. 3 .
REȚELE NEURONALE UNIDIRECȚIONALE CU FUNCȚII DE ACTIVARE LINIARE. REZOLVAREA PROBLEMELOR DE APROXIMARE LINIARĂ ȘI PREDICȚIE TEMPORALĂ

Obiectivele lucrării

- prezentarea reţelelor unidirecționale cu funcții de activare liniare (rețele AdaLiNe – Adaptive Linear Networks);
- prezentarea problemelor de aproximare a funcţiilor liniare şi de prelucrare a seriilor temporale și utilizarea rețelelor liniare pentru rezolvarea acestor tipuri de problem;
- rezolvarea unor probleme de aproximare liniară și prelucrare a seriilor temporale folosind funcțiile pachetului Neural Network Toolbox al mediului MATLAB.

Noțiuni teoretice

Rețelele neuronale cu funcţii de activare liniare folosesc algoritmi de antrenare supervizată care se bazează pe căutarea punctului de minim al unei funcții criteriu prin metode de gradient. Această metodă impune utilizarea unor funcții de activare derivabile pe întreg domeniul de definiție sau cel puțin pe un domeniu de funcționare mai restrâns. Problemele care se pot rezolva cu ajutorul acestui tip de rețele presupun evaluarea numerică a erorii de antrenare (nu calitativă ca în cazul perceptronului), aceasta fiind inclusă în funcția criteriu ce trebuie minimizată. Ajustarea ponderilor rețelei depinde de nivelul erorii înregistrate la fiecare iterație a algoritmului de antrenare. Dacă, la un moment dat, eroarea de antrenare coboară sub o valoare admisă, atunci se consideră că rețeaua este “suficient” antrenată. Astfel, antrenarea rețelei este posibilă și eficientă chiar dacă datele de antrenare sunt afectate de erori.

Modelul neuronului liniar

Modelul unei unități funcționale din cadrul unei reţele liniare este similar modelului neuronal de bază, adică este descris de relațiile:

$$s = \sum_{i=1}^{n} w_i x_i + b \tag{3.1}$$

$$y = f(s) \tag{3.2}$$

Elementul particular este funcția de activare liniară pe un anumit domeniu, cum ar fi funcția de tip identitate

$$f(s) = s \tag{3.3}$$

cel mai frecvent folosită, sau, în anumite condiții, funcțiile de activare liniare cu saturație asimetrică sau simetrică:

$$f(s) = \begin{cases} 0, s < 0 \\ s, 0 \le s \le 1 \\ 1, s > 1 \end{cases} \tag{3.4}$$

$$f(s) = \begin{cases} -1, s < -1 \\ s, -1 \le s \le 1 \\ 1, s > 1 \end{cases} \tag{3.5}$$

Aproximarea unei funcții liniare (regresia liniară)

Se consideră P perechi de vectori $(\mathbf{x}^k, \mathbf{d}^k)$, cu $\mathbf{x}^k \in \mathrm{P}^{n\times 1}$, $\mathbf{d}^k \in \mathrm{P}^{m\times 1}$ și $k = \overline{1,P}$. (Am păstrat notații deja introduse în lucrările anterioare, deși în acest context semnificația lor este puțin diferită.) O problemă de aproximare liniară a acestor date se referă la determinarea unei funcții liniare $f : \mathrm{P}^n \to \mathrm{P}^m$, de forma $f(\mathbf{x}) = \mathbf{w}\mathbf{x}$, care să aproximeze cât mai bine relația necunoscută (și posibil neliniară) dintre $\mathbf{x}^k$ și $\mathbf{d}^k$, pentru toate cele P perechi.

Rețeaua neuronală liniară poate fi utilizată în rezolvarea problemelor din această clasă datorită funcției de activare de tip identitate și reprezintă o abordare alternative celei prezentate în lucrarea {Regresia lineară}.

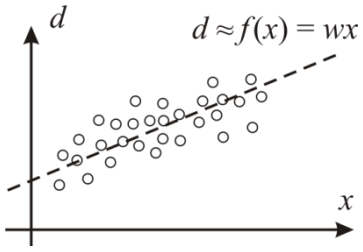

Figura 3.1. Aproximarea liniară a unui set de date (regresia liniară).

Prelucrarea seriilor temporale

O serie temporală este o set de valori, x_k, $k=\overline{1,L}$ asociate unei mărimi ce evoluează în timp, notată $x(t)$. În multe aplicații, seria temporală reprezintă eșantioanele unui semnal continuu înregistrat.

Există situaţii în care semnalul $x(t)$ este distorsionat printr-o transformare necunoscută și pe care dorim să o determinăm. Cunoscând eşantioane din semnalul inițial $x(t)$ şi din cel distorsionat $\tilde{x}(t)$, se pune problema determinării transformării care produce distorsiunea. Presupunând că această transformare este liniară, adică este de forma:

$$\tilde{x}(t)=w_0x(t)+w_1x(t-1)+\ldots+w_nx(t-n)+b \tag{3.6}$$

cu $w_i\in\mathrm{P}$, $i=\overline{0,n}$ și $b\in\mathrm{P}$, atunci determinarea ei se poate face utilizând o rețea neuronală liniară. Ponderile determinate prin procesul de antrenare a rețelei vor reprezenta chiar coeficienţii transformării.

