare sunt informații obținute prin preprocesarea mesajelor (informațiile obținute prin preprocesare se extrag atât din antetul mesajului cât și din conținutul acestuia și se referă de regulă la frecvența de apariție a unor cuvinte sau construcții sintactice stocate anterior într-o colecție de elemente considerate ca fiind potențial suspecte)
  - Clasele sunt reprezentate de cele două categorii: mesaje legitime, respectiv mesaje frauduloase (de tip spam)

- *Diagnoza medicală*:
  - Datele de intrare sunt simptome, rezultate ale investigațiilor, caracteristici ale pacientilor etc.
  - Clasele sunt corelate cu diagnosticul (în cazul binar poate fi vorba de absența sau prezența unei boli)

2.1. Tehnici de clasificare

Tehnicile de clasificare au ca scop principal extragerea unor modele de clasificare (clasificatoare) pornind de la un set de date care ilustrează legătura dintre obiectele de clasificat și clasele corespunzătoare. Modelele de clasificare pot fi binare (în cazul clasificării în două clase) sau multiple (în cazul clasificării în mai multe clase).

In funcție de modul in care se extrage modelul de clasificare şi de structura finală, există mai multe categorii de clasificatoare:

- *Arbori de decizie.* Nodurile interne ale arborelui conțin condiții referitoare la valorile atributelor, iar nodurile frunză conțin etichete ale claselor; clasificarea unui obiect presupune parcurgerea unei căi în arbore pornind de la rădăcină până la un nod frunză – nodul frunză la care se ajunge indică clasa.
- *Reguli de decizie.* Sunt construcții de tip IF-THEN care conțin în antecedent expresii logice în care intervin atributele, iar in partea de concluzie eticheta unei clase; clasificarea presupune parcurgerea setului de reguli şi identificarea regulii care se potrivește cu datele.
- *Clasificatoare bazate pe principiul celui mai apropiat vecin (nearest neighbor).* Clasificatorul constă dintr-un set de exemple de clasificare; clasificarea constă în determinarea pentru un obiect de clasificat a celor mai apropiate exemple din set şi selecția clasei majoritare.
- *Rețele Bayesiene.* Sunt modele probabiliste descrise prin grafuri care surprind relațiile dintre clase şi atribute; nodurile din graf sunt asociate fie atributelor (interpretate ca variabile observabile) fie unor variabile latente, iar arcele sunt asociate corelațiilor dintre entități. Clasificarea se bazează pe estimarea probabilității fiecărei clase, condiționată de datele de intrare.
- *Rețele neuronale.* Sunt structuri de tip graf care modelează dependența neliniară dintre atributele de intrare şi clase. Antrenarea presupune determinarea paramentrilor modelului neliniar pornind de la exemple de antrenare, iar clasificarea se bazează pe determinarea semnalului de ieşire şi stabilirea clasei pe baza acestuia.
- *Clasificatoare bazate pe vectori support (Support Vector Machines).* Sunt modele de clasificare bazate pe identificarea unor hiperplane care asigură separarea claselor. Antrenarea presupune estimarea parametrilor hiperplanelor (şi eventual a unor funcții nucleu utilizate în transformarea problemelor neliniar separabile în probleme linear separabile) iar clasificarea constă în calculul valorii asociate funcției (funcțiilor) separatoare.

2.2. Etapele proiectării unui clasificator

a) *Pregătirea datelor*. Această etapă presupune pre-procesarea datelor (de exemplu eliminarea erorilor, tratarea valorilor absente, normalizarea etc.) şi distribuirea acestora în trei seturi principale:

   - *Date de antrenare*: date utilizate în procesul de antrenare pentru determinarea parametrilor clasificatorului (în cazul rețelelor neuronale se determina ponderile conexiunilor dintre neuroni)
   - *Date de validare*: date utilizate pentru a analiza comportamentul clasificatorului pe parcursul algoritmului de învățare; performanța obținută pe setul de validare în timpul procesului de învățare este utilizată pentru a decide dacă învățarea trebuie continuată sau nu.
   - *Date de testare*: sunt utilizate pentru a analiza performanțele unui clasificator antrenat.

b) *Antrenarea clasificatorului.* Presupune extragerea modelului de clasificare din date. Modul de antrenare depinde de tipul de clasificator. In cazul arborilor de decizie, prin antrenare se identifică ordinea în care se selectează atributele datelor pentru a fi asignate nodurilor şi a

realiza ramificarea. În cazul rețelelor neuronale antrenarea permite determinarea parametrilor asociați conexiunilor dintre neuroni (ponderile conexiunilor).

c) *Evaluarea clasificatorului.* Pentru evaluarea calității antrenării se poate folosi tehnica *validării încrucișate* (cross-validation) care constă în divizarea setului inițial de date într-un număr de S subseturi și repetarea de S ori a procesului de antrenare de fiecare dată utilizând alt subset de antrenare. La fiecare repetare antrenarea se bazează pe S-1 subseturi iar validarea se face utilizând subsetul care nu a fost utilizat la antrenare.

