MIHAELA MATCOVSCHI OCTAVIAN PĂSTRĂVANU

APLICAŢII ALE REŢELELOR NEURALE ÎN AUTOMATICĂ

Editura POLITEHNIUM

2008

Editura POLITEHNIUM
a Universităţii Tehnice „Gh. Asachi” din Iaşi
Bd. Dimitrie Mangeron nr. 67
RO-700050, Iaşi, România
Tel/Fax: +40 – 0232 – 231 343

Editura POLITEHNIUM (fostă „Gh. Asachi”) este recunoscută de
Consiliul Naţional al Cercetării Ştiinţifice
din Învăţământul Superior (CNCSIS)

Director editură:
Prof. univ. dr. ing. Mihail VOICU
Membru corespondent al Academiei Române

Referenţi ştiinţifici:
Prof. univ. dr. ing. Corneliu LAZĂR
Prof. univ. dr. ing. Doru PĂNESCU

Redactor:
Ing. Elena MATCU-ZBRANCA

Descrierea CIP a Bibliotecii Naţionale a României
MATCOVSCHI, MIHAELA HANAKO
Aplicaţii ale reţelelor neurale în automatică /
Mihaela Matcovschi, Octavian Păstrăvanu. – Iaşi:
Politehnium, 2008.
Bibliogr.
ISBN: 978-973-621-221-5

I. Păstrăvanu, Octavian

681.5

Universitatea Tehnică “Gh. Asachi” Iaşi
Facultatea de Automatică şi Calculatoare
Bd. D. Mangeron 53A, 700050 Iaşi
email: {mhanako, opastrav}@delta.ac.tuiasi.ro

Prefață

În pofida volumului mare de informații (cărți, articole, documentații de pachete software etc) existent asupra rețelelor neurale artificiale, realizarea unui material didactic eficient la nivelul primului ciclu de instruire al învățământului universitar tehnic constituie o sarcină dificilă. Dificultatea provine din condițiile restrictive pe care trebuie să le satisfacă un atare material pentru a-şi îndeplini cu adevărat rolul său formativ. Altfel spus, o carte despre aplicații ale rețelelor neurale în automatică care să capteze interesul studenților trebuie (i) să pună la dispoziție o descriere riguroasă a structurilor complexe pe care le reprezintă rețelele neurale cu utilizare reală în practică, dar fără a apela la construcțiile matematice laborioase specifice acestor probleme; (ii) să selecteze arhitecturi de rețele şi tipuri de aplicații pentru care se pot realiza experimente ilustrative pe calculator, dar fără a implica elaborarea de programe sofisticate care să deplaseze centrul de greutate al experimentelor pe partea de programare, distorsionând sensul intuitiv al experimentelor.

Autorii volumului de față au conştientizat delicata lor misiune pe tot parcursul elaborării textului. În organizarea cărții şi a fiecărui capitol în parte o importanță deosebită a avut-o experiența de predare a cunoştințelor respective studenților specializării automatică, de la Facultatea de automatică şi calculatoare a Universității Tehnice "Gh. Asachi" din Iaşi. Într-un sens mai general dar expresiv, am măsurat şi croit după talia clientului cu speranța că vestimentația rezultată va fi purtată cu plăcere. Ne-am străduit să răspundem apetitului pentru tehnologii "miraculoase" care este specific vârstei cititorilor noştri, dar, în aceeaşi măsură, am căutat să inoculăm ideea fundamentului teoretic solid ca singură premisă pentru dezvoltarea unor aplicații funcționale. În ansamblu, am încercat să realizăm un material de nivel elementar spre mediu, care să ofere o orientare profesionistă în acumularea cunoştințelor, extrăgând cititorul din sfera lecturilor facile de popularizare a ştiinței şi antrenându-l ca viitor specialist.

Volumul include câteva teme de studiu care au fost considerate drept esenţiale pentru introducerea cititorului în universul amplu al reţelelor neurale artificiale. Parcurgerea acestor teme asigură înţelegerea proprietăţilor de clasificare şi aproximare pe care le posedă anumite tipuri de reţele. Pe această bază sunt prezentate ulterior aplicaţiile cele mai importante din automatică şi anume identificare şi control neliniar cu modele de tip reţea neurală dinamică. Deşi concepută pentru profilul de instruire ingineria sistemelor, cartea poate fi utilizată şi de cititori cu formaţie profesională înrudită. Primele capitole dedicate proprietăţilor reţelelor neurale artificiale sunt accesibile oricărui cititor care posedă cunoştinţele de bază ale ingineriei electrice.

Pentru a răspunde intenţiilor noastre educaţionale, volumul include două tipuri de probleme care permit aprofundarea şi fixarea cunoştinţelor. Acestea sunt organizate în două capitole distincte şi anume: probleme care se rezolvă pe cale analitică şi, respectiv, probleme care se rezolvă prin procedee numerice. Problemele cu soluţii analitice vizează reţele cu arhitectură simplă, care pot fi abordate printr-un volum redus de calcule. Problemele rezolvabile cu ajutorul calculatorului se referă la arhitecturi mai mari, care necesită un volum mai ridicat de calcule, dar sunt în măsură de a ilustra potenţialul aplicativ real al reţelelor neurale. Problemele din prima categorie sunt acompaniate de răspunsuri şi indicaţii de rezolvare, iar pentru problemele din categoria a doua sunt furnizate informaţii despre întrebuinţarea rutinelor din pachetul Neural Network Toolbox for MATLAB elaborat de compania The MathWorks Inc. Programele MATLAB realizate sunt accesibile pe web, la adresa http://81.180.214.164/web_ARNA/index.html.

Cititorii interesaţi în dobândirea de cunoştinţe suplimentare (detalii, demonstraţii) ce depăşesc nivelul propus pentru această carte, sunt invitaţi să consulte materialul bibliografic recomandat în lista de referinţe, care include monografii cotate (la momentul elaborării cărţii) cu un impact ştiinţific pe plan internaţional foarte ridicat. De asemenea cititorilor le sunt puse la dispoziţie două anexe care furnizează informaţii sumare despre două arii conexe reţelelor neurale, şi anume o trecere în revistă a metodelor de optimizare utilizabile pentru antrenarea reţelelor neurale şi a tehnicilor de clasificare.

Autorii

Cuprins

Capitolul 1

INTRODUCERE

Studierea unor mecanisme biologice naturale, în urma abstractizării şi formalizării matematice, a condus la diferite metode de procesare a informației. Aceste tehnici de procesare, combinate cu concepte din alte domenii au fost dezvoltate pentru a permite simularea pe calculatoare analogice sau numerice. Astfel, prin simulare au fost emulate o serie fenomene naturale, dintre care marea majoritate se desfăşoară la nivelul celulelor nervoase ale creierului uman. În această conjunctură a luat ființă un nou domeniu ştiințific de sine stătător denumit *Inteligența Artificială*, care înglobează mai multe direcții de cercetare. Una dintre aceste direcții o constituie *Rețelele neurale artificiale*.

În tentativa de "deconspirare" a comportamentelor ființelor umane sau ale altor organisme vii, s-a constatat o *abordare euristică*, unde nu funcționează algoritmi expliciți (situație care în literatura ştiințifică anglo-saxonă este caracterizată prin expresia consacrată "rule-of-thumb"). Cel mai adesea este greu de formulat reguli precise conform cărora oamenii cunosc anumite fapte, prelucrează date, trag concluzii sau iau decizii. Experimentele au arătat că majoritatea persoanelor care iau anumite decizii nu pot explica, în sens algoritmic, mecanismul prin care au ajuns la respectivele decizii. Prin astfel de investigații, Inteligența Artificială a reuşit să pună în evidență deosebirile fundamentale dintre două tipuri de procesare a informației şi anume *de tip secvențial* şi *de tip contextual*. Procesarea de tip secvențial corespunde algoritmilor implementabili şi executabili pe maşini de calcul cu arhitectura clasică Von Neumann. Procesarea de tip contextual are în vedere distribuirea informației către mai multe elemente de calcul care conlucrează între ele, fiecare operând însă în baza unor principii locale. Astfel s-a constatat că în cazul de corupere / alterare a unor informații, procesarea de tip secvențial eşuează, în timp ce procesarea de tip contextual poate furniza rezultate utile.

Până în prezent nu a fost formulată încă o definiție a rețelelor neurale artificiale care să se bucure de o acceptare unanimă în literatura de specialitate. Aceasta se explică prin faptul că termenii tehnici la care s-ar putea face apel ar fi ori prea generali şi vagi, ori, dimpotrivă, prea specializați pentru cadrul impus de o definiție. Totuşi, pentru a asigura rigoarea prezentării, în capitolele ce urmează vom defini *tipuri particulare de rețelelor neurale artificiale*, întrucât cititorul va acumula treptat cunoştințele care să permită formalizarea tratării. De asemenea, pentru a simplifica limbajul, vom utiliza exprimarea "rețele neurale", suprimând atributul "artificiale", deoarece în textul nostru cu orientare tehnică nu există pericolul apariției unor confuzii prin referirea la rețele neurale biologice.

Deşi constituie un participant recent la dezvoltarea ştiinței, Rețelele neurale acoperă un teritoriu larg în procesarea informației, cu multiple conotații de interdisciplinaritate. În termeni largi, o rețea neurală *învață să reprezinte* o mulțime de *trăsături* ale mediului informațional în care funcționează. De asemenea poate învăța o serie de răspunsuri care sunt livrate în corelației cu anumiți stimuli relevanți pentru mediul în care funcționează. De asemenea, multe rețele prezintă proprietăți de *generalizare* care permit elaborarea de răspunsuri adecvate pentru stimuli de intrare nemaiîntâlniți anterior.

Având în vedere importanța conceptelor amintite în paragraful anterior, drept bază de plecare pentru studierea rețelelor neurale, considerăm utilă aprofundarea lor, după cum urmează:

- *învățarea:* Este procesul prin care, în potențialul comportamental al unei rețele neurale au loc permanent schimbări,ca rezultat al experienței. Prin acest proces, parametrii unei rețele neurale sunt supuşi unei adaptări continue (în sensul modificării valorilor parametrilor) ca urmare a stimulărilor pe care le primeşte din partea mediului. Tipul de învățare este determinat de maniera în care au loc schimbările parametrilor respectivi. Capacitatea de învățare poate fi privită ca opusul unei funcționări pre-programate, care dispune, de la bun început, de toate informațiile necesare pentru întreaga durată de funcționare.
- *reprezentarea cunoştințelor:* O rețea neurală acumulează cunoştințe despre lumea înconjurătoare. Modul de reprezentare a cunoştințelor diferă de la un tip de arhitectură de rețea neurală la altul. Totuşi există anumite principii general valabile, în sensul că informația se memorează prin intermediul unor ponderi care se alocă conexiunilor dintre elementele de bază ale rețelei.
- *mediul informațional:* O rețea neurală are un acces limitat la informații, de o factură mult mai restrictivă decât procedează o ființă umană. O rețea neurală operează cu date de intrare şi date de ieşire pe care le conectează sub forma unor aplicații (în general neliniare). Rețelele neurale sunt aplicate în probleme specifice pe domenii, fiecare domeniu având anumite tipuri de date relevante.
- *generalizarea:* Un model bun se pretează generalizării, în sensul că el poate acoperi submulțimi de date din domeniul problemei considerate, chiar dacă acele date nu au fost întâlnite pe parcursul învățării. Proprietatea de generalizare este cu atât mai bună cu cât rețeaua necesită mai puține eşantioane de învățare pentru a furniza răspunsuri corecte la

intrări necunoscute. Fără această proprietate de generalizare, rețelele neurale s-ar limita la o funcționalitate similară tabelelor de căutare.

În ciuda similitudinii dintre proprietățile enumerate mai sus pentru rețelele neurale şi acțiunile (cu denumiri identice) pe care le poate desfăşura creierul uman, atragem atenția asupra faptului că suportul care asigură funcționarea rețelelor neurale este de factură matematică, cu diverse implementări, în timp ce suportul biologic este încă în mare măsură necunoscut. În înțelegerea corectă a rolului şi limitelor de utilizare a rețelelor neurale, trebuie evitată orice confuzie, de tip falsă popularizare a ştiinței, care ar crea impresia că rețelele neurale pot fi (oricând şi oriunde) un substitut al creierului uman. Purtând denumirea de rețele neurale pentru a sugera că funcționarea lor este de inspirație biologică, acestea sunt instrumente de procesare a informației, cu avantaje şi dezavantaje ca orice altă ustensilă inventată de om în scopul cunoaşterii. Deşi poate părea desuet, considerăm că cititorul trebuie să fie pe deplin avertizat că de la stadiul actual de dezvoltare a Inteligenței Artificiale, în general şi a Rețelelor neurale, în particular, până la creația tehnică (robotul) cu comportament integral humanoid (prezent în diverse ficțiuni), ştiința şi tehnologia mai au de parcurs paşi însemnați.

La momentul de față, potențialele beneficii obținute prin utilizarea rețelelor neurale se referă la: abilitatea de a extrage date / informații fără specificarea unui anumit model; abilitatea de generalizare a unor date / informații necunoscute anterior; dezvoltarea suportului tehnico-ştiințific pentru decizii autonome (neasistate); dezvoltarea sistemelor de interfațare "prietenoase"; dotarea unor produse cu un anumit grad de "inteligență". Din punct de vedere practic, aplicativ, aceste beneficii trebuie evaluate şi prin prisma complexității problemei care se doreşte a fi rezolvată făcând apel la teoria şi tehnologia rețelelor neurale. Altfel spus, se vor căuta răspunsuri la întrebări de tipul: Există o soluție alternativă pentru utilizarea rețelelor neurale? Prin ce este mai profitabilă soluția bazată pe rețele neurale decât soluția alternativă? Care este gradul de dificultate în implementarea soluției bazate pe rețele neurale? Există personalul cu calificarea necesară pentru exploatarea unei aplicații bazată pe rețele neurale? Răspunsuri corecte la aceste întrebări pot evita experiențe nefructuoase care pot conduce (fără reale justificări) la instalarea unei atitudini de adversitate față de utilizarea rețelelor neurale.