O altă clasă de probleme de prelucrare a seriilor temporale este cea a predicției temporale. În acest caz, cunoscând ultimele n valori ale seriei (eșatioane ale semnalului $x(t)$), înregistrate la momente de timp anterioare, dorim să estimăm valoarea semnalului în momentul curent t. Dacă se poate presupune că există o dependenţă liniară între valorile seriei, adică:

$$x(t)=w_1x(t-1)+\ldots+w_nx(t-n)+b \tag{3.7}$$

atunci o reţea liniară va permite estimarea coeficienţilor w_i și b, precum şi a valorii prezise a semnalului $x(t)$.

Datele de antrenare se obțin dintr-o secvență a semnalului înregistrată apriori. Fiecare pereche de date conține un număr de eșantioane anterioare ale semnalului și valoarea lui curentă înregistrată, adică perechile de date de antrenare $(\mathbf{x},d)^k$ sunt: $\mathbf{x}=[x_1\ \ldots\ x_n]^{\mathrm{T}}$ cu $x_i=x(t-i)$ și $d=x(t)$. Această caracteristică impune utilizarea

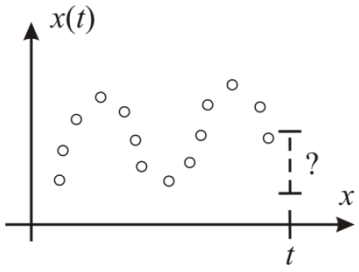

Figura 3.2. Predicția temporală.

Rețelele neuronale liniare utilizate în rezolvarea problemelor de prelucrare a seriilor de timp au un nivel de intrare cu întârzieri (eng. *tapped delay line*). Acest lucru înseamnă că prima unitate funcțională a rețelei va primi eșantionul curent $x(t)$, a doua unitate va primi eșantionul anterior $x(t-1)$, a treia va primi eșantionul $x(t-2)$ și așa mai departe (vezi Figura 3.3).

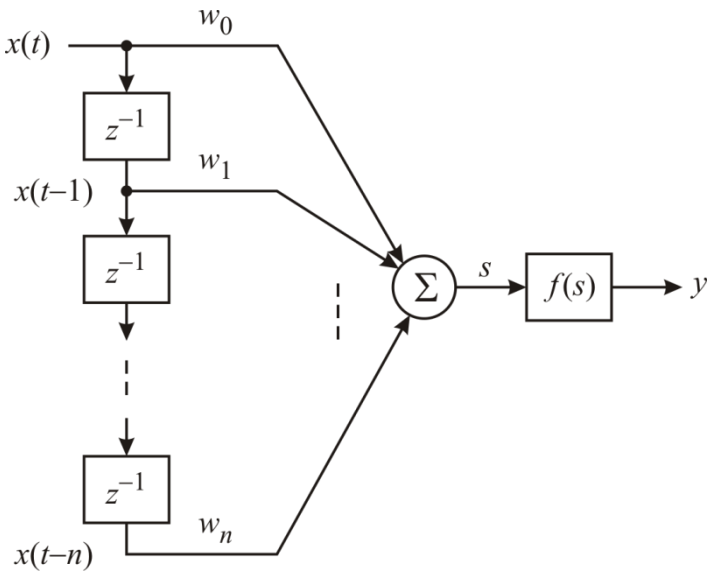

Figura 3.3. Structura neuronului artificial cu nivel de intrare cu întârzieri (tapped delay line).

Desfășurarea lucrării. Exemple

Implementarea Matlab a rețelelor neuronale liniare

În MATLAB, se pot defini și antrena rețele neuronale liniare cu un singur strat de unități funcționale prin funcții specifice. Funcțiile neuronale sunt de tip identitate, realizate prin funcția **purelin**. Antrenarea se face prin implementări ale algoritmului Widrow-Hoff, funcțiile de antrenare fiind tot cele din lucrările anterioare: **adapt**, pentru antrenarea incrementală, și **train**, pentru antrenarea pe lot.

- **Inițializarea unei rețele liniare**

Pentru inițializarea unei rețele liniare cu n semnale de intrare şi m unităţi funcţionale (semnale de ieșire) se folosește funcția **newlin**:

```
adaline = newlin([min_1 max_1; ...; min_n max_n], m, delays, learn_rate)
```

unde:

- „min_i" și „max_i" sunt valorile minimă și maximă a semnalului de intrare x_i; numărul de linii ale acestei matrice stabilește numărul de semnale de intrare;
- „m" este numărul uniţaţilor de ieşire (având funcţii de transfer de tip purelin),
- „delays" reprezintă vectorul valorilor iniţiale asociate momentelor anterioare în cazul prelucrării unei serii temporale (prin intermediul unui nivel de intrare de tip "tapped-delay"),
- „learn_rate" este rata de învăţare (ce influenţează intensitatea ajustării ponderilor).

