Laborator 12

Metoda gradientului conjugat

Cautarea se face de-a lungul unor directii conjugate, ceea ce produce o convergenta mai rapida decat aplicand metoda coborarii pe gradient. Pentru a actualiza ponderile, la fiecare iteratie se cauta in directia gradientului conjugat pentru a determina valoarea care minimizeaza functia de-a lungul acelei linii.

1) **Metoda Fletcher-Reeves (traincgf)**

   La prima iteratie algoritmul cauta in directia gradientului

$$\mathbf{p}_0 = -\mathbf{g}_0$$

   Se face o cautare liniara de-a lungul directiei curente de cautare pentru a determina pasul de actualizare

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

   Urmatoarea directie de cautare se alege astfel incat sa fie conjugata cu directiile anterioare de cautare

$$\mathbf{p}_k = -\mathbf{g}_k + \beta_k \mathbf{p}_{k-1}$$

   unde

$$\beta_k = \frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}}$$

```
p = [-1 -1 2 2;0 5 0 5];
t = [-1 -1 1 1];
net=newff(minmax(p),[3,1],{'tansig','purelin'},'traincgf');
net.trainParam.show = 5;
net.trainParam.epochs = 300;
net.trainParam.goal = 1e-5;
[net,tr]=train(net,p,t);
```

Functia implicita de cautare liniara este **srchcha** (se poate schimba modificand valoarea parametrului **srchFcn**).

Rulati programul demonstrativ **nnd12cg**.

2) Metoda Polak-Ribiere (traincgp)

Diferenta fata de metoda anterioara consta in modul de calculare a lui β_k

$$\beta_k = \frac{\Delta \mathbf{g}_{k-1}^T \mathbf{g}_k}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}}$$

```
p = [-1 -1 2 2;0 5 0 5];
t = [-1 -1 1 1];
net=newff(minmax(p),[3,1],{'tansig','purelin'},'traincgp');
net.trainParam.show = 5;
net.trainParam.epochs = 300;
net.trainParam.goal = 1e-5;
[net,tr]=train(net,p,t);
```

3) Metoda Powel-Beale (traincgb)

In cazul tuturor metodelor gradientului conjugat, directia de cautare va fi resetata periodic la valoarea negativului gradientului. Momentul standard de resetare are loc cand numarul iteratiilor este egal cu numarul parametrilor retelei. Pentru metoda Powel-Beale resetarea se face cand proprietatea de ortogonalitate intre gradientul curent si cel anterior nu mai este indeplinita. Acest lucru este testat cu inegalitatea

$$\left| \mathbf{g}_{k-1}^T \mathbf{g}_k \right| \geq 0.2 \left\| \mathbf{g}_k \right\|^2$$

Daca aceasta conditie este satisfacuta, directia de cautare este resetata la negativul gradientului.

```
p = [-1 -1 2 2;0 5 0 5];
t = [-1 -1 1 1];
net=newff(minmax(p),[3,1],{'tansig','purelin'},'traincgb');
```

```
net.trainParam.show = 5;
net.trainParam.epochs = 300;
net.trainParam.goal = 1e-5;
[net,tr]=train(net,p,t);
```

Rutine de cautare liniara

1) **Golden Section Search (srchgol)**

   Primul pas este localizarea intervalului in care se afla minimul functiei de performanta prin evaluarea functiei intr-o secventa de puncte, aflate la inceput la distanta **delta** si dubland distanta la fiecare pas de-a lungul directiei de cautare. Daca performanta creste intre 2 puncte successive atunci minimul a fost localizat. Urmatorul pas este reducerea intervalului in care se afla minimul. Se aleg 2 puncte in interval si in functie de valorile functie de performanta in aceste puncte, intervalul se micsoreaza. Procedura continua pana cand lungimea intervalului devina mai mica decat **delta/scale_tol.**

   Sa se ruleze programul demonstrativ **nnd12sd1**.

2) **Brent's Search (srchbre)**

   Este o combinatie intre cautarea *golden section* si interpolarea patratica. Se incepe prin stabilirea intervalului ca la metoda anterioara, insa se evalueaza functia de performanta in mai multe puncte. Se determina functia patratica ce trece prin aceste puncte si se calculeaza minimul ei. Daca minimul cade in interval, se foloseste la urmatorul pas al cautarii si se face o noua aproximare patratica. Daca minimul cade in afara intervalului atunci se aplica un pas al metodei golden search.