Calitatea unui clasificator din perspectiva identificării corecte a unei clase se măsoară folosind informațiile din *matricea de confuzie* care conține:

- Numărul de date clasificate corect ca aparținând clasei de interes: True positive cases (TP)
- Numărul de date clasificate corect ca neaparținând clasei de interes: True negative cases (TN)
- Numărul de date clasificate incorect ca aparținând clasei de interes: False positive cases (FP)
- Numarul de date clasificate incorect ca neaparținând clasei de interes: False negative cases (FN)

In cazul clasificării în două clase P(ozitiv) și N(egativ) matricea de confuzie are structura:

	Clasa prezisă: P	Clasa prezisă: N
Clasa reală: P	TP	FN
Clasa reală: N	FP	TN

Pe baza acestor valori se pot calcula o serie de alte măsuri: acurateţe, rata de eroare, senzitivitate (capacitatea clasificatorului de a identifica toate cazurile pozitive), specificitate (capacitatea clasificatorului de a nu identifica ca pozitive cazuri care sunt in realitate negative), precizie și regăsire:

$accuracy = (TP+TN)/(TP+FP+TN+FN)$ $errorRate=(FP+FN)\ )/(TP+FP+TN+FN)$

$sensitivity = TP / (TP + FN)$ $specificity = TN/(TN+FP)$

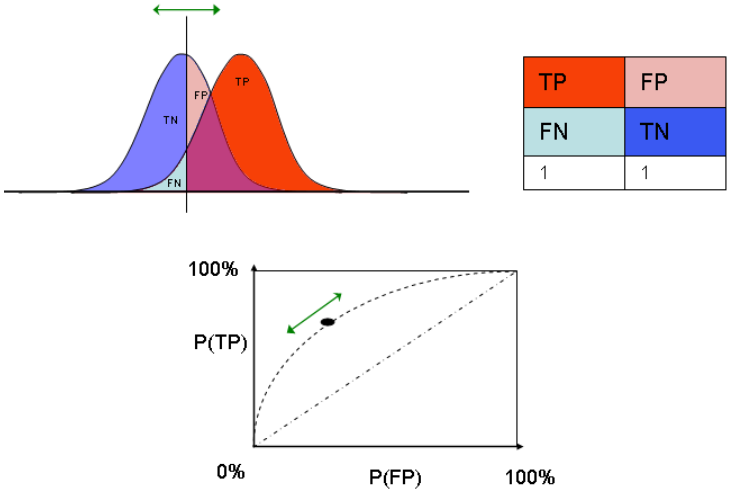
$precision = TP/(TP+FP)$ $recall = TP/(TP+FN)=sensitivity$

$F=2*precision*recall/(precision+recall)$

Specificitatea și senzitivitatea sunt folosite de regulă în analiza datelor biomedicale, iar precizia și gradul de regăsire sunt utilizate în analiza textelor (text mining).

O altă modalitate de a evalua calitatea unui clasificator binar este reprezentată de curba ROC (Receiver Operator Characteristic) care conține puncte având coordonatele *(1-specificity, sensitivity)* și care corespund la valori diferite ale pragului de decizie. Pragul de decizie este folosit pentru a decide pe baza valorii produse de neuronul aferent clasei de interes dacă data de intrare aparține sau nu clasei (dacă valoarea de ieșire a neuronului este mai mare decât pragul

atunci data aparține clasei, altfel nu aparține). Ideal este ca aria suprafeței aflată sub curba ROC să fie cât mai apropiată de 1.



2.3. Utilizarea rețelelor neuronale în rezolvarea problemelor de clasificare

Pentru rezolvarea problemelor de clasificare, arhitectura cea mai frecvent folosită este cea feedforward caracterizată prin:

- Un *nivel de intrare* având atâtea unități câte componente (atribute) au datele de intrare.
- Unul sau mai multe *nivele ascunse* (cu cât numărul de unități ascunse este mai mare cu atât modelul extras de rețea este mai complex; acest lucru poate fi un dezavantaj conducând la diminuarea capacității de generalizare a rețelei).
- Un *nivel de ieșire* având atâtea unități cât este numărul de clase.