Valorile limită ale semnalelor de intrare sunt fie adoptate pe baza cunoștințelor despre aplicația concretă, fie determinate din datele de antrenare disponibile folosind în mod corespunzător funcțiile **min** și **max**.

Ultimii doi parametri pot lipsi, valorile implicite fiind delays = [0] (adică fără întârzieri în nivelul de intrare) și learn_rate = 0,01. După inițializare, ponderile conexiunilor şi pragurile unităţilor de ieşire sunt 0.

O secvență de program pentru inițializarea unei rețele cu o unitate funcțională având două semnale de intrare în domeniul [–1;1] arăta astfel:

```
inputs_minmax = [-1 1; -1 1];
output_units = 1;
adaline = newlin(inputs_minmax, output_units);
```

Asemănător funcției de inițializare a perceptronului, funcţia **newlin** returnează o structură de date neomogenă ce conţine toate informaţiile privind reţeaua.

- **Antrenarea reţelei.**

Algoritmul de antrenare pentru o rețea liniară implementat în NN Toolbox este algoritmul Widrow-Hoff (vezi funcția **learnwh**), care minimizează eroarea medie pătratică pentru setul de date de antrenare. Algoritmul este implementat în ambele variante: serială (funcția **adapt**) și pe globală (funcția **train**). Setul de date de antrenare are aceeași structură ca în cazul antrenării perceptronului, adică este sub forma a două matrice, conținând valorile semnalelor de intrare, respectiv de ieșire.

În cazul rețelelor fără semnale de intrare întârziate, funcțiile de antrenare se pot apela astfel:

```
[trained_adaline, output, error] = adapt(adaline, train_data_input,
    train_data_output);
```

unde

- „trained_adaline" este reţeaua liniară antrenată,
- „output" returnează ieșirile reţelei calculate pentru exemplele din setul de antrenare, după antrenare,
- „error" returnează vectorul erorilor după antrenare,
- „adaline" reprezintă reţeaua inițializată (neantrenată),
- „train_data_input" și „train_data_output" conțin datele de antrenare,

respectiv

```
[trained_adaline, info] = train(adaline, train_data_input, train_data_output);
```

unde

- „info" returnează informaţii despre desfăşurarea învăţării.

Pentru **adapt** este necesar să se stabilească numărul de parcurgeri ale setului de antrenare, prin alegerea valorii câmpul „adaline.adaptParam.passes".

```
adaline.adaptParam.passes = 100;
```

În cazul funcției **train** se stabilesc atât toleranța prin câmpul „retlin.trainParam.goal", cât și numărul maxim de epoci prin câmpul „retlin.trainParam.epochs".

```
adaline.trainParam.epochs = 100;
adaline.trainParam.goal = 0.01;
```

- **Vizualizarea funcţiei de eroare.**

În timpul antrenării este automat deschisă o fereastră în care se trasează grafic în mod secvențial evoluția erorii de antrenare cu fiecare epocă a algoritmului. Pentru a obţine această reprezentare şi după ce antrenarea s-a terminat se poate folosi funcţia **plotperf**:

```
plotperf(info, adaline.trainParam.goal)
```

căruia i se transmit ca parametri variabila „info" obținută după antrenare și toleranţa la antrenare, disponibilă în câmpul „adaline.trainParam.goal".

- **Testarea rețelei antrenate.**

  În cazul în care se dorește testarea rețelei pentru diverse date de intrare (din setul de antrenare sau dintr-un set de test), se utilizează funcţia **sim**:

```
output = sim(trained_adaline, test_data_input);
```

  unde:

  - „test_data_input” este un vector coloană conținând valori de test ale semnalelor de intrare.

- **Vizualizarea rezultatelor antrenării**

  Rezultatele antrenării sunt valorile ponderilor și de prag care descriu funcțiile liniare căutate. Dacă dorim să notăm aceste funcții, atunci vom accesa valorile ponderilor prin comenzi de forma

```
trained_adaline.IW{1}
trained_adaline.b{1}
```

  care returnează valorile ponderilor unității funcționale 1 din cadrul rețelei conținută de variabila “trained_adaline”, respectiv valoarea de prag a acesteia. Pentru a obține ponderea semnalului de intrare cu indicele i putem apela:

```
trained_adaline.IW{1}(i)
```

  Așadar, pentru a calcula mărimea de ieșire am putea avea o instrucțiune de forma:

```
output = trained_adaline.IW{1} .* input + trained_adaline.b{1}
```

  Pe de altă parte, funcțiile de antrenare se pot apela cu returnare de valori, astfel:

```
[trained_adaline, output, error] = adapt(adaline, train_data_input,
    train_data_output);
```

  respectiv

```
[trained_adaline, epochs, output, error] = train(adaline, train_data_input,
    train_data_output);
```

  unde variabila output conține deja valorile de ieșire calculate în urma antrenării rețelei pentru toate exemplele de antrenare.