Dezvoltarea Inteligenței Artificiale în general şi a rețelelor neurale în particular a stimulat şi progresul unor domenii ştiințifice învecinate, cum ar fi Procesarea semnalelor, Teoria şi ingineria controlului, Vederea computerizată, Analiza şi sinteza vorbirii, Transmisiunea informației etc. Prin preluarea şi adaptarea unor rezultate şi / sau instrumente, la începutul anilor 90, domeniile amintite au promovat noi direcții de cercetare care ulterior s-au dovedit foarte productive.

Ultimul deceniu a reuşit să pună în valoare studiile şi cercetările experimentale asupra rețelelor neurale prin realizarea unor aplicații de anvergură industrială. În paragraful curent enumerăm câteva dintre acestea, în baza informațiilor furnizate de (Howard Demuth, Mark Beale, Martin Hagan, 2008, Neural Network Toolbox User's Guide, The MathWorks Inc.):

- Tehnologii aerospațiale: Pilot automat de înaltă performanță pentru aeronave; simularea şi detecția erorilor pentru componentele aeronavelor.
- Tehnologii pentru autovehicule: analiza activităților de garanție
- Tehnologii pentru electronică: predicția secvențelor de codare, realizarea layout-ului pentru chip-uri cu circuite integrate, sinteză vocală, recunoaştere de imagini şi vorbire, conversie text-vorbire.
- Tehnologii pentru telecomunicații: compresii de date şi imagini, traducere automată, servicii de informații automatizate, sisteme de procesare a plăților
- Tehnologii de fabricație: analiză şi proiectare de produse chimice; sisteme de control al calității (bere, sudură, hârtie, chipuri, pulberi şi particule); planificare şi management de proiecte
- Tehnici de conducere şi diagnoză pentru: procese de fabricație, roboți, manipulatoare.
- Tehnici şi servicii medicale: analiză automatizată de celule, analiză automatizată a diagramelor EEG şi EKG; proiectare de proteze; asistarea unor teste ATI
- Operații şi servicii financiar-bancare: evaluări de proprietăți; consiliere împrumuturi; analiza liniilor de credit, investigații ipotecare; urmărirea activității de pe cărțile de credit; analiză şi predicție financiară; citirea automată a unor documente; evaluarea solicitărilor de credite; predicția piețelor
- Operații şi servicii de asigurări: evaluarea solicitărilor pentru polițe; optimizarea produselor asigurate
- Servicii de transport: planificarea şi routarea vehiculelor
- Televiziune şi cinematografie: animații şi efecte speciale:
- Tehnologii militare: ghidarea diferitelor tipuri de armament; urmărirea țintelor; clasificarea obiectelor, recunoaştere facială, tipuri noi de senzori; noi strategii de procesare a informațiilor provenite de la sonare şi radare, noi strategii de identificare a semnalelor şi imaginilor.

Capitolul 2

ARHITECTURI DE REȚELE NEURALE FEEDFORWARD – PRINCIPII GENERALE DE ORGANIZARE

Rețelele neurale feedforward sunt sisteme statice, adică sisteme ce realizează un transfer instantaneu intrare-ieşire. Acest transfer poate fi descris prin compunerea unor funcții dependente de topologia (arhitectura) rețelei. Termenul de „feedforward" (însemnând „cu alimentare înainte" – semnalele propagându-se numai în sensul de la intrare către ieşire) marchează diferența față de *rețelele neurale recurente*, sau cu feedback, care sunt sisteme dinamice, posedând căi de reacție ce permit propagarea semnalelor şi de la ieşire către intrare. Deosebirea dintre cele două clase de rețele neurale, introdusă prin terminologie, se regăseşte şi în descrierea matematică a transferului intrare-ieşire. Rețelele neurale recurente sunt sisteme dinamice şi, drept urmare descrierea lor se realizează prin ecuații diferențiale (în cazul rețelelor cu timp continuu) şi prin ecuații cu diferențe (pentru rețelele cu timp discret). Elementul de bază în structura unei rețele neurale (feedforward sau recurente) îl constituie *neuronul artificial*. Pentru construirea unei rețele neurale, neuronii artificiali se conectează în unul sau mai multe straturi, definind, astfel, *arhitectura rețelei*.

Rețelele neurale artificiale s-au dezvoltat ca obiect de studiu în domeniul informaticii, matematicii aplicate, calculatoarelor, etc. Ele prezintă proprietăți de factură matematică (riguros dovedite) care seamănă cu o serie de activități curente desfăşurate de mintea umană (învățare, recunoaştere, aproximare, adaptare). Aceste activități desfăşurate de ființa umană au inspirat ca denumire proprietățile rețelelor neurale artificiale, adică în sens ştiințific aproximarea, învățarea, etc. a unei rețele neurale artificiale înseamnă un concept foarte precis definit matematic.

Punctul de vedere formulat de acest curs: rețelele neurale artificiale ca parte componentă a inteligenței artificiale (inteligență computațională) oferă instrumente cu fundament matematic care aparent copie activitatea creierului uman. În prezent, inteligența artificială nu a deconspirat mecanismele raționamentului uman; ea pune la dispoziție doar unele instrumente de factură matematică care permit abordări euristice.

Rețelele neurale artificiale au drept componentă de bază neuronul artificial. Datorită numărului mare de neuroni structura ce descrie rețeaua neurală operează cu un număr mare de semnale de intrare, interne şi de ieşire. Din acest motiv, tratarea matematică este specifică sistemelor de dimensiuni mari şi operării cu proprietăți generice. Pe tot parcursul cursului, proprietăți de factură matematică le vom analiza pe cazuri simple ce evită complexitatea rețelelor de lucru (rațiuni didactice); rețelele de lucru pentru care se urmăresc performanțe au întotdeauna dimensiuni mari.

Tipurile de rețele neurale artificiale folosite azi în practică au rezultat ca urmare a încercărilor mai vechi de a modela activitățile neuronilor biologici şi a creierului. Modelarea neuronului biologic este o problemă forte complexă şi, la un anumit moment al dezvoltării de modele, s-a constatat că ceea ce se vroia a fi modele pentru activitatea nervoasă biologică, puteau fi folosite ca instrumente de cunoaştere matematică. Modelele la nivelul respectiv au devenit de fapt ceea ce azi denumim neuron artificial, rețele neurale artificiale.

Altfel spus, actualmente neuronul artificial nu mai e privit ca un model al neuronului biologic. În prezent cercetările pentru modelarea neuronului biologic şi al sistemului nervos țin de biomedicină, bioinformatică.

2.1. Modelul neuronului artificial

Neuronul artificial constituie elementul fundamental din structura unei rețele neurale. În această secțiune, vom prezenta neuronul artificial într-o manieră suficient de generală, care să permită unele particularizări pe parcursul capitolelor ce urmează. Pentru a asigura înțelegerea cât mai deplină a noțiunilor, expunerea se realizează gradat, tratând, inițial, cazul neuronului cu o intrare şi, apoi, cazul neuronului cu mai multe intrări.

2.1.1. Neuronul artificial cu o singură intrare

Pentru prezentarea arhitecturii vom utiliza descrieri de tip diagramă bloc (schemă bloc) cu semnificați simbolurilor de la disciplina „Introducere în Automatică”. În figura 2.1.1 se prezintă schema bloc a unui neuron cu o singură intrare (intrare scalară), notată $x \in \mathbb{R}$ şi ieşirea scalară $y \in \mathbb{R}$. Constanta $w \in \mathbb{R}$ se numeşte *pondere* (eng. *weight*), iar constanta $b \in \mathbb{R}$ poartă denumirea de *deplasare* (eng. *bias*). Intrarea cu valoarea precizată “1” arată că, indiferent de semnalul de intrare *x*, valoarea cu care este alimentat blocul *b* este tot timpul 1. Referirea comună a constantelor $w, b \in \mathbb{R}$ se face prin termenul de “*parametrii neuronului*”.

În utilizarea neuronului, valorile parametrilor $w, b \in \mathbb{R}$ sunt *ajustabile*. Pe ajustarea adecvată a acestor valori se bazează capacitatea de învățare a neuronului artificial.

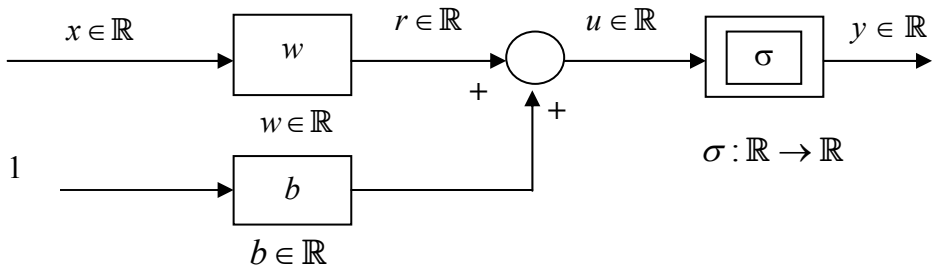


Figura 2.1.1. *Schema bloc a unui neuron cu o singură intrare*

Funcția $\sigma : \mathbb{R} \to \mathbb{R}$ se numeşte *funcția de activare a neuronului* (eng. *activation function*). Pentru funcția de activare σ se folosesc *funcții analitice standard*, liniare sau neliniare. Blocul σ defineşte *nodul* neuronului, iar ansamblul care generează semnalul u defineşte partea *liniară*.

Funcția de activare σ aparține unor clase de funcții care au anumite destinații in raport cu aplicațiile practice de utilizare. Clasele de funcții uzuale sunt liniară, treaptă (unipolară sau bipolară) şi sigmoidală (unipolară sau bipolară). În tabelul 2.1.1 sunt prezentate analitic şi grafic funcțiile de activare cu cea mai frecventă utilizare în practică; de asemenea sunt furnizate *numele funcțiilor* MATLAB aferente, disponibile în ***Neural Network Toolbox***.

Revenind la schema bloc din figura 2.1.1, se constată că neuronul artificial cu o singură intrare este *un sistem static* (asigură transferul *instantaneu* al semnalului de intrare către ieşire, spre deosebire de un sistem dinamic, a cărui ieşire depinde atât de intrarea la momentul respectiv cât şi de istoria procesului), a cărui funcționare este descrisă prin aplicația:

$$f : \mathbb{R} \to \mathbb{R}, \tag{2.1.1}$$

$$y = f(x) = \sigma(wx + b). \tag{2.1.2}$$

Aşadar f este o funcție realizată prin *compunere*, în următoarea manieră ilustrată sugestiv de schema bloc din figura 2.1.1:

$$y = f(x) = \sigma(u(x)), \tag{2.1.3}$$

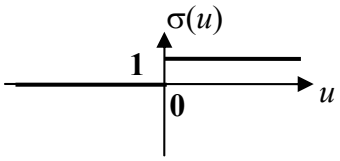
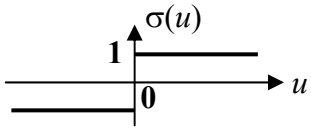
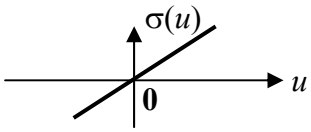
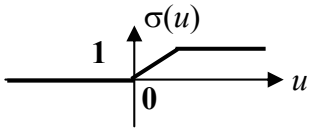
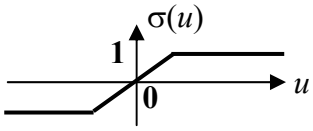
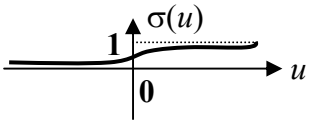
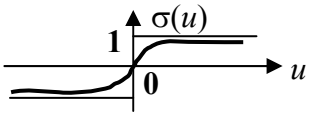
unde:

$$u(x) = r(x) + b, \tag{2.1.4}$$

$$r(x) = wx. \tag{2.1.5}$$

Din expresia lui $y = f(x)$ dată in relația (2.1.2), se observă că *transferul instantaneu* intrare-ieşire $x \to y$ este *parametrizat* de $w, b \in \mathbb{R}$ sau, cu alte cuvinte, *dependența* lui y de x este *parametrizată* de valorile ponderii $w \in \mathbb{R}$ şi deplasării $b \in \mathbb{R}$. Subliniem faptul că, deşi conform (2.1.2), ieşirea y se prezintă ca funcție de trei variabile reale (x, w, b), *din punct de vedere sistemic*, variabila x este *singura* cu semnificație de *semnal de intrare*, remarcă pusă in evidență de către schema bloc din fig. 2.1.1. Dintr-un semnal de intrare x nu se poate obține orice formă a semnalului de ieşire y (parametrizat de w şi b), cu alte cuvinte, y va păstra informații specifice nodului.

Tabelul 2.1.1. *Funcții de activare uzuale*

Nume Funcție	Expresie analitică	Reprezentare Grafică	Nume funcție MATLAB
Treaptă unipolară	$\sigma(u)=\begin{cases}0, & u<0\\ 1, & u\geq 0\end{cases}$		hardlim
Treaptă bipolară	$\sigma(u)=\begin{cases}-1, & u<0\\ 1, & u\geq 0\end{cases}$		hardlims
Funcție liniară	$\sigma(u)=u$		purelin
Liniară cu saturație	$\sigma(u)=\begin{cases}0, & u<0\\ u, & 0\leq u\leq 1\\ 1, & u>1\end{cases}$		satlin
Liniară cu saturație, simetrică	$\sigma(u)=\begin{cases}0, & u<-1\\ u, & -1\leq u\leq 1\\ 1, & u>1\end{cases}$		satlins
Sigmoid unipolar	$\sigma(u)=\dfrac{1}{1+e^{-u}}$		logsig
Sigmoid bipolar	$\sigma(u)=\dfrac{e^{u}-e^{-u}}{e^{u}+e^{-u}}$		tansig

Observație

"1" ca semnal care alimentează blocul *b* nu este tratat ca o intrare efectivă, întrucât valoarea lui nu se poate schimba (este un semnal constant). Dacă folosim pentru "1" interpretarea de semnal de intrare, atunci putem considera vectorul $\tilde{\boldsymbol{x}}=\begin{bmatrix}x & 1\end{bmatrix}^T$ ca un semnal vectorial extins cu a doua componentă constantă, egală cu 1. Notând $\boldsymbol{\theta}=\begin{bmatrix}w & b\end{bmatrix}$ (vectorul parametrilor

neuronului), obținem $u(x) = \boldsymbol{\theta}\,\tilde{\boldsymbol{x}}$. Această convenție de notație prezintă avantajul că, prin înglobarea deplasării *b* într-un vector, se ajunge la o scriere liniară, mai compactă. ■

2.1.2. Neuronul artificial cu mai multe intrări

Neuronul cu mai multe intrări generalizează modelul prezentat în paragraful anterior, în sensul că *semnalul de intrare este vectorial* (posedă mai multe componente). În figura 2.1.2 se prezintă schema detaliată a unui neuron cu ieşirea $y \in \mathbb{R}$ şi cu *n* intrări, notate $x_i \in \mathbb{R}$, $i = 1,\ldots,n$.