3) **Hybrid Bisection-Cubic Search (srchhyb)**

   Este o combinatie intre metoda bisectiei si interpolarea cubica. Se realizeaza o interpolare cubica a unei functii folosind valorile functiei de performanta si derivatele ei in 2 punte. Daca minimul functiei obtinute cade in interval atunci se foloseste pentru a micsora intervalul. Altfel se aplica metoda bisectiei.

4) **Charalambous'Search (srchcha)**

   Este o metoda hibrida. Foloseste interpolarea cubica impreuna cu o metoda de sectionare.

Algoritmi quasi-Newton

1) **Algoritmul BFGS (trainbfg)**

Metoda lui Newton este o alternativa la metodele gradientului conjugat pentru optimizare rapida. Actualizarea parametrilor se face cu regula

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{A}_k^{-1}\mathbf{g}_k$$

unde A_k este Hessiana functei de performanta la momentul *k*. Calcularea Hessianei in cazul unei retele neurale *feedforward* necesita calcule complexe. Metodele quasi-Newton lucreaza cu aproximari ale Hessianei.

```
p = [-1 -1 2 2;0 5 0 5];
t = [-1 -1 1 1];
net=newff(minmax(p),[3,1],{'tansig','purelin'},'trainbfg');
net.trainParam.show = 5;
net.trainParam.epochs = 300;
net.trainParam.goal = 1e-5;
[net,tr]=train(net,p,t);
```

Acest algoritm necesita mai multe calcule si mai multa memorie decat metodele gradientului conjugat. La fiecare pas se memoreaza aproximarea Hessianei care are dimensiunea numarului de parametri din retea.

2) **Algoritmul One Step Secant (trainoss)**

Poate fi considerat un compromis intre metoda gradientului conjugat si metoda quasi-Newton. Nu se memoreaza matricea Hessiana completa; la fiecare pas se presupune ca aproximarea anterioara este matricea identitate.

```
p = [-1 -1 2 2;0 5 0 5];
t = [-1 -1 1 1];
net=newff(minmax(p),[3,1],{'tansig','purelin'},'trainoss');
net.trainParam.show = 5;
```

```
net.trainParam.epochs = 300;
net.trainParam.goal = 1e-5;
[net,tr]=train(net,p,t);
```

3) Algoritmul Levenberg-Marquardt (trainlm)

Cand functia de performanta este suma de termeni patratici, Hessiana poate fi aproximata astfel

$$\mathbf{H} = \mathbf{J}^T\mathbf{J}$$

si gradientul poate fi calculat

$$\mathbf{g} = \mathbf{J}^T\mathbf{e}$$

unde matricea **J** este *Jacobianul* ce contine derivatele functiei de eroare in raport cu ponderile si *bias*-urile iar **e** este vectorul erorilor. Calcularea matricei **J** este mai putin complexa decat calcularea Hessianei.
Algoritmul Levenberg-Marquardt foloseste urmatoarea regula de actualizare a parametrilor

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\mathbf{J}^T\mathbf{J} + \mu\mathbf{I}]^{-1}\mathbf{J}^T\mathbf{e}$$

Daca μ este 0 se obtine metoda Newton. Daca μ este foarte mare se obtine metoda coborarii pe gradient cu rata de invatare foarte mica. Metoda Newton este rapida in jurul minimului erorii. La o iteratie μ este micsorat daca eroarea scade si este marit daca eroarea creste. Astfel, eroarea va fi intotdeauna micsorata la fiecare pas al algoritmului.

Parametrii invatarii: **mu, mu_dec, mu_inc, mu_max.**

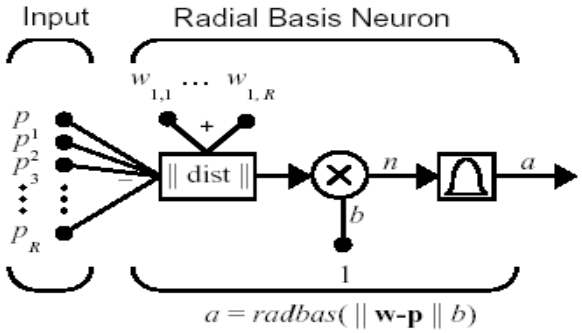
```
p = [-1 -1 2 2;0 5 0 5];
t = [-1 -1 1 1];
net=newff(minmax(p),[3,1],{'tansig','purelin'},'trainlm');
net.trainParam.show = 5;
net.trainParam.epochs = 300;
net.trainParam.goal = 1e-5;
```

[net,tr]=train(net,p,t);

Rulati programul demonstrativ **nnd12m**.