Exemple

- **Exemplul 3.1.**

Pentru primul exemplu, vom considera funcția liniară $f(x) = 2x + 1$ care este evaluată pentru 20 de valori ale argumentului x în intervalul $[0;2)$ și asupra căreia se suprapune un zgomot în domeniul $[-1;1]$. Generarea valorilor argumentului și funcției se face cu secțiunea următoare:

```
% argumentul funcției
x = 0:0.1:1.9;

% valorile funcției f(x) = 2x + 1, pentru fiecare valoare a argumentului x
f = 2*x + 1;

% generare de valori aleatoare pentru fiecare valoare calculată a funcției
z = -1 + 2 * rand(1, length(f));

% valorile funcției sunt afectate de zgomot
d = f + z;
```

În acest moment, avem disponibile 20 de perechi de date formate din valorile argumentului și cele afectate de zgomot ale funcției date. Pe baza acestora, vom antrena o rețea liniară cu o singură unitate funcțională și cu o singură mărime de intrare cu scopul de a obține o aproximare a funcției date.

```
% initializarea retelei liniare
adaline = newlin([min(x) max(x)], 1, [0], 0.001);

% A) antrenare cu functia adapt
% numarul de parcurgeri ale setului de date de antrenare
adaline.adaptParam.passes = 50;

% antrenarea folosind datele obținute anterior
trained_adaline_1 = adapt(adaline, x, d);

% parametrii retelei sunt coeficientii aproximarii liniare, deci calculam valorile
    acesteia
f1 = trained_adaline_1.IW{1} * x + trained_adaline_1.b{1};
```

```
% B) antrenare cu functia train
% toleranta
adaline.trainParam.goal = 1;

% numarul maxim de epoci pentru algoritmul de antrenare
adaline. trainParam.epochs = 50;

% antrenarea folosind datele obținute anterior
trained_adaline_2 = train(adaline, x, d);

% parametrii retelei sunt coeficientii aproximarii liniare, deci calculam valorile
    acesteia
f2 = trained_adaline_2.IW{1} * x + trained_adaline_2.b{1};

% reprezentarea grafica a functiei originale, datelor de antrenare
% si aproximarilor liniare obtinute
plot(x,f,'g', x,d,'o', x,f1,'r', x,f2,'m')
```

În urma unei execuții a acestui program, aproximările liniare obținute sunt:

$f_1(x) = 1{,}5424x + 1{,}2305$, respectiv

$f_2(x) = 1{,}3269x + 1{,}0802$

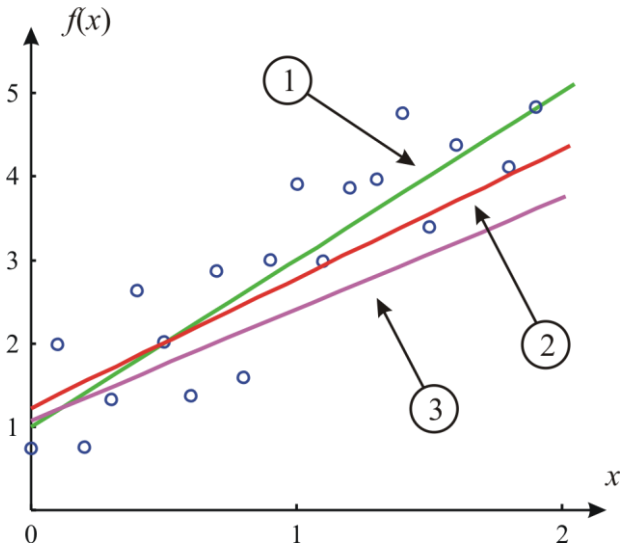

Figura 3.4. Aproximări ale unei funcții liniare (1), afectată de zgomote, obținute prin antrenarea unei rețele liniare cu **adapt** (2), respectiv cu **train** (3).

Parametrii de antrenare stabiliți (toleranța, numărul de epoci) nu sunt exigenți, fapt care conduce la obținerea acestor rezultate nesatisfăcătoare. Valorile sunt utilizate doar pentru a evidenția diferențele ce se pot obține prin cele două funcții de antrenare. Dacă însă vom stabili:

```
adaline.adaptParam.passes = 1000;
adaline.trainParam.goal = 0.01;
adaline.trainParam.epochs = 1000;
```

atunci rezultatele sunt mult îmbunătățite, identice pentru ambele variante de antrenare:

$$f_1(x) = f_2(x) = 1{,}9334x + 1{,}0039$$

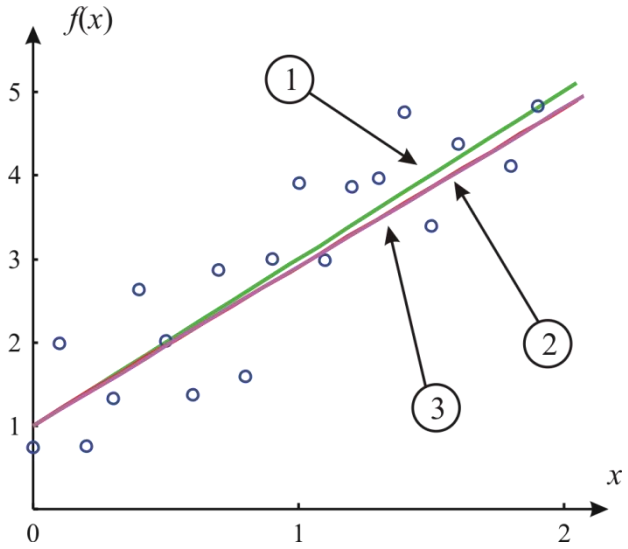

Figura 3.5. Alegerea potrivită a parametrilor de antrenare conduce la obținerea unor aproximări liniare mult îmbunătățite.

Observație. Zgomotul adăugat funcției originale este obținut folosind funcția **rand**, care generează valori aleatoare. Din acest motiv, la fiecare execuție a programului, rezultatele obținute pot să difere puțin.

- **Exemplul 3.2.**

Fie perechile de date (x, y) listate mai jos

(0 ; 2)	(1 ; 2)	(2 ; 3)	(3 ; 4)	(4 ; 3)
(5 ; 4)	(6 ; 5)	(7 ; 6)	(8 ; 6)	(9 ; 7)
(10 ; 7)	(11 ; 7)	(12 ; 8)	(13 ; 9)	(14 ; 8)
(15 ; 10)	(16 ; 10)	(17 ; 11)	(18 ; 12)	(19 ; 12)

Se cere să se determime o aproximare liniară relației dintre valorile variabilelor x și y.

- *Rezolvare*

Datele de antrenare se inițializează prin vectorii:

```
x = [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19];
y = [2 2 3 4 3 4 5 6 6 7  7  7  8  9  8 10 10 11 12 12];
```

apoi se inițializează o rețea liniară cu o singură unitate funcțională și fără întârzieri:

```
adaline = newlin([min(x) max(x)], 1, [0], 0.0001);
```

Se stabilește numărul de parcurgeri ale setului de date pentru antrenarea serială

```
adaline.adaptParam.passes = 1000;
```

iar în urma antrenării

```
trained_adaline_1 = adapt(adaline, x, y);
```

se obține funcția liniară descrisă în program de linia

```
f1 = trained_adaline_1.IW{1} * x + trained_adaline_1.b{1};
```

Rezultatul este $f_1(x) = 0{,}61x + 0{,}74$.

Observație. În acest caz, la inițializarea rețelei s-a stabilit valoarea de 0,0001 pentru rata de învățare (ultimul parametru din apelul funcției **newlin**). De regulă, o valoare prea mare face ca algoritmul de antrenare să nu conveargă la soluția problemei.

Pentru antrenarea pe lot, se stabilesc parametrii antrenării:

```
adaline.trainParam.goal = 0.01;
adaline.trainParam.epochs = 1000;
```

În urma antrenării cu linia de cod

```
trained_adaline_2 = train(adaline, x, y);
```

se obține funcția liniară descrisă prin

```
f2 = trained_adaline_2.IW{1} * x + trained_adaline_2.b{1};
```

iar rezultatul obținut este același, $f_2(x) = 0{,}61x + 0{,}74$.

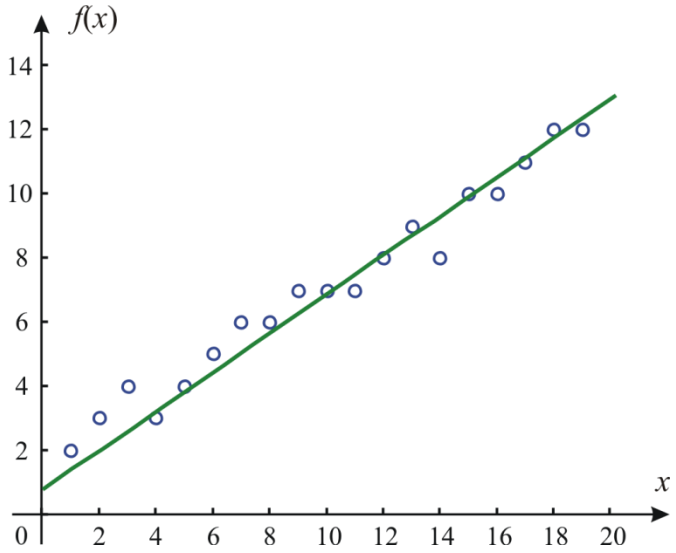

Figura 3.6. Aproximarea liniară a datelor din exemplul 3.2.