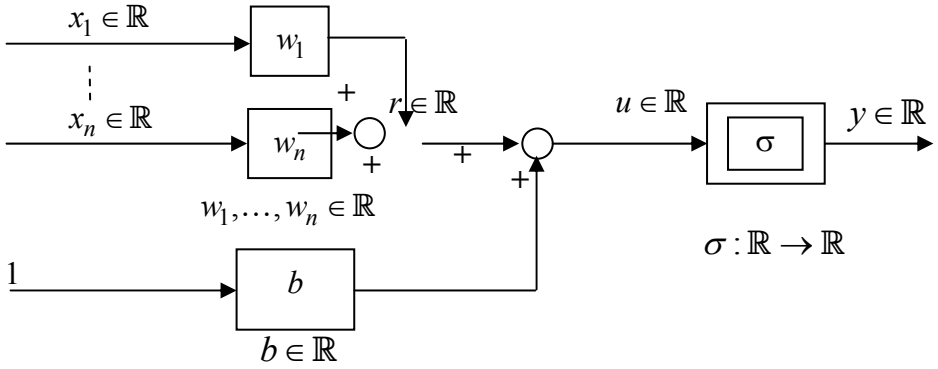


Figura 2.1.2. *Schema detaliată a unui neuron cu n intrări*

Funcția de activare rămâne neschimbată, dar intrarea ei, *u*, are expresia:

$$u = w_1 \cdot x_1 + \ldots + w_n \cdot x_n + b = \sum_{i=1}^{n} w_i x_i + b\,, \tag{2.1.6}$$

care se poate scrie în formă vectorial-matriceală:

$$u = \boldsymbol{w}\,\boldsymbol{x} + b\,, \tag{2.1.7}$$

unde $\boldsymbol{x}$ este *vectorul coloană* conținând intrările, $\boldsymbol{x} = [x_1 \ \ldots \ x_n]^T \in \mathbb{R}^n$, iar $\boldsymbol{w}$ este *vectorul linie* (adică un caz particular de matrice) conținând ponderile, $\boldsymbol{w} = [w_1 \ \ldots \ w_n] \in \mathbb{R}^{1\times n}$.

Această scriere ne arată că reprezentarea grafică compactă a neuronului cu mai multe intrări se poate realiza ca în figura 2.1.3. În respectiva figură, pentru semnalul de intrare $\boldsymbol{x} \in \mathbb{R}^n$ s-a utilizat convenția de reprezentarea grafică cu săgeată dublă (adică $\Rightarrow$), ceea ce permite compactarea reprezentării.

Observație

Precizăm că, începând din acest punct al textului nostru, vom face apel la convenția de reprezentarea grafică cu săgeată dublă ($\Rightarrow$) a semnalelor vectoriale fără a mai atrage atenția în mod special asupra semnificației, considerând că cititorul şi-a însuşit deja respectiva convenție. ■

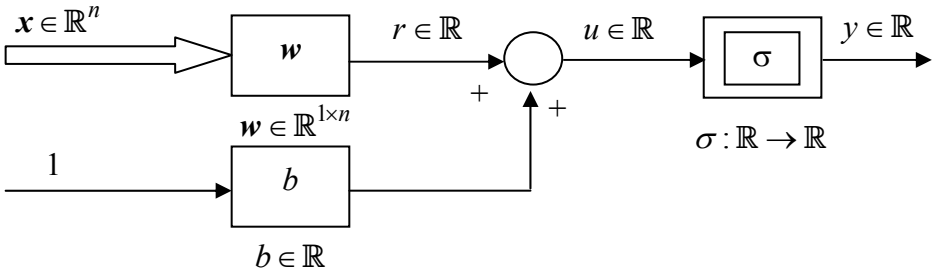


Figura 2.1.3. *Schema bloc a unui neuron cu n intrări*

Neuronul cu n intrări este un *sistem static* a cărui funcționare este descrisă prin *aplicația de variabilă vectorială*:

$$f : \mathbb{R}^n \to \mathbb{R}, \tag{2.1.8}$$

$$y = f(\boldsymbol{x}) = \sigma(\boldsymbol{w}\,\boldsymbol{x} + b), \tag{2.1.9}$$

ce rezultă din următoarea compunere de funcții (pusă în evidență de schema din figura 2.1.3):

$$y = f(\boldsymbol{x}) = \sigma(u(\boldsymbol{x})) \tag{2.1.10}$$

unde:

$$u(\boldsymbol{x}) = r(\boldsymbol{x}) + b, \tag{2.1.11}$$

$$r(\boldsymbol{x}) = \boldsymbol{w}\,\boldsymbol{x}. \tag{2.1.12}$$

Se observă că transferul $\boldsymbol{x} \to y$ este *parametrizat* de n+1 elemente: n elemente conținute în vectorul linie al ponderilor $\boldsymbol{w} \in \mathbb{R}^{1\times n}$ şi scalarul $b \in \mathbb{R}$ (deplasarea fiind unică pentru un neuron).

2.1.3. Alte tipuri de neuroni artificiali

Există tipuri de neuroni artificiali care diferă de structura neuronului artificial prezentată în fig. 2.1.2 (sau, echivalent, în fig. 2.13). Un exemplu îl constituie neuronul artificial cu funcție de activare de tip radial (eng. *Radial Basis Function*, abreviat RBF). Fără a furniza detalii referitoare la alte tipuri de neuroni artificiali, considerăm oportună prezența acestui paragraf pentru a asigura o vedere de ansamblu corectă asupra problematicii tratate în capitolul curent. Astfel, deşi modul de abordare vizează un grad mare de generalitate, capitolul de față nu îşi propune să acopere integral prezentarea structurilor neuronilor artificiali şi a rețelelor neurale feedforward.

Totodată mai precizăm că elementele de noutate ce vor apărea în viitoarele capitole vor fi grefate cu uşurință pe scheletul informațional construit în capitolul de față, ca o extindere naturală a manierei prin care se poate defini semnalul $u(\boldsymbol{x})$ aplicat drept intrare în nodul neuronului. Concret, în cazul altor tipuri de neuroni, transferul intrare-ieşire este descris tot printr-o relație de forma (2.1.10), cu deosebirea că expresia analitică a lui $u(\boldsymbol{x})$ nu mai este dată de (2.1.11) şi (2.1.12). Atragem însă atenția asupra faptului că, şi pentru aceste tipuri noi de neuroni artificiali, expresia analitică a lui $u(\boldsymbol{x})$ va rămâne parametrizată de w, b (existența parametrilor fiind cea care asigură proprietățile neuronului şi respectiv a unei rețelei compuse din mai mulți neuroni).

2.2. Rețele de neuroni artificiali

Prin conectarea în diverse moduri a neuronilor artificiali, se pot construi rețele neurale artificiale. Rețelele neurale cu arhitecturi specifice posedă proprietăți specifice, ceea ce face ca, în aplicații, unui anumit tip de problematică să i se asocieze un anumit tip de rețea neurală. Scopul acestei secțiuni constă în prezentarea generală a principiilor ce stau la baza organizării unei rețele neurale feedforward, urmând ca în capitolele viitoare să se detalieze asocierea dintre tipologia rețelelor şi specificitatea aplicațiilor.

2.2.1. Rețea cu un singur strat

În figura 2.2.1 este ilustrată schema bloc a unei rețele neurale cu un singur strat (eng. *layer*) conținând p neuroni, utilizând convenția de reprezentare grafică cu săgeți duble pentru semnale multivariabile. Rețeaua prezintă n intrări, notate compact $\boldsymbol{x} \in \mathbb{R}^n$, şi p ieşiri, notate compact $\boldsymbol{y} \in \mathbb{R}^p$:

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n, \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_p \end{bmatrix} \in \mathbb{R}^p. \tag{2.2.1}$$

Numărul de ieşiri ale rețelei coincide cu numărul de neuroni: fiecărui neuron îi corespunde o ieşire a stratului. Cei p neuroni din strat au funcții de activare proprii: neuronul i are funcția de activare σ_i.

Semnalele de intrare $x_1, \ldots, x_n$ sunt utilizate de toți cei p neuroni, dar cu combinații diferite de ponderi pentru fiecare neuron în parte. Funcția de activare a fiecărui neuron în parte are un singur semnal de intrare (semnal scalar) u_i corespunzător neuronului respectiv. (Se respectă arhitectura studiată a neuronului cu mai multe intrări unde s-a pus în evidență faptul că funcția de activare are un singur semnal de intrare). Pentru compactitatea desenului diagramei, funcțiile de activare a celor p neuroni sunt reprezentate unitar în blocul Σ având drept intrare semnalul multivariabil $\boldsymbol{u}$ şi drept ieşire semnalul multivariabil $\boldsymbol{y}$. În diagramă este specificat faptul că fiecare componentă a funcției vectoriale Σ acționează doar pe componenta corespunzătoare ei a semnalului de intrare $\boldsymbol{u}$, adică $\sigma_i(\boldsymbol{u}) = \sigma_i(u_i) = y_i$, $i = \overline{1, p}$.

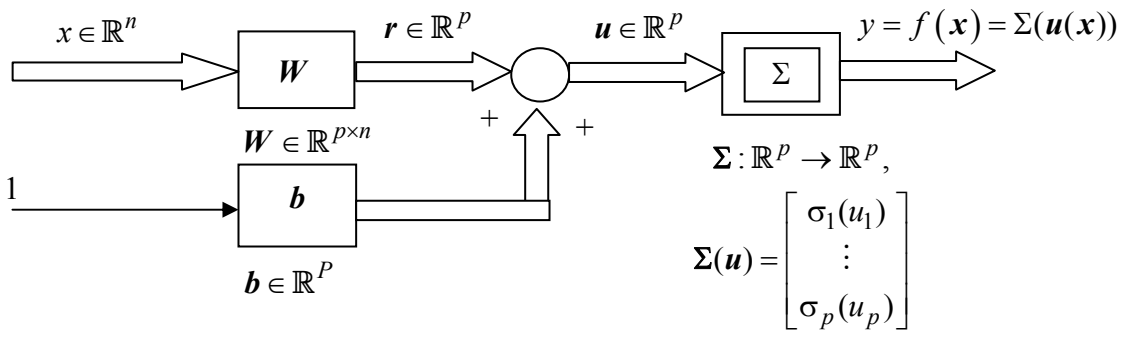


Figura 2.2.1. *Schema bloc a unei rețele neurale cu un singur strat de p neuroni*

Fiecare din cei p neuroni are în structură n+1 parametri (n parametri de tip pondere, care fac legătura cu cele n semnale de intrare x_1, x_2, ..., x_n, şi o deplasare). Cele n ponderi ale neuronului i, notate $w_{i,1}, w_{i,2}, \ldots, w_{i,n}$, constituie elementele liniei i a matricei $\boldsymbol{W} \in \mathbb{R}^{p \times n}$; deplasarea corespunzătoare aceluiaşi neuron, notată b_i, reprezintă componenta i a vectorului coloană $\boldsymbol{b} \in \mathbb{R}^p$.

Rețeaua neurală cu un singur strat este un *sistem static* a cărui funcționare este descrisă prin *aplicația vectorială, de variabilă vectorială*:

$$\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^p, \tag{2.2.2}$$

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) = \Sigma\left(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}\right), \tag{2.2.3}$$

ce rezultă din următoarea compunere de funcții (pusă în evidență de schema din figura 2.2.1)

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) = \Sigma\left(\boldsymbol{u}(\boldsymbol{x})\right), \tag{2.2.4}$$

unde:

$$\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{r}(\boldsymbol{x}) + \boldsymbol{b}, \tag{2.2.5}$$

$$\boldsymbol{r}(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x}. \tag{2.2.6}$$

Se observă că transferul $\boldsymbol{x} \to \boldsymbol{y}$ este *parametrizat* de cele pn elemente ale matricei $\boldsymbol{W} \in \mathbb{R}^{p \times n}$ şi cele p elemente ale vectorului $\boldsymbol{b} \in \mathbb{R}^p$.

Observație

În arhitecturile de rețele cu aplicabilitate practică toți neuronii au aceeaşi funcție de activare de tipul celor precizate în tabelul 2.1.1. Toate componentele funcției Σ sunt identice ca exprimare analitică, adică $y_i = \sigma_i(u_i) = \sigma(u_i)$, $i = \overline{1, p}$. ■

2.2.2. Rețea cu două straturi

În figura 2.2.2 se prezintă schema bloc a unei rețele neurale cu două straturi, conținând p_1 şi, respectiv, p_2 neuroni. Rețeaua posedă n_1 intrări, notate compact $\boldsymbol{x}^1 \in \mathbb{R}^{n_1}$, şi p_2 ieşiri, notate compact $\boldsymbol{y}^2 \in \mathbb{R}^{p_2}$:

$$\boldsymbol{x}^1 = \begin{bmatrix} x_1^1 \\ \vdots \\ x_{n_1}^1 \end{bmatrix} \in \mathbb{R}^{n_1}, \quad \boldsymbol{y}^2 = \begin{bmatrix} y_1^2 \\ \vdots \\ y_{p_2}^2 \end{bmatrix} \in \mathbb{R}^{p_2}. \tag{2.2.7}$$

Arhitectura acestei rețele rezultă prin înserierea a două rețele cu câte un singur strat fiecare (de tipul celor prezentate în paragraful 2.2.1), în aşa manieră încât ieşirile primului strat să constituie intrările celui de-al doilea strat.