Semnalul de ieșire corespunzător unui semnal de intrare (vector de caracteristici asociate obiectului de clasificat) indică clasa căreia îi aparține obiectul de clasificat. Variante de interpretare:

- In cazul general, indicele unității pentru care semnalul de ieșire este maxim corespunde clasei
- In cazul în care valorile produse de către unitățile de ieșire sunt cuprinse în intervalul (0,1) și suma lor este egală cu 1 atunci pot fi interpretate ca probabilități de apartenență la fiecare clasă.

3. Platforma Weka pentru explorarea datelor și învățare automată

(http://www.cs.waikato.ac.nz/ml/weka/)

- Weka este o colecție de algoritmi pentru analiza datelor și învățare automată (incluzând câteva tipuri de rețele neuronale: perceptron multinivel, rețea cu functii radiale, clasificatori cu vectori suport) proiectate să rezolve probleme de analiză a datelor: vizualizare, selecție atribute, clasificare, grupare, extragere reguli de asociere.

- Este o colecție open-source și metodele pot fi apelate atât din interfața grafică cât și din cod Java. Există trei variante principale de interfață: Explorer (pentru efectuarea de prelucrări individuale asupra unui set de date), Experimenter (pentru compararea performanțelor mai multor metode pe același set de date) și Knowledge Flow (pentru definirea unui flux de prelucrari).

Exercitiul 1. Deschideți fișierul "breast_cancer_nominal.csv" în Weka și comparați acuratețea următorilor clasificatori:

- Reguli de clasificare: Rules-> OneR, Rules-> ConjunctiveRules, Rules->NNge
- Arbori de decizie: Trees->J48, Trees->RandomForest
- Cel mai apropiat vecin: Lazy->IBk
- Retele bayesiene: bayes->NaiveBayes
- Retele neuronale: functions->MultilayerPerceptron, functions->RBFnetwork,
- Clasificatori bazați pe vectori suport: functions->SMO

Indicație: construiți un fișier Excel care conține pe o coloana tipul clasificatorului iar pe cealaltă coloana valoarea acurateței ("correctly classified instances").

4. Machine Learning Repository (http://archive.ics.uci.edu/ml/)

Arhiva conține numeroase seturi de date din diferite domenii (medicină, fizică, informatică si inginerie) care pot fi utilizate pentru antrenarea, validarea și compararea clasificatorilor și a altor metode de învățare automată.

Exercitiul 2. Vizitați colecția, descărcați unul dintre seturile de date pentru clasificare (la alegere) și utilizați-l pentru analiza metodelor de clasificare implementate în Weka.

5. Probleme de aproximare și predicție

Scop: determinarea unui model care descrie dependența între niște date de intrare (predictori) și niște date de ieșire (rezultate).

Exemple:

- Analiza datelor experimentale:
  - Datele de intrare sunt rezultate ale măsurătorilor efectuate asupra uneia/unor mărimi (de exemplu caracteristici ale unui calculator/ masini/ case)
  - Rezultatele sunt valori estimate pentru altă/alte mărimi despre care se consideră că sunt corelate cu primele (de exemplu prețul calculatorului/ masinii/ casei)
- Predicție în serii temporale:
  - Datele de intrare sunt valori anterioare ale unei mărimi care variază în timp (de exemplu temperatura, numărul de accesări ale unei resurse web, cursul de schimb valutar etc.)
  - Rezultatul este o estimare a valorii următoare din serie

5.1. Utilizarea rețelelor neuronale de tip feedforward pentru aproximare și predicție

Etape:

a) Construirea setului de antrenare (depinde de problema de rezolvat)
b) Stabilirea arhitecturii rețelei (număr de nivele, număr de unități pe fiecare nivel, funcții de activare)
c) Stabilirea parametrilor algoritmului de antrenare: rata de învățare, acuratețe (nivel toleranță la eroare), număr maxim de epoci de antrenare).
d) Antrenare propriu-zisă
e) Testarea rețelei

Implementarea rețelelor feedforward antrenate cu BackPropagation – funcții SciLab (www.scilab.org) pentru crearea și antrenarea unei rețele cu un nivel ascuns