- **Exemplul 3.3.**

Fie datele de antrenare $(\mathbf{x}, y)$, cu vectorul $\mathbf{x} = [x_1 \quad x_2]$, având valorile din tabelul următor:

x_1 \ x_2	0	1	2	3	4
0	0.79	-1.02	-3.86	-5.34	-7.42
1	3.96	1.27	-0.99	-2.72	-4.58
2	6.52	4.25	2.89	0.47	-1.48
3	9.88	7.88	5.2	3.06	1.33
4	12.17	10.74	8.3	6.99	4.43

Se cere să se determime o aproximare liniară relației dintre x_1, x_2 și y.

- *Rezolvare*

Având două variabile de intrare, matricea datelor de intrare are două linii conținând toate combinațiile posibile de valori ale intrărilor. Inițializarea datelor de antrenare ar arata astfel:

```
x1 = 0:4;
x2 = 0:4;
x = combvec(x1,x2);
y = [0.79 3.96 6.52 9.88 12.17 -1.02 1.27 4.25 7.88 10.74 -3.86 -0.99 2.89 5.2 8.3 -5.34
    -2.72 0.47 3.06 6.99 -7.42 -4.58 -1.48 1.33 4.43];
```

De asemenea, inițializarea rețelei se modifică astfel:

```
adaline = newlin([min(x1) max(x1); min(x2) max(x2)], 1, [0], 0.0001);
```

Observație. Funcția **combvec** returnează o matrice cu toate combinațiile de elemente ale vectorilor cu care se apelează.

Păstrând parametrii de antrenare de la exemplul anterior, pentru ambele funcții de antrenare se obține funcția liniară:

$$f(x) = w_1 x_1 + w_2 x_2 + b = 3{,}0244x_1 - 1{,}9843x_2 + 0{,}3799$$

Cele două ponderi și valoarea de prag ai rețelei antrenate se obțin din variabila-obiect care conține rețeaua cu apeluri de forma:

```
trained_adaline_1.IW{1}(1)
trained_adaline_1.IW{1}(2)
trained_adaline_1.b{1}
```

Dacă este necesară trasarea grafică a acestei funcții, împreună cu datele de antrenare, programul se completează cu liniile următoare:

```
% Domeniul pentru care se traseaza funcția obținută, cu alt pas decat cel din datele
    initiale
[X,Y] = meshgrid(0:0.1:4,0:0.1:4);
% Funcția obținuta dupa antrenare (cu train sau adapt)
Z1 = trained_adaline_1.IW{1}(1) * X + trained_adaline_1.IW{1}(2) * Y +
    trained_adaline_1.b{1};
% Reprezentarea grafica a punctelor din datele de antrenare
scatter3(x(1,:), x(2,:), y)
hold on
% Reprezentarea grafica a funcției obținute pe domeniul definit
surf(X,Y,Z1)
```

(în care funcțiile **meshgrid**, **scatter3** și **surf** sunt funcții MATLAB).

- **Exemplul 3.4.**

Fie semnalul inițial $x(t) = 2\sin(2t+1)$ asupra căruia este aplicată o transformare $T\{x(t)\}$ care conduce la semnalul $\tilde{x}(t) = 2x(t) + x(t-1)$. Se cere să se identifice această transformare utilizând o rețea neuronală.

Observație. Este important de menționat că transformarea este considerată necunoscută, iar aici este dată doar pentru a exemplifica tipul de probleme de identificare a transformării unui

semnal. În practică, sunt disponibile eșantioanele celor două semnale, înregistrate la aceleași momente de timp.

- *Rezolvare*

Mai întâi, vom genera cele două semnale, pentru un interval de 6.28 secunde și cu perioada de eșantionare de 0,01 secunde:

```
% domeniul de timp (discret, cu perioada 0.01)
t = 0:0.01:6.28;
% semnalul initial
x = 2 * sin(2*t + 1);
% semnalul distorsionat
xt = 2 * x + [0 x(1:(end-1))];
```

Rețeaua liniară va avea o singură unitate funcțională, cu două semnale de intrare: valoarea curentă a semnalului $x(t)$ și cea anterioară $x(t-1)$ întârziată cu o iterație (perioadă de eșantionare, indice). Structura este prezentată în Figura 3.7. În această situație, inițializarea rețelei neuronale se face incluzând și întârzierile în al treilea parametru al funcției **newlin**:

```
adaline = newlin([min(x) max(x)], 1, [0 1], 0.001);
```

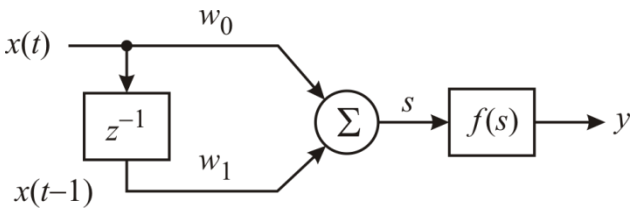

Figura 3.7. Structura neuronului liniar pentru identificarea transformării de la Exemplul 3.4.