Primul strat al rețelei mai este referit cu denumirea de *strat de intrare* (eng. *input layer*) iar al doilea cu denumirea de *strat de ieşire* (eng. *output layer*). Pentru conectarea în serie a celor două rețele se impune ca numărul ieşirilor primului strat (adică p_1) să coincidă

cu numărul intrărilor în cel de al doilea strat (adică n_2) deoarece semnalul vectorial de la ieşirea primului strat, $\boldsymbol{y}^1 \in \mathbb{R}^{p_1}$, constituie semnalul vectorial de intrare pentru al doilea strat, $\boldsymbol{x}^2 \in \mathbb{R}^{n_2}$. Astfel, are loc identitatea de semnale:

$$\boldsymbol{y}^1 (\in \mathbb{R}^{p_1}) \equiv \boldsymbol{x}^2 (\in \mathbb{R}^{n_2}) . \tag{2.2.8}$$

Numărul de intrări ale rețelei este numărul intrărilor primului strat, iar numărul de ieşiri ale rețelei este dat de numărul de neuroni din stratul doi.

Rețeaua neurală cu două straturi este un sistem static a cărui funcționare este descrisă prin *aplicația vectorială, de variabilă vectorială*:

$$\boldsymbol{f} : \mathbb{R}^{n_1} \to \mathbb{R}^{p_2} , \tag{2.2.9}$$

$$\boldsymbol{y}^2 = \boldsymbol{f}(\boldsymbol{x}^1) = \Sigma^2 \left(\boldsymbol{W}^2 \sigma^1 (\boldsymbol{W}^1 \boldsymbol{x}^1 + \boldsymbol{b}^1) + \boldsymbol{b}^2 \right). \tag{2.2.10}$$

Această aplicație rezultă din următoarea compunere de funcții (pusă în evidență de schema bloc din figura 2.2.2):

- *La nivelul celui de al doilea strat:*

$$\boldsymbol{y}^2 = \boldsymbol{f}^2 (\boldsymbol{x}^2) = \Sigma^2 \left(\boldsymbol{u}^2 (\boldsymbol{x}^2) \right), \tag{2.2.11}$$

unde:

$$\boldsymbol{u}^2 (\boldsymbol{x}^2) = \boldsymbol{r}^2 (\boldsymbol{x}^2) + \boldsymbol{b}^2 , \tag{2.2.12}$$

$$\boldsymbol{r}^2 (\boldsymbol{x}^2) = \boldsymbol{W}^2 \boldsymbol{x}^2 . \tag{2.2.13}$$

- *La nivelul primului strat:*

$$\boldsymbol{x}^2 = \boldsymbol{y}^1 = \boldsymbol{f}^1 (\boldsymbol{x}^1) = \Sigma^1 \left(\boldsymbol{u}^1 (\boldsymbol{x}^1) \right), \tag{2.2.14}$$

unde:

$$\boldsymbol{u}^1 (\boldsymbol{x}^1) = \boldsymbol{r}^1 (\boldsymbol{x}^1) + \boldsymbol{b}^1 , \tag{2.2.15}$$

$$\boldsymbol{r}^1 (\boldsymbol{x}^1) = \boldsymbol{W}^1 \boldsymbol{x}^1 . \tag{2.2.16}$$

Se observă că transferul $\boldsymbol{x}^1 \to \boldsymbol{y}^2$ realizat de rețeaua cu două straturi de neuroni este *parametrizat* de elementele matricelor $\boldsymbol{W}^1 \in \mathbb{R}^{p_1 \times n_1}$, $\boldsymbol{W}^2 \in \mathbb{R}^{p_2 \times n_2}$ (ponderile celor două straturi) şi elementele vectorilor $\boldsymbol{b}^1 \in \mathbb{R}^{p_1}$, $\boldsymbol{b}^2 \in \mathbb{R}^{p_2}$ (deplasările celor două straturi).

Observație

Funcțiile de activare de pe primul strat pot fi diferite de funcțiile de activare de pe stratul doi (dar, de regulă, pe acelaşi strat funcțiile de activare sunt identice).

În aplicațiile practice arhitecturile uzuale nu depăşesc trei straturi.

Ca terminologie rețeaua considerată în diagrama bloc conține, după unele texte, un strat de intrare şi unul de ieşire iar, după alte texte, un strat ascuns şi un strat de ieşire. ■

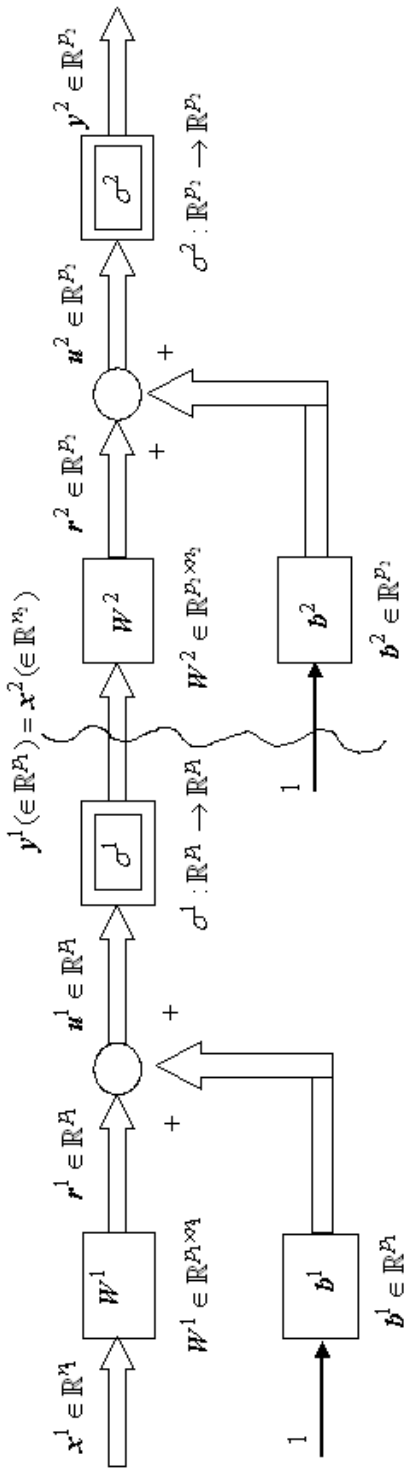


Figura 2.2.2. *Schema bloc a unei rețele neurale cu două straturi, conținând p_1 și, respectiv, p_2 neuroni*

2.2.3. Cazul general - rețea cu mai multe straturi

Într-o manieră similară celei prezentate în paragraful 2.2.2 se poate construi o rețea cu trei (sau mai multe) straturi de neuroni, conectând trei (sau mai multe) rețele cu un singur strat, astfel încât intrările fiecărui strat (exceptând primul) să coincidă cu ieşirile stratului care îl precede. Cele amintite mai înainte cu privire la terminologia prin care se referă straturile rețelei rămân valabile. Existența mai multor straturi are drept efect atât posibilitatea de a defini transferul intrare-ieşire cu ajutorul a mai mulți parametri, cât şi mai multe nivele de neliniarități care se compun în acest transfer. Asupra acestor aspecte se va reveni în detaliu atunci când se va discuta despre capacitatea de aproximare pe care o posedă unele rețele neurale. În practica uzuală se face apel la arhitecturi *cu cel mult trei straturi.*

Straturile de neuroni cuprinse între stratul de intrare şi cel de ieşire se numesc *straturi ascunse* (eng. *hidden*). În unele texte, orice strat al rețelei care nu este strat de ieşire este referit drept *strat ascuns* (inclusiv stratul de intrare). Ca urmare, exprimarea fără nici un fel de echivoc este cea care atribuie numere de ordine straturilor, astfel în cazul rețelei cu două straturi, stratul 1 desemnează stratul de intrare şi stratul 2 este stratul de ieşire.

2.3. Instrumente software pentru simularea rețelelor neurale cu structură feedforward oferite de *Neural Network Toolbox*

Neural Network Toolbox este unul dintre primele pachete de programe MATLAB lansate de The MathWorks, ceea ce demonstrează importanța arătată domeniului rețelelor neurale de către firma dezvoltatoare a acestui software. Versiunea 4.0 a toolbox-ului dedicat rețelelor neurale a fost lansată în anul 2000, împreună cu versiunea MATLAB Release 12. Odată cu dezvoltarea software-ului de bază (MATLAB), pachetului ***Neural Network Toolbox*** i-au fost aduse modificări minore, astfel că versiunea 5.0 a fost lansată împreună cu versiunea MATLAB R2006a.

Neural Network Toolbox admite lucrul cu diferite tipuri de rețele neurale. Această flexibilitate se datorează reprezentării orientate obiect a rețelelor, care permite definirea unor arhitecturi diverse şi implementarea algoritmilor specifici acestora. Un obiect de tip network constă din numeroase proprietăți ce pot fi setate în mod adecvat pentru specificarea arhitecturii şi comportării rețelei neurale şi poate fi creat cu funcția network. Pentru crearea unor rețele neurale cu o structură standard există însă funcții special concepute (de exemplu newp, newlin, newlind, newff) ce vor fi studiate în capitolele ce urmează.

Proprietățile unui obiect network sunt reunite, în funcție de tipul lor, în mai multe grupe, şi anume:

- architecture: proprietăți ce permit setarea numărului de intrări şi de straturi ale rețelei neurale, precum şi modul de conectare a acestora;
- functions: pentru definirea funcțiilor pentru operațiile de bază cu rețeaua neurală (inițializare, modul de adaptare a parametrilor, criteriul de performanță, funcția de antrenare, precum şi parametri specifici acestora);

- weight and bias values: pentru inspectarea sau setarea valorilor ponderilor şi deplasărilor neuronilor din rețea;
- other: pentru păstrarea altor date dorite de utilizator (de exemplu, comentarii).

În pachetul ***Neural Network Toolbox*** există şi o interfață grafică cu utilizatorul (*Graphical User Interface - GUI*), nntool, pentru importarea, crearea, utilizarea şi exportarea rețelelor neurale şi/sau datelor. De asemenea, ***Neural Network Toolbox*** pune la dispoziția utilizatorilor un set de blocuri pentru construirea rețelelor neurale în Simulink, precum şi funcția gensim pentru generarea versiunii Simulink a unei rețele neurale oarecare care a fost creată în MATLAB. Modul de utilizare a funcțiilor este ilustrat prin numeroase programe demonstrative ce pot fi lansate în execuție prin intermediul comenzii demos.

Pentru simularea comportării unei rețelei neurale se poate utiliza funcția sim.

Pentru aprofundarea noțiunilor prezentate în acest capitol se recomanda studierea problemelor cu soluții analitice 7.1, 7.2 şi a problemelor cu soluții numerice 8.1 – 8.4.

Capitolul 3

APLICAȚII ALE REȚELELOR FEED-FORWARD CU FUNCȚII DE ACTIVARE TREAPTĂ ÎN PROBLEME DE CLASIFICARE

Neuronul artificial cu funcție de activare treaptă reprezintă forma cea mai simplă a unei rețele neurale artificiale. El a fost introdus in 1943 de Warren McCulloch şi Walter Pitts în lucrarea „A Logical Calculus of Ideas Immanent in Nervous Activity", în încercarea de a modela activitatea unui neuron biologic. Neuronul McCulloch-Pitts acceptă doar intrări binare, furnizând la ieşire un semnal binar: ieşirea unui perceptron este 1 sau 0 după cum neuronul respectiv răspunde sau nu răspunde la semnalele de intrare aplicate. Toate intrările au ponderi egale.

Donald Hebb a revoluționat domeniul rețelelor neuronale artificiale în 1949. În monografia *The Organization of Behavior*, Hebb a propus un mecanism calitativ ce reflectă modul în care se modifică conexiunile sinaptice pentru a reflecta mecanismul de învățare realizat de neuronii interconectați atunci când aceştia sunt influențați de diferiți stimuli din mediu. Acest mecanism poartă numele de *regula lui Hebb* şi stă la baza proceselor de învățare (antrenare) a rețelelor neurale artificiale.

Aplicând acest principiu neuronului McCulloch-Pitts, Frank Rosenblatt a dezvoltat primul *perceptron* (1957) capabil să învețe prin modificarea ponderilor corespunzătoare intrărilor sale, modelând astfel capacitatea de (recunoaştere a formelor) clasificare a sistemelor de vedere biologice. Rosenblatt a demonstrat că dacă vectorii prototip utilizați pentru antrenarea unui perceptron fac parte din două clase liniar separabile, atunci algoritmul de antrenare converge într-un număr finit de paşi, furnizând un hiperplan ce separă cele două clase. Modelul propus în anul 1962 în cartea *Principles of Neurodynamics* a influențat decisiv dezvoltarea rețelelor neurale.

În prezent, rețelele neurale feed-forward cu *funcții de activare treaptă* poartă denumirea de *rețele perceptron*. Asemenea rețele pot fi utilizate numai în probleme de clasificare liniară şi necesită algoritmi de antrenare specifici, care exploatează forma particulară a funcțiilor de activare.

3.1. Studierea transferului intrare-ieşire pentru un neuron

Considerăm dat un perceptron cu n intrări (figura 3.1.1), având ponderile w_1, w_2, …, w_n, deplasarea b şi funcție de activare de tip treaptă unitate:

$$\sigma(u)=\begin{cases}0, & u<0,\\ 1, & u\geq 0.\end{cases} \tag{3.1.1}$$

Cu notațiile introduse în paragraful 2.1.2, şi anume $\boldsymbol{x}=\begin{bmatrix}x_1 & \dots & x_n\end{bmatrix}^T\in\mathbb{R}^n$ pentru vectorul coloană corespunzător intrărilor şi $\boldsymbol{w}=\begin{bmatrix}w_1 & \dots & w_n\end{bmatrix}\in\mathbb{R}^{1\times n}$ pentru vectorul linie conținând ponderile perceptronului, intrarea funcției de activare se scrie sub forma

$$u(\boldsymbol{x})=\boldsymbol{w}\,\boldsymbol{x}+b. \tag{3.1.2}$$

Ieşirea perceptronului este dată de

$$y(\boldsymbol{x})=\begin{cases}0, \text{ dacă } \boldsymbol{w}\,\boldsymbol{x}+b<0,\\ 1, \text{ dacă } \boldsymbol{w}\,\boldsymbol{x}+b\geq 0.\end{cases} \tag{3.1.3}$$

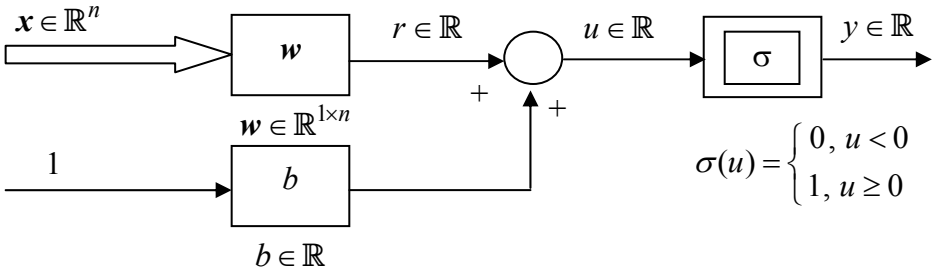


Figura 3.1.1. *Schema bloc a unui perceptron cu n intrări*

Se observă că spațiul de intrare, $\mathbb{R}^n$ poate fi separat în două clase în funcție de valorile ieşirii perceptronului: $\mathscr{C}_0=\left\{\boldsymbol{x}\in\mathbb{R}^n\,\middle|\,y(\boldsymbol{x})=0\right\}$ şi $\mathscr{C}_1=\left\{\boldsymbol{x}\in\mathbb{R}^n\,\middle|\,y(\boldsymbol{x})=1\right\}$. Suprafața de separație dintre cele două clase este hiperplanul de ecuație $u(\boldsymbol{x})=0$.