Radial Basis Networks

In figura este prezentat un neuron RB

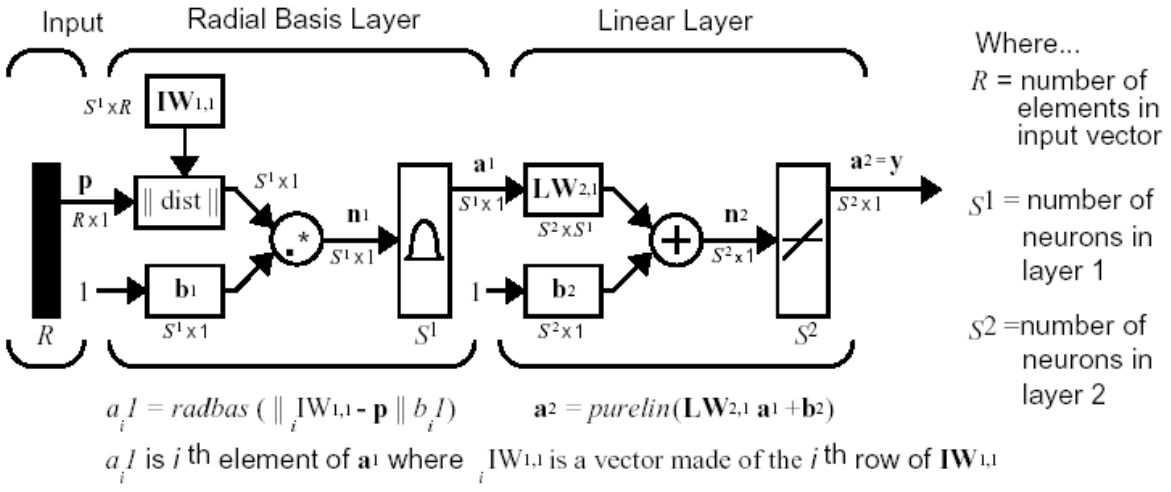


Intrarea in functia de transfer este distanta Euclidiana dintre vectorul de intrare **p** si vectorul ponderilor **w**, inmultita cu *bias*-ul **b**. Functia de transfer este

$$radbas(n) = e^{-n^2}$$

Sa se reprezinte grafic functia **radbas** pe intervalul [-5, 5].

O retea RB are 2 nivele: unul ascuns si unul de iesire.



Iesirea primului nivel al unei retele *feed forward* **net** poate fi calculata astfel:

```
a{1} = radbas(netprod(dist(net.IW{1,1},p),net.b{1}))
```

Design-ul exact al unei retele RB (newrbe)

Functia **newrbe** creeaza o retea RB care are eroarea zero pentru datele de antrenare.

net = newrbe(P,T,SPREAD)

Reteua are atatia neuroni RB cati vectori de intrare sunt in P iar ponderile din primul nivel vor fi egale cu P. *Bias*-urile din primul nivel au valoarea 0.8326/SPREAD. Astfel, iesirea din functia RB va fi mai mare decat 0.5 daca distanta dintre vectorul de intrare si vectorul de ponderi este mai mica decat SPREAD.
Parametrii din al doilea nivel sunt determinati astfel incat

[W{2,1} b{2}] * [A{1}; ones] = T

Sistemul liniar de mai sus are un numar infinit de solutii deoarece numarul de variabile este cu 1 mai mare decat numarul de constrangeri.

Dezavantajul acestei metode consta in numarul mare de neuroni necesari in nivelul ascuns.

Pentru un design mai efficient se foloseste

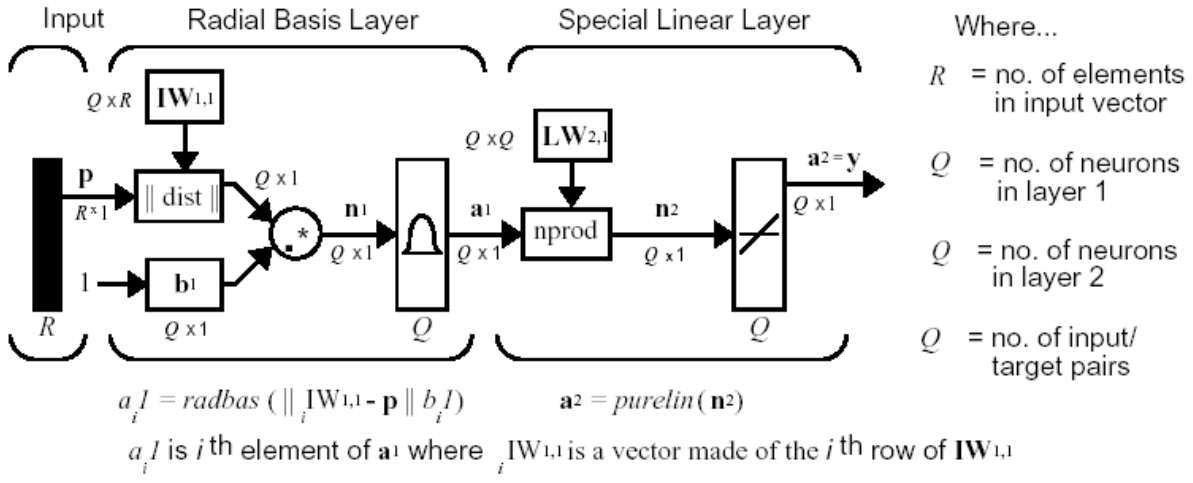
net = newrb(P,T,GOAL,SPREAD)