Vom utiliza parametrii pentru funcțiile de antrenare utilizați la exemplul anterior, iar rezultatele obținute sunt:

$T\{x(t)\} = w_1 x(t) + w_2 x(t-1) + b = 2{,}9976 \cdot x(t) + 0 \cdot x(t-1) - 0{,}0027$, cu **adapt**, respectiv

$T\{x(t)\} = w_1 x(t) + w_2 x(t-1) + b = 2{,}9842 \cdot x(t) + 0 \cdot x(t-1) - 0{,}0025$, cu **train**

În Figura 3.8 se observă că transfromarea liniară obținută este foarte apropiată de cea reală. Acest lucru se datorează și faptului că transformarea initială este una liniară (situație frecventă în aplicațiile practice).

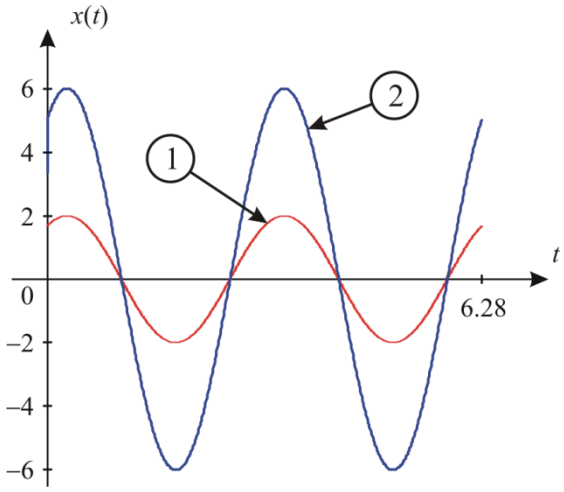

Figura 3.8. Rezultatul identificării unei transformari de semnal liniare: 1 – semnalul inițial, 2 – semnalul distorsionat și cel obținut prin transformarea identificată sunt aproape identice.

- **Exemplul 3.5.**

Pentru semnalul de la exemplul anterior, considerăm o transformare inițială neliniară și care conține două eșantioane mai vechi ale semnalului inițial, ca în relația

$$\tilde{x}(t) = 0{,}5[x(t)]^2 + x(t-2) + 0{,}5x(t-5).$$

Să se găsească o transformare liniară care aproximează această relație.

- *Rezolvare*

Semnalul distorsionat este "generat" cu linia de program:

```
xt = 0.5 * x .* x + [zeros(1,2) x(1:(end-2))] + 0.5*[zeros(1,5) x(1:(end-5))];
```

Întârzierile sunt definite în inițializarea rețelei astfel:

```
adaline = newlin([min(x) max(x)], 1, [0 2 5], 0.001);
```

având o singură unitate cu 3 semnale de intrare.

În programele de prelucrare a seriilor temporale, în care structura rețelei este cea cu întârzieri cu un pas (Figura 3.3), datele de antrenare trebuie furnizate în format de matrici de celule (*cell array*). Acestea se obțin cu funcția **con2seq**:

```
train_data_input = con2seq(x);
train_data_output = con2seq(xt);
```

Restul programului este similar celui de la exemplul anterior, iar transformările obținute cu cele două funcții de antrenare sunt identice:

$$T\{x(t)\} = w_1 x(t) + w_2 x(t-2) + w_3 x(t-5) + b = 1{,}4870 \cdot x(t) + 0{,}9887$$

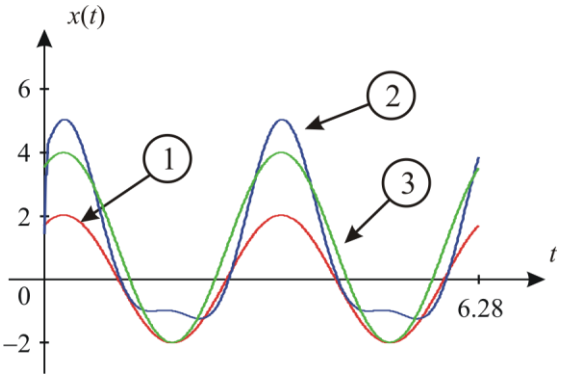

Figura 3.9. Identificarea transformării neliniare a unui semnal: 1 (roșu) – semnalul inițial, 2 (albastru) – semnalul distorsionat prin transformarea neliniară, 3 (verde) – semnalul distorsionat prin transformarea liniară obținută.

- **Exemplul 3.5.**

Considerăm o serie temporală x_k, $k = \overline{1, L}$, obținută, spre exemplu, prin eșantionarea unui semnal $x(t)$. Să se determine un model liniar pentru predicția valorii următoare a semnalului, x_{k+1}, pe baza ultimelor n eșantioane disponibile.