Problema inversă este următoarea: având date un număr N de *eşantioane* sau *prototipuri* (eng. *pattern*), $\boldsymbol{x}_i\in\mathbb{R}^n$, $i=1,\dots,N$, despre care *se presupune că* aparțin că aparțin la două clase, notate $\mathscr{C}_0$ şi $\mathscr{C}_1$, să se determine parametrii unui perceptron care să realizeze separarea dorită. Dacă această problemă admite soluție (dacă există un hiperplan care să separe punctele ce aparțin unei clase de punctele celeilalte clase) se spune că cele două clase sunt liniar separabile.

În cazuri simple, parametrii unui perceptron pot fi calculați în mod direct astfel încât să se realizeze separarea spațiului de intrare în modul dorit, aşa cum se prezintă în continuare.

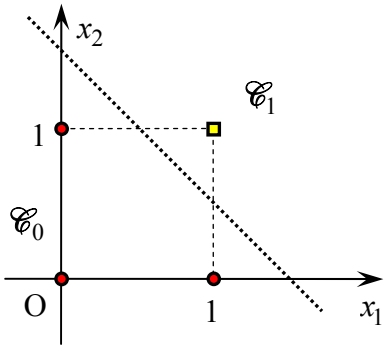
Exemplul 3.1.1. Se doreşte implementarea funcției logice AND utilizând un perceptron cu două intrări şi funcție de activare unipolară. Tabela de adevăr şi reprezentarea grafică a acestei funcții sunt prezentate în figura 3.1.2.

Se observă că dreapta de ecuație $d(\boldsymbol{x}) = x_1 + x_2 - 1.5 = 0$ realizează separarea liniară a celor patru puncte ale mulțimii $P = \{(x_1, x_2) | x_1, x_2 \in \{0, 1\}\}$ în funcție de rezultatul operației logice $x_1 \wedge x_2$. Prin identificare se obțin parametrii perceptronului $w_1 = w_2 = 1$ şi $b = -1.5$.

Evident că dreapta de separație a celor două clase $\mathscr{C}_0 = \{(x_1, x_2) \in P \mid x_1 \wedge x_2 = 0\}$ şi $\mathscr{C}_1 = \{(x_1, x_2) \in P \mid x_1 \wedge x_2 = 1\}$ nu este unică, ceea ce arată că implementarea funcției logice AND sub formă de perceptron nu este unică.

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

(a)



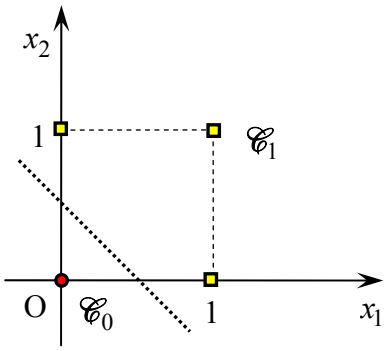
(b)

Figura 3.1.2. (a) *Tabela de adevăr şi* (b) *reprezentarea grafică a funcției logice AND.*

Similar se obțin parametrii unui perceptron cu două intrări şi funcție de activare unipolară ce implementează funcția logică OR (figura 3.1.3), cu valorile $w_1 = w_2 = 1$ şi $b = -0.5$.

x_1	x_2	$x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1

(a)



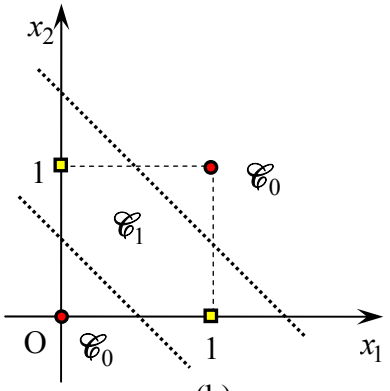
(b)

Figura 3.1.3. (a) *Tabela de adevăr şi* (b) *reprezentarea grafică a funcției logice OR.*

Pentru funcția logică XOR cu tabela de adevăr şi reprezentarea grafică din figura 3.1.4, clasele $\mathscr{C}_0$ şi $\mathscr{C}_1$ nu sunt liniar separabile, ceea ce arată că această funcție nu poate fi implementată utilizând o rețea neurală de tip perceptron cu un singur strat.

x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

(a)



(b)

Figura 3.1.4. (a) *Tabela de adevăr şi* (b) *reprezentarea grafică a funcției logice XOR.*

■

3.2. Utilizarea rețelelor cu un singur strat

3.2.1. Proprietatea de separare folosită în clasificare

O rețea neuronală cu un singur strat cu n intrări, $\boldsymbol{x} \in \mathbb{R}^n$, p ieşiri (neuroni) $\boldsymbol{y} \in \mathbb{R}^p$ şi funcții de activare treaptă (unipolară sau bipolară) posedă proprietatea de a partiționa spațiul vectorilor de intrare (adică $\mathbb{R}^n$) în 2^p mulțimi prin p *hiperplane*. O asemenea rețea va putea fi utilizată în probleme de clasificare *numai dacă* regiunile de decizie sunt mulțimi din $\mathbb{R}^n$ separabile liniar (adică cu ajutorul unor hiperplane).

3.2.2. Obiectivul antrenării rețelei

La intrarea rețelei sunt prezentate N *eşantioane*, sau *prototipuri*, notate $\boldsymbol{x}_i \in \mathbb{R}^n$, $i = 1,\ldots,N$, despre care *se presupune că* aparțin claselor $\mathscr{C}_j$, $j = 1,\ldots,2^p$, clase ce se pot defini ca regiuni separabile liniar din $\mathbb{R}^n$. Să introducem, pentru simplificarea scrierii ulterioare, notația $M = 2^p$. Evident problema clasificării are sens pentru $N > M$, altminteri situându-ne într-o situație pur banală. În termeni concreți, pentru fixare, vom considera situațiile următoare:

- m_1 vectori din $\mathbb{R}^n$ aparțin clasei $\mathscr{C}_1$, adică:

$$\boldsymbol{x}_k \in \mathscr{C}_1,\ k = 1,\ldots,m_1; \tag{3.2.1-1}$$

- m_2 vectori din $\mathbb{R}^n$ aparțin clasei $\mathscr{C}_2$, adică:

$$\boldsymbol{x}_k \in \mathscr{C}_2,\ k = m_1+1,\ldots,m_1+m_2; \tag{3.2.1-2}$$

…

- m_M vectori din $\mathbb{R}^n$ aparţin clasei $\mathscr{C}_M$, adică:

$$\boldsymbol{x}_k \in \mathscr{C}_M,\ k = m_1 + m_2 + \ldots + m_{M-1} + 1, \ldots, m_1 + m_2 + \ldots + m_{M-1} + m_M = N. \qquad (3.2.1\text{-}M)$$

Modalităţile de definire prin expresii analitice a celor $M = 2^p$ clase (adică ecuaţiile celor p hiperplane ce realizează separarea) *nu sunt cunoscute aprioric*. Determinarea sub formă analitică a acestor hiperplane reprezintă tocmai *obiectivul antrenării rețelei*, astfel încât, în urma unei antrenări adecvate, parametrii rețelei (ponderi şi deplasări) să furnizeze coeficienții ecuațiilor ce definesc hiperplanele de separare.

În acest scop, fiecăreia dintre cele 2^p clase $\mathscr{C}_j$ i se ataşează câte un vector $\underline{\boldsymbol{y}}_j \in \mathbb{R}^p$, $j = 1, \ldots, 2^p$, cu elemente binare, astfel:

- valori 0 şi 1 – în cazul când funcțiile de activare ale neuronilor sunt de tip treaptă unipolară;
- valori –1 şi 1 – în cazul când funcțiile de activare ale neuronilor sunt de tip treaptă bipolară.

Subliniem faptul că prin acest procedeu pot fi construiți exact 2^p vectori distincți, cu elemente binare, ceea ce permite realizarea unei corespondențe biunivoce între vectorii $\underline{\boldsymbol{y}}_j$ şi clasele $\mathscr{C}_j$. Astfel, problema clasificării revine la a antrena rețeaua (adică a determina parametrii rețelei) de aşa manieră încât *rețeaua cu parametrii rezultați din antrenare* să asigure satisfacerea următoarelor N egalități ce decurg din (1-1)...(1-M):

$$\boldsymbol{f}(\boldsymbol{x}_k) = \begin{cases} \underline{\boldsymbol{y}}_1, & k = 1, \ldots, m_1; \\ \underline{\boldsymbol{y}}_2, & k = m_1 + 1, \ldots, m_1 + m_2; \\ \ldots & \\ \underline{\boldsymbol{y}}_M, & k = m_1 + \ldots + m_{M-1} + 1, \ldots, N; \end{cases} \qquad (3.2.2)$$

Cu alte cuvinte, fiecărui vector prototip $\boldsymbol{x}_i$, $i = 1, \ldots, N$, i se poate ataşa un vector țintă (eng. *target*) $\boldsymbol{z}_i$, cu:

$$\boldsymbol{z}_i \in \left\{ \underline{\boldsymbol{y}}_1, \underline{\boldsymbol{y}}_2, \ldots, \underline{\boldsymbol{y}}_M \right\},\ i = 1, \ldots, N, \qquad (3.2.3)$$

astfel încât să fie satisfăcute egalitățile (3.2.2). Subliniem faptul că dintre cei N vectori țintă $\boldsymbol{z}_i$, $i = 1, \ldots, N$, numai $M < N$ sunt distincți (coincizând cu unul din vectorii $\underline{\boldsymbol{y}}_1, \underline{\boldsymbol{y}}_2, \ldots, \underline{\boldsymbol{y}}_M$). Astfel, prin antrenare, se va urmări ca:

$$\boldsymbol{f}(\boldsymbol{x}_i) = \boldsymbol{z}_i,\ i = 1, \ldots, N, \qquad (3.2.4)$$

unde forma particulară a lui $\boldsymbol{z}_i$ din mulțimea (3.2.3) este precizată prin egalitatea (3.2.2), corespunzător valorii concrete a indicelui i.

Observații

1° În formularea obiectivului antrenării s-a presupus că se operează cu clase separabile liniar. Totuşi, uzual, în practică, această ipoteză de separabilitate liniară *nu poate fi verificată*

înainte de a începe antrenarea. Altfel spus, nu dispunem de un procedeu care să ne asigure, *înainte de efectuarea antrenării*, că vectorii de intrare $\boldsymbol{x}_i$, $i=1,\ldots,N$, pot fi grupați cu satisfacerea condițiilor (3.2.1-1), …, (3.2.1-*M*), în $M=2^p$ clase $\mathscr{C}_j$, $j=1,\ldots,M$, separabile liniar. Astfel, *însuşi rezultatul antrenării* este cel care *confirmă* sau *infirmă* (prin verificarea satisfacerii celor N egalități (3.2.4)) ipoteza separabilității liniare a claselor, iar în caz de confirmare, sunt furnizate şi valorile numerice ale coeficienților ecuațiilor pentru hiperplanele de separare.

2° În cazul când rezultatul antrenării arată că cele N egalități (3.2.4) nu pot fi satisfăcute, nu înseamnă că vectorii $\boldsymbol{x}_i$, $i=1,\ldots,N$, nu *ar putea* fi grupați în $M=2^p$ *clase oarecare*, separabile liniar, ci semnifică faptul că *modul concret de asignare* a lui $\boldsymbol{x}_i$ la $\mathscr{C}_j$ definit prin relațiile (1-1),...,(1-*M*) *nu permite* separarea liniară a claselor. Deci, *un alt mod de definire* a relațiilor de apartenență (3.2.1-1), ..., (3.2.1-*M*) *ar putea* (nu în mod sigur) conduce la clase separabile liniar. ■

3.2.3. Algoritmul de antrenare al rețelei

Algoritmul de *antrenare*, *învățare*, sau *instruire* (eng. *training*, *learning*) constă în actualizarea parametrilor rețelei, adică a ponderilor (elementele matricei $\boldsymbol{W}\in\mathbb{R}^{p\times n}$) şi a deplasărilor (elementele vectorului coloană $\boldsymbol{b}\in\mathbb{R}^p$), cu scopul de a satisface cele N egalități (3.2.4).

Se organizează vectorii prototip $\boldsymbol{x}_i\in\mathbb{R}^n$, $i=1,\ldots,N$, şi cei țintă $\boldsymbol{z}_i\in\mathbb{R}^p$, $i=1,\ldots,N$, sub forma matricelor:

$$\boldsymbol{X}=[\boldsymbol{x}_1\ldots\boldsymbol{x}_N]\in\mathbb{R}^{n\times N}, \tag{3.2.5}$$

respectiv:

$$\boldsymbol{Z}=[\boldsymbol{z}_1\ldots\boldsymbol{z}_N]\in\mathbb{R}^{p\times N}. \tag{3.2.6}$$

Pentru algoritmul de antrenare se stabileşte de către utilizator un *număr maxim de iterații* sau *epoci*. La *fiecare iterație* a algoritmului de antrenare se parcurg etapele descrise mai jos. Valorile parametrilor rețelei la începerea unei iterații sunt notate prin $\boldsymbol{W}_{\text{old}}\in\mathbb{R}^{p\times n}$ (pentru ponderi) şi $\boldsymbol{b}_{\text{old}}\in\mathbb{R}^p$ (pentru deplasări).