Crearea rețelei (rețea feedforward cu un nivel ascuns):

```
function network=NNcreate(N, K, M, activationFunction1, activationFunction2)
  network=tlist(["Retea feedforward", "N", "K", "M", "X0", "Y0", "X1", "Y1", "X2", "Y2", "W1", "W2",
                 "f1", "f2"], N, K, M, zeros(N+1,1), zeros(N+1,1), zeros(K,1), zeros(K+1,1), zeros(M,1),
                  zeros(M,1), zeros(K,N+1),zeros(M,K+1), activationFunction1,activationFunction2);
endfunction
```

Funcții de activare (utilizate în rețelele neuronale antrenate cu algoritmul BackPropagation)

```
// functie de activare:  logistica
function output=logistic(x)
  output=1/(1+exp(-x));
endfunction
```

```
// functie de activare:  liniara
function output=linear(x)
  output=x;
endfunction
```

Obs. Funcția tanh este predefinită în SciLab

Calculul semnalului de ieșire (etapa Forward din algoritmul BackPropagation):

```
function network=forward(network, X)
 network.X0=[-1;X]; network.Y0=network.X0;
 network.X1=network.W1*network.Y0;
 network.Y1=[-1; feval(network.X1,network.f1)];
 network.X2=network.W2*network.Y1;
 network.Y2=feval(network.X2,network.f2);
endfunction
```

Calculul erorii medii pătratice:

```
function err=error_computation(network, tset)
err=0;
L=tset.L;
for i=1:L
    network=forward(network,tset.X(:,i));
    delta=(tset.d(i)-network.Y2).^2;
```

```
    err=err+sum(delta);
end
err=err/L;
endfunction
```

Ajustarea ponderilor (algoritmul BackPropagation):

```
function [network,ap,aE]=BP(network, tset, pmax, Emax, eta)
L=tset.L;
// initializarea ponderilor cu valori aleatoare in [-1,1]
network.W1=2*rand(network.W1)-ones(network.W1);
network.W2=2*rand(network.W2)-ones(network.W2);
E=error_computation(network,tset); // calcul eroare
p=0;   // initializare contor epoca de antrenare
ap=[]; aE=[];
while p<pmax & E>Emax
  for l=1:L
    network=forward(network,tset.X(:,l));
    delta2=tset.d(l)-network.Y2;   // calcul eroare la nivel de iesire
    if (network.f2==logistic) delta2=delta2.*(network.Y2.*(ones(network.Y2)-network.Y2)); end;
    if (network.f2==tanh) delta2=delta2.*(ones(network.Y2)-network.Y2.^2); end;
    delta=network.W2'*delta2; // propagare eroare
    if (network.f1==logistic) delta=delta.*(network.Y1.*(ones(network.Y1)-network.Y1)); end;
    if (network.f1==tanh) delta=delta.*(ones(network.Y1)-network.Y1.^2); end;
    delta1=delta(2:length(delta)); // ignorarea componentei fictive
    network.W1=network.W1+eta*(delta1*network.Y0');
    network.W2=network.W2+eta*(delta2*network.Y1');
  end;
  E=error_computation(network,tset);
  disp(E,"Eroare=",p,"p=");
  ap=[ap p]; aE=[aE E];
  p=p+1;  // incrementarea contorului de epoca
 end
endfunction
```

Aplicație 2. Reprezentarea funcției XOR (fișier XOR_BP.sci)

Construirea setului de antrenare:

```
function tset=trainingSet()
  tset=tlist(["Set antrenare","L","X","d"],4,[-1 -1 1 1;-1 1 -1 1],[-1 1 1 -1]);
endfunction
```

Aproximarea funcției XOR utilizând o rețea neuronală cu două unități ascunse și funcții de activare de tip tanh

```
function rna=approximation_XOR(hiddenUnits, eta, epochs)
rn=NNcreate(2,hiddenUnits,1,tanh,tanh); // crearea retelei
tset=trainingSet();                  // construirea setului de antrenare
rna=BP(rn,tset,epochs,0.0001,eta);      // antrenarea retelei
// testarea retelei
for i=1:tset.L;
   rna=forward(rna,tset.X(:,i));
```

```
   disp(tset.X(:,i),"input="");       // afisare valori de intrare
   disp(rna.Y2,"output="");       // afisare valoare de iesire
end
endfunction
```

Apelul funcției de aproximare:

```
approximation_XOR(10,0.1,1000);
```

Exerciții:

1.Analizați influența numărului de unități ascunse și a ratei de învățare asupra performanței rețelei (Indicație: valori de testat pentru hiddenUnits: 1,2,5,20; valori de testat pentru eta: 0.01,0.1,0.5)

2. Analizați influența funcțiilor de activare asupra abilității rețelei de a aproxima XOR: (logistic, logistic), (tanh,logistic), (logistic, tanh). (Indicație: în cazul în care se utilizeaza funcția logistică la nivelul de ieșire în setul de antrenare răspunsurile corecte trebuie să fie [0,1,1,0] în loc de [-1,1,1,-1]

Aplicație 3. Aproximarea unei funcții neliniare (f:[a,b]->R) pornind de la un set de puncte corespunzătoare graficului său. (fișier regresieBP.sci)

Preluarea datelor de intrare (puncte aflate în apropierea graficului funcției de aproximat) – punctele se selectează folosind butonul drept al mouse-ului; finalizarea selectării punctelor se realizează folosind butonul din stânga al mouse-ului):

```
// construirea setului de antrenare
function tset=trainingSet()
   tset=tlist(["Training set","L","X","d"],0,zeros(1,100),zeros(1,100));
   clf;
   plot2d(0,0,0,rect=[0,0,10,10]); // definirea domeniului si a codomeniului
   xgrid(3);
   x=locate(-1,1);
   tset.L=size(x,2);
   tset.X(1:tset.L) = x(1,:);
   tset.d(1:tset.L) = x(2,:);
endfunction
```

Aproximarea funcției pornind de la punctele selectate:

```
function tset=regression(hiddenUnits, eta, epochs)
rn=NNcreate(1,hiddenUnits,1,logistic,linear); // crearea retelei
disp("dupa creare retea");
tset=trainingSet();          // construirea setului de antrenare
inf=min(tset.X); sup=max(tset.X);
rna=BP(rn,tset,epochs,0.0001,eta);  // antrenarea retelei
// testarea retelei
x=inf:(sup-inf)/100.:sup;
y=[];
for i=1:size(x,2);
   rna=forward(rna,x(1,i));
   y=[y rna.Y2];
end
```

```
plot(tset.X,tset.d,'b*',x,y,'r-');
endfunction
```

Apelul funcției de regresie:

```
regression(10,0.01,10000)
```

Exerciții:

1.Analizați influența numărului de unități ascunse și a ratei de învățare asupra performanței rețelei (Indicație: valori de testat pentru hiddenUnits: 1,2,5,20; valori de testat pentru eta: 0.01,0.1,0.5)

2. Analizați influența funcției de activare de la nivelul ascuns (tanh în loc de logistic)

Aplicație 4. Aproximarea unei serii temporale (fișier predictieBP.sci)

Considerăm o succesiune de valori (serie temporală): $x_1,x_2,\ldots,x_n$ care pot fi interpretate ca valori înregistrate la momente succesive de timp. Scopul este să se estimeze valoarea corespunzătoare momentului (n+1). Ideea principală este de a presupune că valoarea x_i depinde de N valori anterioare: x_{i-1}, $x_{i-2},\ldots,x_{i-N}$. Pe baza acestei ipoteze se poate proiecta o rețea neuronală care să învețe modelul dependenței dintre o valoare din serie și N valori anterioare. Rețeaua va avea N unități de intrare, un anumit număr de unități ascunse și o unitate de ieșire. Setul de antrenare va avea L=n-N perechi de forma (data intrare, răspuns corect) adică va fi: $\{((x_1,x_2,\ldots,x_N),x_{N+1}),((x_2,x_3,\ldots,x_{N+1}),x_{N+2}),\ldots,((x_{n-N},x_{n-N+1},\ldots,x_{n-1}),x_n)\}$.

Presupunem că într-un fișier text se află valori ale cursului valutar înregistrate într-o anumită perioadă. Pe fiecare linie din fișier se află o valoare, iar prima linie corespunde celei mai recente valori înregistrate.