Diferenta fata de metoda anterioara consta in faptul ca functia **newrb** adauga neuroni pas cu pas. Se verifica eroarea, si daca este mai mica decat GOAL atunci constructia retelei se opreste. Daca nu, se mai adauga un neuron si se repeta pasul anterior.

Rulati programele demonstrative **demorb1, demorb3, demorb4**.

Retele de regresie generalizata (GRNN)

Sunt folosite pentru aproximare de functii.



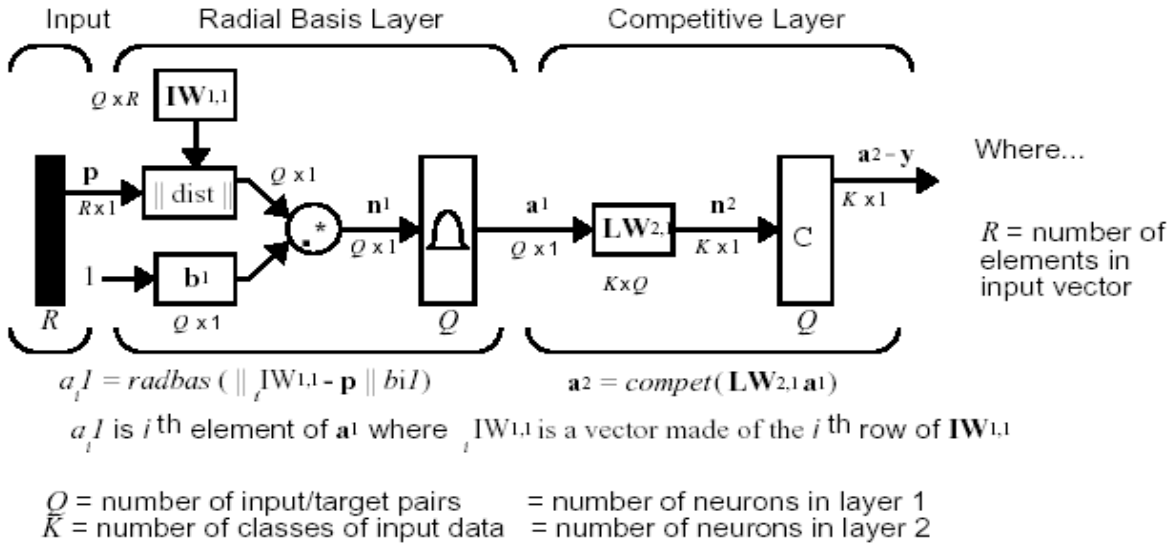
Primul nivel este construit ca in varianta de mai sus. Ponderile celui de-al doilea nivel sunt egale cu *target*-urile T.

```
P = [4 5 6];
T = [1.5 3.6 6.7];
net = newgrnn(P,T);
P = 4.5;
```

v = sim(net,P)

Rulati programul demonstrativ **demogrn1**.
Retele neurale probabiliste

Sunt folosite pentru probleme de clasificare.



Target-urile sunt vectori de dimensiune K, pe o singura componenta au valoarea 1 si in rest 0 (1 corespunde clasei din care face parte intrarea corespunzatoare). Ponderile din al doilea nivel au valoarea T iar functia de transfer este **compet** (returneaza 1 pe componenta de valoare maxima si 0 in rest)

```
P = [0 0;1 1;0 3;1 4;3 1;4 1;4 3]'
Tc = [1 1 2 2 3 3 3];
T = ind2vec(Tc)
net = newpnn(P,T);
Y = sim(net,P)
Yc = vec2ind(Y)
```

Simulati reteaua pentru intrarile P2 = [1 4;0 1;5 2]'