Etapa 1 (Etapa de prezentare)

Pentru *valorile curente* ale parametrilor rețelei ($\boldsymbol{W}_{\text{old}}\in\mathbb{R}^{p\times n}$, $\boldsymbol{b}_{\text{old}}\in\mathbb{R}^p$) se calculează ieşirile rețelei:

$$\boldsymbol{y}_i=\boldsymbol{f}(\boldsymbol{x}_i)=\sigma\left(\boldsymbol{W}_{\text{old}}\cdot\boldsymbol{x}_i+\boldsymbol{b}_{\text{old}}\right),\ i=1,\ldots,N, \tag{3.2.7}$$

care se organizează sub forma matricei:

$$\boldsymbol{Y}=\left[\boldsymbol{y}_1\ldots\boldsymbol{y}_N\right]\in\mathbb{R}^{p\times N}. \tag{3.2.8}$$

Etapa 2 (Etapa de verificare)

Se calculează vectorii erorilor:

$$\boldsymbol{e}_i = \boldsymbol{z}_i - \boldsymbol{y}_i \in \mathbb{R}^p,\ i = 1,\ldots,N, \tag{3.2.9}$$

ca diferențe dintre vectorii țintă $\boldsymbol{z}_i$ (aparținând mulțimii (3.2.3)) şi vectorii ieşirii la iterația curentă $\boldsymbol{y}_i$ (definiți conform (3.2.7)).

Algoritmul se opreşte dacă este îndeplinită una din următoarele condiții:
($\chi 1$) toți vectorii eroare sunt nuli, adică:

$$\boldsymbol{e}_i \equiv \boldsymbol{0},\ i = 1,\ldots,N; \tag{3.2.10}$$

sau
($\chi 2$) a fost atins numărul maxim de iterații.

La oprire, algoritmul va furniza valorile parametrilor rețelei rezultate din antrenare, care vor fi notate prin $\boldsymbol{W}_f \in \mathbb{R}^{p\times n}$ (ponderi) şi $\boldsymbol{b}_f \in \mathbb{R}^p$ (deplasări).

Dacă nici una din condițiile ($\chi 1$) sau ($\chi 2$) nu este îndeplinită, se continuă cu etapa următoare a iterației curente.

Etapa 3 (Etapa de actualizare a parametrilor)

Se construieşte matricea erorilor:

$$\boldsymbol{E} = [\boldsymbol{e}_1 \ldots \boldsymbol{e}_N] = \boldsymbol{Z} - \boldsymbol{Y} \in \mathbb{R}^{p\times N}, \tag{3.2.11}$$

cu ajutorul căreia se calculează matricele de actualizare a parametrilor:

- pentru ponderi:

$$\boldsymbol{D}_W = \boldsymbol{E}\cdot\boldsymbol{X}^T \in \mathbb{R}^{p\times n}, \tag{3.2.12-a}$$

- pentru deplasări:

$$\boldsymbol{D}_b = \boldsymbol{E}\cdot \underbrace{[1\ldots 1]}_{N \text{ elemente}}^T \in \mathbb{R}^p. \tag{3.2.12-b}$$

Se calculează noile valori ale parametrilor:

- pentru ponderi:

$$\boldsymbol{W}_{\text{new}} = \boldsymbol{W}_{\text{old}} + \boldsymbol{D}_W, \tag{3.2.13-a}$$

- pentru deplasări:

$$\boldsymbol{b}_{\text{new}} = \boldsymbol{b}_{\text{old}} + \boldsymbol{D}_b. \tag{3.2.13-b}$$

Cu aceste calcule se încheie iterația curentă şi se trece la o nouă iterație efectuând atribuirile $\boldsymbol{W}_{\text{new}} \to \boldsymbol{W}_{\text{old}}$, $\boldsymbol{b}_{\text{new}} \to \boldsymbol{b}_{\text{old}}$.

Observații

1° Inițializarea parametrilor rețelei (adică $\boldsymbol{W}_{\text{old}}$ şi $\boldsymbol{b}_{\text{old}}$ la prima iterație a algoritmului) se face cu valori arbitrare.

2° Condiția ($\chi 1$) din Etapa 2 (anularea tuturor vectorilor eroare) poate fi exprimată unitar, prin intermediul *erorii pătratice globale* (eng. *sum squared error*) $J_{SSE} = \sum_{i=1}^{N} \boldsymbol{e}_i^T \boldsymbol{e}_i$, a

erorii pătratice medii (eng. *mean squared error*) $J_{MSE} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{e}_i^T \boldsymbol{e}_i$, sau a *erorii medii absolute* (eng. *mean absolute error*) $J_{MAE} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{p}\sum_{k=1}^{p} |\, \boldsymbol{e}_{i,k} \,|$, (unde $\boldsymbol{e}_{i,k}$ notează componenta k a vectorului eroare $\boldsymbol{e}_i = [\boldsymbol{e}_{i,1} \ldots \boldsymbol{e}_{i,p}]^T \in \mathbb{R}^p$) sub forma:

$$J_{SSE} = 0\,, \tag{3.2.14}$$

$$J_{MSE} = 0\,, \tag{3.2.14'}$$

respectiv:

$$J_{MAE} = 0\,, \tag{3.2.14"}$$

care sunt echivalente cu (3.2.10). Datorită simplității, în programare se preferă utilizarea unui test de forma (3.2.14), (3.2.14') sau (3.2.14"), față de cele N teste formulate în (3.2.10).

3° Este demonstrat faptul că vectorii $\boldsymbol{x}_i$, $i = 1,\ldots,N$, pot fi grupați în clasele C_j, separabile liniar, conform (1-1), (1-2), ... , (1-M), *dacă şi numai dacă* algoritmul de antrenare va conduce la satisfacerea condiției ($\chi 1$) într-un număr finit de iterații. Din acest motiv, la stabilirea numărului maxim de iterații, utilizatorul trebuie să aibă în vedere o valoare acoperitoare pentru satisfacerea condiției ($\chi 1$), evident, sub rezerva că ($\chi 1$) poate fi efectiv îndeplinită.

4° În cazul când condiția ($\chi 1$) este satisfăcută, valorile $\boldsymbol{W}_f \in \mathbb{R}^{p\times n}$ şi $\boldsymbol{b}_f \in \mathbb{R}^p$ furnizate în finalul algoritmului de antrenare definesc ecuațiile celor p hiperplane (care separă cele $M = 2^p$ clase) prin intermediul celor p componente scalare ale egalității:

$$\boldsymbol{W}_f \cdot \boldsymbol{x} + \boldsymbol{b}_f = \boldsymbol{0}\,. \tag{3.2.15}$$

5° Dacă condiția ($\chi 1$) nu poate fi îndeplinită într-un număr suficient de mare de iterații ale algoritmului de antrenare, rezultă că vectorii $\boldsymbol{x}_i$, $i = 1,\ldots,N$, nu pot fi grupați în clasele C_j separabile liniar, conform (1-1), (1-2), ... , (1-M). Este evident că în acest caz eroarea pătratică globală *nu se va anula*. ■

6° Rezultatul antrenării este independent de ordinea în care sunt considerați vectorii prototip. În secțiunea 3.2.2 a fost considerată ipoteza că vectorii prototip sunt ordonați în funcție de clasa din care fac parte numai datorită simplificării expunerii. ■

Exemplul 3.2.1. Ilustrăm aplicarea algoritmului de antrenare prezentat anterior în cazul determinării parametrilor unui perceptron ($p = 1$) cu două intrări ($n = 2$) ce implementează funcția logică OR (figura 3.1.3). Pentru aceasta considerăm vectorii prototip

$$\boldsymbol{x}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \boldsymbol{x}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \boldsymbol{x}_3 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \boldsymbol{x}_4 = \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

cu care formăm matricea

$$\boldsymbol{X} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}.$$

Matricea țintelor este

$$\boldsymbol{Z} = [0 \quad 1 \quad 1 \quad 1].$$

Inițializăm parametrii perceptronului cu valori nule:

$$\boldsymbol{W}_{\text{old}} = [0 \quad 0], \quad \boldsymbol{b}_{\text{old}} = 0.$$

Iteraţia 1: Se calculează:

$$\boldsymbol{Y} = [1 \quad 1 \quad 1 \quad 1], \ \boldsymbol{E} = [-1 \quad 0 \quad 0 \quad 0],$$
$$\boldsymbol{D}_W = [0 \quad 0], \quad \boldsymbol{D}_b = -1 \ \Rightarrow \ \boldsymbol{W}_{\text{new}} = [0 \quad 0], \quad \boldsymbol{b}_{\text{new}} = -1.$$

Iteraţia 2:

$$\boldsymbol{W}_{\text{old}} = [0 \quad 0], \quad \boldsymbol{b}_{\text{old}} = -1$$
$$\boldsymbol{Y} = [0 \quad 0 \quad 0 \quad 0], \ \boldsymbol{E} = [0 \quad 1 \quad 1 \quad 1],$$
$$\boldsymbol{D}_W = [2 \quad 2], \quad \boldsymbol{D}_b = 3 \ \Rightarrow \ \boldsymbol{W}_{\text{new}} = [2 \quad 2], \quad \boldsymbol{b}_{\text{new}} = 2.$$

Iteraţia 3:

$$\boldsymbol{W}_{\text{old}} = [2 \quad 2], \quad \boldsymbol{b}_{\text{old}} = 2$$
$$\boldsymbol{Y} = [1 \quad 1 \quad 1 \quad 1], \ \boldsymbol{E} = [-1 \quad 0 \quad 0 \quad 0]$$
$$\boldsymbol{D}_W = [0 \quad 0], \quad \boldsymbol{D}_b = -1 \ \Rightarrow \ \boldsymbol{W}_{\text{new}} = [2 \quad 2], \quad \boldsymbol{b}_{\text{new}} = 1.$$

Iteraţia 4

$$\boldsymbol{W}_{\text{old}} = [2 \quad 2], \quad \boldsymbol{b}_{\text{old}} = 1$$
$$\boldsymbol{Y} = [1 \quad 1 \quad 1 \quad 1], \ \boldsymbol{E} = [-1 \quad 0 \quad 0 \quad 0],$$
$$\boldsymbol{D}_W = [0 \quad 0], \quad \boldsymbol{D}_b = -1 \ \Rightarrow \ W_{new} = [2 \quad 2] \quad b_{new} = 0.$$

Iteraţia 5:

$$\boldsymbol{W}_{\text{old}} = [2 \quad 2], \quad \boldsymbol{b}_{\text{old}} = 0$$
$$\boldsymbol{Y} = [1 \quad 1 \quad 1 \quad 1], \ \boldsymbol{E} = [-1 \quad 0 \quad 0 \quad 0],$$
$$\boldsymbol{D}_W = [0 \quad 0], \quad \boldsymbol{D}_b = -1 \ \Rightarrow \ \boldsymbol{W}_{\text{new}} = [2 \quad 2], \quad \boldsymbol{b}_{\text{new}} = -1.$$

Iteraţia 6:

$$\boldsymbol{W}_{\text{old}} = [2 \quad 2], \quad \boldsymbol{b}_{\text{old}} = -1$$
$$\boldsymbol{Y} = [0 \quad 1 \quad 1 \quad 1], \ E = [0 \quad 0 \quad 0 \quad 0] \ \Rightarrow \ \text{STOP}$$

Condiția $\chi 1$ este satisfăcută. Algoritmul se încheie furnizând valorile parametrilor rețelei rezultate din antrenare, $\boldsymbol{W}_f = [2 \quad 2]$ şi $\boldsymbol{b} = -1$. Ecuația dreptei de separație este $d(\boldsymbol{x}) = 2x_1 + 2x_2 - 1 = 0$, fiind chiar cea reprezentată grafic în figura 3.1.3(b). ■

3.3. Utilizarea rețelelor cu mai multe straturi

Este demonstrat faptul că rețelele cu mai multe straturi permit realizarea clasificării în condițiile când clasele *nu constituie* regiuni separabile liniar din $\mathbb{R}^n$. În aceste situații, complexitatea rețelei (ca şi număr de straturi) este dependentă de proprietățile mulțimilor din $\mathbb{R}^n$ utilizate pentru clasificare (de exemplu, satisfacerea condiției de convexitate). Procedeele de antrenare constituie generalizări ale algoritmului discutat în cazul rețelelor cu un singur strat.

3.4. Facilități software oferite de *Neural Network Toolbox*

În pachetul ***Neural Network Toolbox*** există funcția newp crează o rețea perceptron cu un singur strat. După ce a fost creată, o rețea poate fi antrenată utilizând funcția train care implementează algoritmul de antrenare a unei rețele perceptron conform parametrilor de antrenare aleşi după creare. Pentru folosirea algoritmului de antrenare pe lot se alege rutina 'trainb' drept funcție de antrenare a rețelei (net.trainFcn = 'trainb'). Valoarea implicită este net.trainFcn = 'trainc'.

Pentru simularea comportării unei rețelei neurale create anterior se poate utiliza funcția sim.

Pentru reprezentarea grafică a vectorilor de intrare bi- şi tridimensionali se poate apela funcția plotpv. În scopul ataşării pe acest grafic a hiperplanelor de separare dintre clase se poate apela funcția plotpc (după un apel prealabil al funcției plotpv). Modul de apel al funcțiilor descrise mai sus poate fi studiat în urma consultării help-ului aferent.

Pentru aprofundarea noțiunilor prezentate în acest capitol se recomanda studierea problemelor cu soluții analitice 7.3 – 7.5 şi a problemelor cu soluții numerice 8.5 – 8.7.

Capitolul 4

APLICAȚII ALE REȚELELOR FEEDFORWARD CU FUNCȚII DE ACTIVARE LINIARE ÎN PROBLEME DE APROXIMARE

Rețelele neuronale feed-forward cu un singur strat cu funcții de activare liniare pot fi utilizate pentru a realiza aproximarea liniară a unor funcții definite cu ajutorul unei mulțimi de vectori prototip şi respectiv a unei mulțimi de vectori țintă. Arhitectura rețelei (adică numărul de neuroni, precum şi numărul de intrări) este impusă de dimensiunea vectorilor țintă şi respectiv dimensiunea vectorilor prototip.

Utilizarea în problemele de aproximare liniară a rețelelor neuronale feed-forward cu funcții de activare liniare se bazează pe antrenarea rețelelor prin algoritmul Widrow-Hoff. Aceste rețele mai poartă denumirea de ADALINE (ADAptive LINear Element), denumire propusă de Widrow şi Hoff pentru că algoritmul de antrenare are efectul adaptării (ADA) valorii parametrilor rețelei cu funcții liniare (LIN) în scopul aproximării.