Construirea setului de antrenare:

```
// construirea setului de antrenare
function tset=trainingSet(inputUnits)
  tset=tlist(["Training set","L","X","d"],0,zeros(1,100),zeros(1,100));
  date1=csvRead("cursEuroZilnic.csv")';
  date2(1:length(date1))=date1(length(date1):-1:1);
  tset.L=size(date2,2)-inputUnits;
  tset.X=zeros(inputUnits,tset.L);
  tset.d=zeros(1,tset.L);
  for i=1:tset.L
    tset.X(:,i)=date2(1,i:i+inputUnits-1)';
    tset.d(i)=date2(1,i+inputUnits);
  end
endfunction
```

Crearea, antrenarea și vizualizarea modelului descoperit de rețea:

```
function tset=prediction(inputUnits, hiddenUnits, eta, epochs)
rn=NNcreate(inputUnits,hiddenUnits,1,logistic,linear); // crearea retelei
tset=trainingSet(inputUnits);          // construirea setului de antrenare
```

```
rna=BP(rn,tset,epochs,0.0001,eta);  // antrenarea retelei
y=[];
for i=1:tset.L;
   rna=forward(rna,tset.X(:,i));
   y=[y rna.Y2];
end
plot(tset.d,'b-',y,'r-');
endfunction
```

Implementarea rețelelor RBF – funcții SciLab pentru crearea și antrenarea unei rețele cu un nivel ascuns

Crearea unei rețele RBF

```
function network=RBFcreate(N, K, M, sigma)
  network=tlist(["RBF NN", "N", "K", "M", "X0", "Y0", "X1", "Y1", "X2", "Y2", "C", "W", "f", "sigma"],
                N, K, M, zeros(N,1), zeros(N,1), zeros(K,1), zeros(K+1,1), zeros(M,1), zeros(M,1),
                zeros(K,N), zeros(M,K+1), gaussian,sigma);
endfunction
```

Funcție de activare:

```
function output=gaussian(x)
  output=exp(-x^2/2);
endfunction
```

Funcție de agregare:

```
function d=dist(x, y)
  d=sqrt((x-y)'*(x-y));
endfunction
```

Calcul semnal de ieșire:

```
function network=forward(network, X)
 network.X0=X; network.Y0=network.X0;
 for k=1:network.K
   network.X1(k)=dist(network.C(k)',network.Y0);
 end
 network.Y1=[-1; feval(network.X1./network.sigma,network.f)];
 network.X2=network.W*network.Y1;
 network.Y2=network.X2;
endfunction
```

Calcul funcție de eroare:

```
function err=error_computation(network, tset)
err=0;
[N,L]=size(tset.X);
for i=1:L
    network=forward(network,tset.X(:,i));
    delta=(tset.d(i)-network.Y2).^2;
    err=err+sum(delta);
end
err=err/L
endfunction
```

Algoritm de antrenare (centrii coincid cu datele din setul de antrenare)

```
function [network, ap, aE]=train(network, tset, pmax, Emax, eta)
L=tset.L;
network.C=tset.X';
network.W=2*rand(network.W)-ones(network.W);
E=error_computation(network,tset);
p=0;
ap=[];aE=[];
while p<pmax & E>Emax
  E=error_computation(network,tset);
  for i=1:L
     network=forward(network,tset.X(:,i));
     y=network.W*network.Y1;
     delta=tset.d(i)-y;
     network.W=network.W+eta*(delta*network.Y1')
  end;
  E=error_computation(network,tset);
  if (modulo(p,10)==0) then
      disp(p,"Iteration:");
      disp(E,"error=");
      ap=[ap p]; aE=[aE E];
  end
  p=p+1;
 end
 disp(E,"error=");
endfunction
```

Exemplu: regresie neliniară (fisier RBFnetwork.sci)

Construire set de antrenare:

```
function tset=trainingSet()
   tset=tlist(["Training set","L","X","d"],0,zeros(1,100),zeros(1,100));
   tset.X=[0:1:10*%pi];
   tset.L=length(tset.X);
   tset.d=sin(tset.X)+0.2*rand(tset.X)-0.1*ones(tset.X);
endfunction
```

Creare rețea, antrenare, vizualizare funcție aproximată, vizualizare eroare:

```
function tset=regressionRBF(eta, epochs, sigma)
tset=trainingSet();          // construirea setului de antrenare
rbf=RBFcreate(1,length(tset.X),1,sigma); // crearea retelei
disp(tset,"tset=");
[rbfa,ap,aE]=train(rbf,tset,epochs,0.0001,eta); // antrenarea retelei
// testarea retelei
test_plot(tset,rbfa);
pause;    // pauza in executie;  se continua tastand <resume>
clf;
plot(ap,aE);
endfunction
```

Temă:

Să se modifice algoritmul de antrenare al unei rețele RBF astfel încât numărul de centri și vectorii prototip să fie stabiliți în mod dinamic (algoritm de antrenare incrementală – curs 11 slide 62).