- *Rezolvare*

Mai întâi să construim seria temporală

```
t = 0:0.01:4;
x = cos(t) .* sin(t.*t/2);
```

Rețeaua neuronală necesară pentru rezolvarea problemei conține o singură unitate funcțională cu n semnale de intrare. Aparent, datele de antrenare ar trebui să conțină n valori de intrare, pentru ultimele n eșantioane ale seriei temporale, și valoarea dorită a ieșirii egală valoarea curentă a semnalului, adică:

$$(\mathbf{x}, d)^k = \left([x_{k-1} \quad x_{k-2} \quad \ldots \quad x_{k-n}], x_k\right)$$

Datele de antrenare nu vor avea însă această organizare, deoarece se consideră structura cu intrări întârziate descrisă în Figura 3.3. În această situație, inițializarea rețelei liniare este:

```
n = 4;
delays = [1:n];  % vectorul obtinut este [1 2 … n]
```

```
adaline = newlin([min(x) max(x)], 1, delays, 0.001);
```

datele de antrenare sunt:

```
train_data_input = con2seq(x);
train_data_output = train_data_input;
```

iar antrenarea rețelei este

```
[trained_adaline, output, error] = adapt(adaline, train_data_input,
    train_data_output);
% sau [trained_adaline, epochs, output, error] = train(adaline, train_data_input,
    train_data_output);
```

Putem evalua eficiența predicției comparând valorile corecte ale semnalul cu cele calculate de rețeaua antrenată, fie prin variabila de eroare returnată de funcția de antrenare, fie prin reprezentarea lor grafică:

```
plot(t,x,'r', t,cell2mat(output),'b');
```

Observație. Variabila returnată de funcția de antrenare este un vector de celule (*cell array*) care trebuie convertit într-un vector obișnuit, folosind funcția **cell2mat**.

Pentru o evaluare relevantă, vom compara seria de timp generată pentru un domeniu extins cu cea calculată de rețeaua antrenată, adică:

```
t = 0:0.01:8;
x = cos(t) .* sin(t.*t/2);
x_predicted = sim(trained_adaline, con2seq(x));
plot(t,x,'r', t,cell2mat(x_predicted),'b');
```

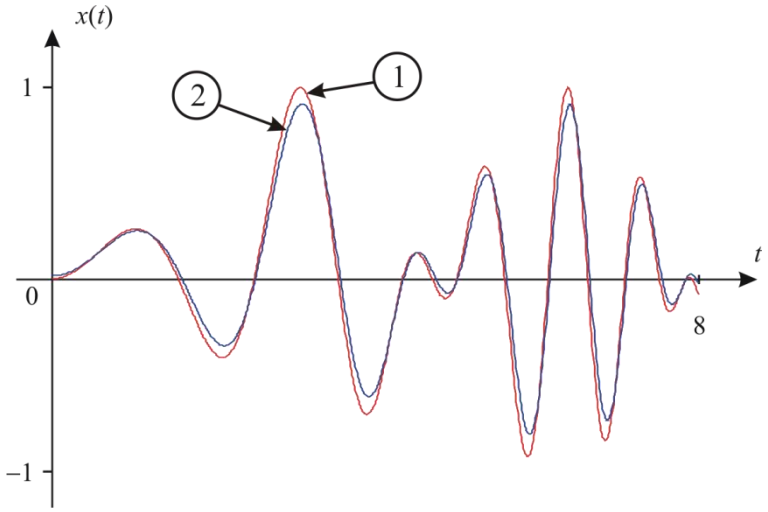

Figura 3.10. Predicția unei serii temporale: 1 (roșu) – seria temporală extinsă, 2 (albastru) – valorile determinate de rețeaua antrenată.

Probleme propuse

- **Problema 3.1**

  Să se studieze efectul parametrilor de antrenare (*passes*, *goal*, *epochs*) asupra rezultatelor obținute în urma antrenării unei rețele neuronale liniare (cu ambele funcții de antrenare).

  Indicație. Se poate lucra cu programul de la exemplul 3.2, în care se aleg diverse valori pentru parametrii de antrenare.

- **Problema 3.2**

  Să se studieze efectul valorii ratei de învățare stabilite la inițializarea rețelelor neuronale liniare asupra convergenței algoritmului, pentru ambele funcții de antrenare.

  Indicație. Se poate lucra cu programul de la exemplul 3.2, în care se aleg diverse valori pentru rata de învățare în linia de program care inițializează rețeaua.

- **Problema 3.3**

  Scrieți un program MATLAB pentru antrenarea unei rețele neuronale liniare care să calculează media aritmetică a 5 semnale de intrare.

  Indicație: datele de antrenare vor fi generate prin alegerea unor seturi de 5 valori și calcularea mediei lor.

- **Problema 3.5**

  Fișierele “eur-ron-rates-train-data.txt” și “eur-ron-rates-test-data.txt” conțin ratele de schimb EUR-RON pentru zilele lucrătoare din perioada 2016-2017, respectiv pentru ianuarie – iunie 2018. (Sursa: http://www.bnro.ro/)

  Să se scrie un program MATLAB pentru antrenarea unei rețele neuronale pe baza datelor disponibile în primul fișier, care să estimeze evoluția cursului de schimb, iar apoi pentru evaluarea predicției prin compararea evoluției estimate cu datele reale, disponibile în cel de-al doilea fișier.

  Indicație. Antrenarea rețelei se va face luând în considerare la fiecare iterație un număr relevant de eșantioane anterioare din seria de date.