4.1. Transferul liniar intrare-ieşire şi proprietatea de aproximare

Pentru o rețea feedforward cu un singur strat cu n intrări, p neuroni cu *funcții de activare liniare*, transferul intrare-ieşire este descris prin dependența liniară:

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}, \quad \boldsymbol{W} \in \mathbb{R}^{p \times n}, \ \boldsymbol{b} \in \mathbb{R}^{p}, \tag{4.1.1}$$

unde $\boldsymbol{x} \in \mathbb{R}^{n}$ notează vectorul de *intrare* iar $\boldsymbol{y} \in \mathbb{R}^{p}$ notează vectorul de *ieşire*. Acest transfer este parametrizat de ponderile $\boldsymbol{W} \in \mathbb{R}^{p \times n}$ şi deplasările $\boldsymbol{b} \in \mathbb{R}^{p}$.

Se consideră o mulțime de N *vectori prototip*:

$$\boldsymbol{x}_1, \ \boldsymbol{x}_2, \ \ldots, \ \boldsymbol{x}_N \in \mathbb{R}^{n}, \tag{4.1.2}$$

şi o mulțime de N *vectori țintă*:

$$\boldsymbol{z}_1 , \boldsymbol{z}_2 , ..., \boldsymbol{z}_N \in \mathbb{R}^p , \tag{4.1.3}$$

care corespund vectorilor prototip în sensul că există o aplicație:

$$\varphi : \mathbb{R}^n \to \mathbb{R}^p , \tag{4.1.4}$$

care asigură:

$$\varphi (\boldsymbol{x}_i) = \boldsymbol{z}_i , \ i = 1,..., N . \tag{4.1.5}$$

Aplicând funcția (4.1.1) pentru cei N vectori prototip $\boldsymbol{x}_i$, putem scrie:

$$\begin{aligned} \boldsymbol{y}_1 &= \boldsymbol{W}\boldsymbol{x}_1 + \boldsymbol{b} = \begin{bmatrix} \boldsymbol{W} & \boldsymbol{b} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ 1 \end{bmatrix} = \Theta \underline{\boldsymbol{x}}_1 , \\ &\vdots \\ \boldsymbol{y}_N &= \boldsymbol{W}\boldsymbol{x}_N + \boldsymbol{b} = \begin{bmatrix} \boldsymbol{W} & \boldsymbol{b} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_N \\ 1 \end{bmatrix} = \Theta \underline{\boldsymbol{x}}_N , \end{aligned} \tag{4.1.6}$$

unde s-au folosit notațiile:

$$\Theta = \begin{bmatrix} \boldsymbol{W} & \boldsymbol{b} \end{bmatrix} \in \mathbb{R}^{p \times (n+1)} , \tag{4.1.7}$$

$$\underline{\boldsymbol{x}}_i = \begin{bmatrix} \boldsymbol{x}_i \\ 1 \end{bmatrix} \in \mathbb{R}^{n+1} , \quad i = 1,..., \boldsymbol{N} . \tag{4.1.8}$$

Partajând pe linii matricea parametrilor $\Theta \in \mathbb{R}^{p \times (n+1)}$ sub forma:

$$\Theta = \begin{bmatrix} \boldsymbol{\theta}_1^T \\ \vdots \\ \boldsymbol{\theta}_p^T \end{bmatrix}, \quad \boldsymbol{\theta}_k \in \mathbb{R}^{n+1} , \ k = 1, \ldots , p , \tag{4.1.9}$$

se constată că relația ce corespunde unui indice oarecare i în (4.1.6) poate fi rescrisă astfel:

$$\boldsymbol{y}_i = \begin{bmatrix} \boldsymbol{\theta}_1^T \underline{\boldsymbol{x}}_i \\ \vdots \\ \boldsymbol{\theta}_p^T \underline{\boldsymbol{x}}_i \end{bmatrix} = \begin{bmatrix} \underline{\boldsymbol{x}}_i^T \boldsymbol{\theta}_1 \\ \vdots \\ \underline{\boldsymbol{x}}_i^T \boldsymbol{\theta}_p \end{bmatrix} = \boldsymbol{A}_i \boldsymbol{v}, \quad i = 1,..., N . \tag{4.1.10}$$

În relația (4.1.10) matricea $\boldsymbol{A}_i$ este definită prin:

$$\boldsymbol{A}_i = \begin{bmatrix} \underline{\boldsymbol{x}}_i^{\,T} & \boldsymbol{O}_{1\times(n+1)} & \cdots & \boldsymbol{O}_{1\times(n+1)} \\ \boldsymbol{O}_{1\times(n+1)} & \underline{\boldsymbol{x}}_i^{\,T} & \cdots & \boldsymbol{O}_{1\times(n+1)} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{O}_{1\times(n+1)} & \boldsymbol{O}_{1\times(n+1)} & \cdots & \underline{\boldsymbol{x}}_i^{\,T} \end{bmatrix} \in \mathbb{R}^{p \times p(n+1)}, \quad i = 1,..., N , \tag{4.1.11}$$

iar $\boldsymbol{v} \in \mathbb{R}^{p(n+1)}$ este vectorul tuturor parametrilor rețelei:

$$\boldsymbol{v} = \begin{bmatrix} \boldsymbol{\theta}_1 \\ \vdots \\ \boldsymbol{\theta}_p \end{bmatrix} \in \mathbb{R}^{p(n+1)} . \tag{4.1.12}$$

Într-o scriere globală, cele N relații (4.1.10) (toate conținând același vector al parametrilor rețelei, $\boldsymbol{v} \in \mathbb{R}^{p(n+1)}$) conduc la relația vectorială (compusă din pN egalități scalare):

$$\begin{bmatrix} \boldsymbol{y}_1 \\ \vdots \\ \boldsymbol{y}_N \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_1 \\ \vdots \\ \boldsymbol{A}_N \end{bmatrix} \boldsymbol{v} . \tag{4.1.13}$$

Se observă imediat că (4.1.13) descrie comportarea rețelei (pentru *valori arbitrare* ale parametrilor rețelei) în cazul când la intrare se prezintă vectorii prototip $\boldsymbol{x}_i \in \mathbb{R}^n$, $i = 1,\ldots,N$.

Dacă în (4.1.13), vectorii $\boldsymbol{y}_i \in \mathbb{R}^p$, $i = 1,\ldots,N$, se înlocuiesc cu vectorii țintă $\boldsymbol{z}_i \in \mathbb{R}^p$, $i = 1,\ldots,N$, şi dacă se schimbă membrul stâng cu cel drept, se obține relația:

$$\begin{bmatrix} \boldsymbol{A}_1 \\ \vdots \\ \boldsymbol{A}_N \end{bmatrix} \boldsymbol{v} = \begin{bmatrix} \boldsymbol{z}_1 \\ \vdots \\ \boldsymbol{z}_N \end{bmatrix} . \tag{4.1.14}$$

Această relație poate fi privită ca un sistem algebric liniar de pN ecuații cu $p(n+1)$ necunoscute (necunoscutele fiind parametrii rețelei, grupați în vectorul $\boldsymbol{v} \in \mathbb{R}^{p(n+1)}$. În sistemul respectiv, matricea coeficienților se construieşte din elementele vectorilor prototip $\boldsymbol{x}_i \in \mathbb{R}^n$, $i = 1,\ldots,N$ (conform celor N egalități (4.1.11)), iar vectorul termenilor liberi se construieşte din elementele vectorilor țintă $\boldsymbol{z}_i \in \mathbb{R}^p$, $i = 1,\ldots,N$.

După cum este prezentat în secțiunea A.1.2 din Appendix, sistemul (4.1.14) posedă întotdeauna o *soluție în sensul celor mai mici pătrate* (unică sau nu), care se va nota prin $\boldsymbol{v}^* \in \mathbb{R}^{p(n+1)}$.

Pentru un set dat de vectori prototip, $\boldsymbol{x}_i \in \mathbb{R}^n$, $i = 1,\ldots,N$, relația (4.1.10) poate fi privită ca exprimând dependența ieşirii rețelei neurale de parametrii acesteia, $\boldsymbol{y}_i = \boldsymbol{y}_i(\boldsymbol{v})$, $i = 1,\ldots,N$. Utilizând vectorii de eroare:

$$\boldsymbol{e}_i(\boldsymbol{v}) = \boldsymbol{z}_i - \boldsymbol{y}_i(\boldsymbol{v}) \in \mathbb{R}^p, \; i = 1,...,N , \tag{4.1.15}$$

se calculează *eroarea pătratică globală* (*Sum Squared Error* – SSE):

$$J_{SSE}(\boldsymbol{v}) = \sum_{i=1}^{N} [\boldsymbol{e}_i(\boldsymbol{v})]^T \boldsymbol{e}_i(\boldsymbol{v}) , \tag{4.1.16}$$

şi *eroarea pătratică medie* (*Mean Squared Error* – MSE):

$$J_{MSE}(\boldsymbol{v}) = \frac{1}{N} \sum_{i=1}^{N} [\boldsymbol{e}_i(\boldsymbol{v})]^T \boldsymbol{e}_i(\boldsymbol{v}) . \tag{4.1.16'}$$

Valoarea parametrilor rețelei definiți prin cele $p(n+1)$ elemente ale vectorului $\boldsymbol{v}^* \in \mathbb{R}^{p(n+1)}$ asigură *minimizarea MSE* (echivalentă cu *minimizarea SSE*) pe întreg spațiul $\mathbb{R}^{p(n+1)}$,

$$0 \le J(\boldsymbol{v}^*) \le J(\boldsymbol{v}), \quad \forall \boldsymbol{v} \in \mathbb{R}^{p(n+1)} , \tag{4.1.17}$$

(unde J notează una intre funcțiile J_{SSE} şi J_{MSE}) fiind notat: $\boldsymbol{v}^* = \arg\min_{\boldsymbol{v} \in \mathbb{R}^{p(n+1)}} J(\boldsymbol{v})$.

Vectorului $\boldsymbol{v}^* \in \mathbb{R}^{p(n+1)}$ îi corespund matricele $\boldsymbol{W}^* \in \mathbb{R}^{p\times n}$ şi $\boldsymbol{b}^* \in \mathbb{R}^p$ conținând valorile parametrilor rețelei (vezi relațiile (4.1.7), (4.1.9) şi (4.1.12)). Astfel, pentru aceste valori $\boldsymbol{W}^*$ şi $\boldsymbol{b}^*$, rețeaua va furniza prin funcția $\boldsymbol{f}$ din (4.1.1) o aproximare liniară, în sensul celor mai mici pătrate, a funcției φ (4.1.5) (care definește corespondența dintre vectorii prototip şi țintă).

Observație:

Cu notațiile unde $\tilde{\boldsymbol{A}} = \left[\boldsymbol{A}_1^T \ldots \boldsymbol{A}_N^T\right]^T$ şi $\tilde{\boldsymbol{z}}^T = \left[\boldsymbol{z}_1^T \ldots \boldsymbol{z}_N^T\right]^T$, sistemul de ecuații algebrice (4.1.14) poate fi adus la forma $\tilde{\boldsymbol{A}}\boldsymbol{v} = \tilde{\boldsymbol{z}}$. Rezolvarea acestui sistem comportă următoarea discuție, care decurge din Teorema Kronecker-Capelli:

1° Dacă:

$$\operatorname{rang}\begin{bmatrix}\boldsymbol{A}_1\\ \vdots \\ \boldsymbol{A}_N\end{bmatrix} = \operatorname{rang}\left[\begin{array}{c|c}\boldsymbol{A}_1 & \boldsymbol{z}_1\\ \vdots & \vdots \\ \boldsymbol{A}_N & \boldsymbol{z}_N\end{array}\right] < \min\left(pN, p(n+1)\right), \tag{4.1.18}$$

atunci sistemul (4.1.14) posedă o *infinitate de soluții exacte* $\boldsymbol{v}^* \in \mathbb{R}^{p(n+1)}$, pentru care vectorii eroare $\boldsymbol{e}_i$, $i = 1,\ldots,N$, definiți prin (4.1.15) se anulează şi implicit, rezultă:

$$J(\boldsymbol{v}^*) = 0. \tag{4.1.19}$$

2° Dacă:

$$\operatorname{rang}\begin{bmatrix}\boldsymbol{A}_1\\ \vdots \\ \boldsymbol{A}_N\end{bmatrix} = \operatorname{rang}\left[\begin{array}{c|c}\boldsymbol{A}_1 & \boldsymbol{z}_1\\ \vdots & \vdots \\ \boldsymbol{A}_N & \boldsymbol{z}_N\end{array}\right] = \min\left(pN, p(n+1)\right), \tag{4.1.20}$$

atunci sistemul (4.1.14) posedă *o singură soluție exactă* $\boldsymbol{v}^* = \tilde{\boldsymbol{A}}^{-1}\tilde{\boldsymbol{z}}$, pentru care vectorii eroare $\boldsymbol{e}_i$, $i = 1,\ldots,N$, definiți prin (4.1.15) se anulează şi, implicit, rezultă satisfacerea egalității (4.1.19).

3° Dacă:

$$\operatorname{rang}\begin{bmatrix}\boldsymbol{A}_1\\ \vdots \\ \boldsymbol{A}_N\end{bmatrix} < \operatorname{rang}\left[\begin{array}{c:c}\boldsymbol{A}_1 & \boldsymbol{z}_1\\ \vdots & \vdots \\ \boldsymbol{A}_N & \boldsymbol{z}_N\end{array}\right], \tag{4.1.21}$$

atunci sistemul (14) posedă numai *pseudo-soluții* $\boldsymbol{v}^*$, sau *soluții în sensul celor mai mici pătrate* conform (17), pentru care avem inegalitatea *strictă*:

$$J(\boldsymbol{v}^*) > 0. \tag{4.1.22}$$

Se disting următoarele două subcazuri:

a) Dacă

$$\operatorname{rang}\begin{bmatrix} \boldsymbol{A}_1 \\ \vdots \\ \boldsymbol{A}_N \end{bmatrix} < \min\left(pN, p(n+1)\right), \tag{4.1.23}$$

atunci există *o infinitate de pseudo-soluții* satisfăcând (4.1.17). Dintre acestea, soluția

$$\boldsymbol{v}^* = \left(\tilde{\boldsymbol{A}}^T \tilde{\boldsymbol{A}}\right)^{-1} \tilde{\boldsymbol{A}}^T \tilde{\boldsymbol{z}}\,, \tag{4.1.24}$$

este are norma minimă.

b) Dacă

$$\operatorname{rang}\begin{bmatrix} \boldsymbol{A}_1 \\ \vdots \\ \boldsymbol{A}_N \end{bmatrix} = \min\left(pN, p(n+1)\right), \tag{4.1.25}$$

atunci există *o singură pseudo-soluție* $\boldsymbol{v}^*$ satisfăcând (4.1.17), şi anume cea dată de (4.1.24).

Această discuție formulată cu privire la vectorul $\boldsymbol{v}^* \in \mathbb{R}^{p(n+1)}$, *se transpune direct în termenii parametrilor rețelei* $\boldsymbol{W}^* \in \mathbb{R}^{p\times n}$ (ponderi) şi $\boldsymbol{b}^* \in \mathbb{R}^p$ (deplasări), parametri care constituie elementele vectorului $\boldsymbol{v}^* \in \mathbb{R}^{p(n+1)}$. ■

4.2. Utilizarea rețelelor cu un singur strat

4.2.1. Obiectivul antrenării rețelei

Pornind de la *mulțimea vectorilor prototip* (4.1.2) şi *mulțimea vectorilor țintă* (4.1.3), prin antrenarea rețelei trebuie să se determine parametrii acesteia $\boldsymbol{W}^* \in \mathbb{R}^{p\times n}$ şi $\boldsymbol{b}^* \in \mathbb{R}^p$ care să asigure *minimizarea erorii pătratice globale*, adică satisfacerea inegalității (16). Funcția $\boldsymbol{f}$ definită prin (4.1.1) pentru $\boldsymbol{W} = \boldsymbol{W}^*$ şi $\boldsymbol{b} = \boldsymbol{b}^*$ realizează o *aproximare liniară în sensul celor mai mici pătrate* a funcției φ definită conform (4.1.5).

4.2.2. Algoritmul de antrenare Widrow-Hoff

În baza celor discutate în paragraful 4.1, parametrii rețelei $\boldsymbol{W}^* \in \mathbb{R}^{p\times n}$ şi $\boldsymbol{b}^* \in \mathbb{R}^p$ pot fi determinați prin rezolvarea sistemului algebric (4.1.14) cu pN ecuații şi $p(n+1)$ necunoscute. Totuşi, din punct de vedere numeric, acest procedeu nu este recomandabil întrucât acuratețea soluției depinde de condiționarea numerică a sistemului (4.1.14). Din acest motiv, se preferă pentru antrenare, *algoritmul iterativ Widrow-Hoff* (denumit şi *regula delta*), care se bazează pe *minimizarea erorii pătratice* globale prin *metoda gradientului*.

Considerând vectorii eroare $\boldsymbol{e}_i$, $i = 1,\ldots,N$, definiți prin (4.1.15), ca funcții de parametri rețelei (care sunt conținuți în vectorul $\boldsymbol{v} \in \mathbb{R}^{p(n+1)}$) se formează funcția obiectiv MSE $J(\boldsymbol{v})$ dată de (4.1.16').

Minimizarea prin metoda gradientului a lui $J(\boldsymbol{v})$ (care înseamnă satisfacerea condiției (4.1.17)) constă în modificarea vectorului $\boldsymbol{v} \in \mathbb{R}^{p(n+1)}$ la fiecare iterație, în conformitate cu formula:

$$\boldsymbol{v}_{\text{new}} = \boldsymbol{v}_{\text{old}} - \eta \nabla J(\boldsymbol{v}_{\text{old}}), \tag{4.2.1}$$

unde $\eta > 0$ notează *pasul metodei de optimizare*, iar $\nabla J = \left[\partial J / \partial \boldsymbol{v}\right]^T$ notează vectorul gradient, normal la suprafața $J(\boldsymbol{v})$ definită prin (4.1.16').

Se observă că $J(\boldsymbol{v})$ este o formă pătratică definită pe spațiul parametrilor, $\mathbb{R}^{p(n+1)}$:

$$\begin{aligned} J(\boldsymbol{v}) &= \frac{1}{N}\sum_{i=1}^{N}[\boldsymbol{e}_i(\boldsymbol{v})]^T \boldsymbol{e}_i(\boldsymbol{v}) = \frac{1}{N}\sum_{i=1}^{N}\left[\boldsymbol{z}_i - \boldsymbol{A}_i\boldsymbol{v}\right]^T\left[\boldsymbol{z}_i - \boldsymbol{A}_i\boldsymbol{v}\right] = \\ &= \frac{1}{N}\sum_{i=1}^{N}\left[\boldsymbol{z}_i^T\boldsymbol{z}_i - 2\boldsymbol{z}_i^T\boldsymbol{A}_i\boldsymbol{v} + \boldsymbol{v}^T\boldsymbol{A}_i^T\boldsymbol{A}_i\boldsymbol{v}\right] = c + \boldsymbol{d}^T\boldsymbol{v} + \frac{1}{2}\boldsymbol{v}^T\boldsymbol{R}\boldsymbol{v}, \end{aligned} \tag{4.2.2}$$

unde: $\boldsymbol{R} = 2\frac{1}{N}\sum_{i=1}^{N}\boldsymbol{A}_i^T\boldsymbol{A}_i = \frac{2}{N}\tilde{\boldsymbol{A}}^T\tilde{\boldsymbol{A}} \in \mathbb{R}^{p(n+1)\times p(n+1)}$, $\boldsymbol{d} = -2\frac{1}{N}\sum_{i=1}^{N}\boldsymbol{A}_i^T\boldsymbol{z}_i = -\frac{2}{N}\tilde{\boldsymbol{A}}^T\tilde{\boldsymbol{z}} \in \mathbb{R}^{p(n+1)}$,

$c = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{z}_i^T\boldsymbol{z}_i = \frac{1}{N}\tilde{\boldsymbol{z}}^T\tilde{\boldsymbol{z}} \in \mathbb{R}$, astfel că $\nabla J(\boldsymbol{v}) = \boldsymbol{d} + \boldsymbol{R}\boldsymbol{v}$ şi $D^2 J(\boldsymbol{v}) = \boldsymbol{R}$ (matricea Hessian).

Printr-o exprimare adecvată a vectorului gradient $\nabla J(\boldsymbol{v})$, relația (4.2.1) poate fi formulată *direct în termenii parametrilor rețelei* $\boldsymbol{W} \in \mathbb{R}^{p\times n}$ (ponderi) şi $\boldsymbol{b} \in \mathbb{R}^p$ (deplasări), furnizând, astfel, *mecanismul de efectuare a unei iterații* pentru *algoritmul de antrenare Widrow-Hoff*.

Pentru algoritmul de antrenare se stabilesc de către utilizator un *număr maxim de iterații* sau *epoci*, precum şi o valoare ε care se dorește a fi atinsă în minimizarea funcției obiectiv (4.1.16') (adică un *prag pentru eroarea pătratică globală*). Atragem atenția asupra faptului că funcția obiectiv $J(\boldsymbol{v})$ definită prin (4.1.16') satisface condiția (4.1.19), dacă şi numai dacă sistemul de ecuații algebrice liniare (4.1.14) posedă o soluție exactă (sau soluții exacte). În cazul când relația (4.1.19) nu este satisfăcută, atingerea unui prag *oarecare* de eroare ε impus de utilizator *nu este întotdeauna posibilă* (întrucât ne situăm în situația caracterizată prin (4.1.22)). Facem observația că utilizând funcția de tip MSE, valoarea de prag poate fi aleasă independent de numărul de eşantioane cu care se lucrează.

De asemenea, se fixează o valoare pentru pasul metodei de optimizare (η), care, în contextul problemei de antrenare, este referită sub denumirea de *rată de învățare* (eng. *learning rate*).

Se organizează vectorii prototip $\boldsymbol{x}_i \in \mathbb{R}^n$, $i = 1,\ldots,N$, şi cei țintă $\boldsymbol{z}_i \in \mathbb{R}^p$, $i = 1,\ldots,N$, sub forma matricelor:

$$\boldsymbol{X} = [\boldsymbol{x}_1 \ldots \boldsymbol{x}_N] \in \mathbb{R}^{n\times N}, \tag{4.2.3}$$

respectiv:

$$\boldsymbol{Z} = [\boldsymbol{z}_1 \ldots \boldsymbol{z}_N] \in \mathbb{R}^{p\times N}. \tag{4.2.4}$$

La *fiecare iterație* a algoritmului Widrow-Hoff se parcurg etapele descrise mai jos.

Valorile parametrilor rețelei la începutul unei iterații sunt notate prin $\boldsymbol{W}_{\text{old}} \in \mathbb{R}^{p\times n}$ (pentru ponderi) şi $\boldsymbol{b}_{\text{old}} \in \mathbb{R}^{p}$ (pentru deplasări).

Etapa I (Etapa de prezentare)

Pentru valorile curente ale parametrilor rețelei ($\boldsymbol{W}_{\text{old}} \in \mathbb{R}^{p\times n}$, $\boldsymbol{b}_{\text{old}} \in \mathbb{R}^{p}$) se calculează ieşirile rețelei:

$$\boldsymbol{y}_i = \boldsymbol{f}(\boldsymbol{x}_i) = \boldsymbol{W}_{\text{old}}\boldsymbol{x}_i + \boldsymbol{b}_{\text{old}},\ \ \boldsymbol{y}_i \in \mathbb{R}^{p},\ i = 1,\ldots,N, \tag{4.2.5}$$

care se organizează sub forma matricei:

$$\boldsymbol{Y} = [\boldsymbol{y}_1 \ldots \boldsymbol{y}_N] = \boldsymbol{W}_{\text{old}}\boldsymbol{X} + \boldsymbol{b}_{\text{old}} \in \mathbb{R}^{p\times N}. \tag{4.2.6}$$

Etapa II (Etapa de verificare)

Se calculează vectorii eroare $\boldsymbol{e}_i = \boldsymbol{e}_i(\boldsymbol{W}_{\text{old}}, \boldsymbol{b}_{\text{old}})$, $i = 1,\ldots,N$, definiți prin (4.1.15) ca diferența între vectorii țintă $\boldsymbol{z}_i \in \mathbb{R}^{p}$, $i = 1,\ldots,N$, şi vectorii ieşire la iterația curentă $\boldsymbol{y}_i = \boldsymbol{y}_i \in \mathbb{R}^{p}$, $i = 1,\ldots,N$.

Algoritmul se opreşte dacă este îndeplinită una din următoarele condiții:

($\chi 1$) eroarea pătratică globală este inferioară valorii de prag (eng. *threshold*) ε stabilite la startarea algoritmului, adică:

$$J(\boldsymbol{W}_{\text{old}}, \boldsymbol{b}_{\text{old}}) = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{e}_i^{T}\boldsymbol{e}_i < \varepsilon; \tag{4.2.7}$$

sau:

($\chi 2$) a fost atins numărul maxim de iterații stabilit la startarea algoritmului.

La oprire, algoritmul va furniza valorile parametrilor rețelei rezultate din antrenare, care vor fi notați prin $\boldsymbol{W}_f \in \mathbb{R}^{p\times n}$ (ponderi) şi $\boldsymbol{b}_f \in \mathbb{R}^{p}$ (deplasări).

Dacă nici una din condițiile ($\chi 1$) sau ($\chi 2$) nu este îndeplinită, se continuă cu etapa următoare a iterației curente.

Etapa III (Etapa de actualizare a parametrilor)

Se construieşte matricea erorilor:

$$\boldsymbol{E} = [\boldsymbol{e}_1 \ldots \boldsymbol{e}_N] = \boldsymbol{Z} - \boldsymbol{Y} \in \mathbb{R}^{p\times N}, \tag{4.2.8}$$

cu ajutorul căreia se calculează matricele de actualizare a parametrilor:

- pentru *ponderi*:

$$\boldsymbol{D}_W = \boldsymbol{E}\cdot\boldsymbol{X}^{T} \in \mathbb{R}^{p\times n}, \tag{4.2.9-a}$$

- pentru *deplasări*:

$$\boldsymbol{D}_b = \boldsymbol{E}\cdot \underbrace{[1 \ldots 1]^{T}}_{N \text{ elemente}} \in \mathbb{R}^{p}. \tag{4.2.9-b}$$

Se calculează noile valori ale parametrilor:

- pentru *ponderi*:

$$\boldsymbol{W}_{\text{new}} = \boldsymbol{W}_{\text{old}} + \eta \boldsymbol{D}_W , \tag{4.2.10-a}$$

- pentru *deplasări*:

$$\boldsymbol{b}_{\text{new}} = \boldsymbol{b}_{\text{old}} + \eta \boldsymbol{D}_b . \tag{4.2.10-b}$$

Cu aceste calcule se încheie iterația curentă şi se trece la o nouă iterație efectuând atribuirile: $\boldsymbol{W}_{\text{new}} \to \boldsymbol{W}_{\text{old}}$, $\boldsymbol{b}_{\text{new}} \to \boldsymbol{b}_{\text{old}}$.

Observații:

1° Inițializarea parametrilor rețelei (adică $\boldsymbol{W}_{\text{old}}$ şi $\boldsymbol{b}_{\text{old}}$ la prima iterație a algoritmului) se face cu valori arbitrare. Pentru simplitate, inițializarea se poate face cu valori nule.

2° Este demonstrat faptul că algoritmul Widrow-Hoff converge, după caz, *către soluția unică* sau *către una din soluțiile* sistemului algebric de ecuații liniare (în accepțiunea discuției din paragraful 4.1). Convergența se realizează în condițiile specifice *metodei gradientului* (ca tehnică de optimizare), aplicată formei pătratice (4.2.2), fiind garantată pentru valori ale ratei de învățare mai mici decât valoarea critică:

$$\eta_c = \frac{2}{\lambda_{\max}(\boldsymbol{R})}, \tag{4.2.11}$$

unde $\lambda_{\max}(\boldsymbol{R})$ notează valoarea proprie maximă a matricei simetrice pozitiv definite $\boldsymbol{R}$.

Astfel, se pot formula următoarele observații:

- pentru η inadecvat mic, deplasarea către optim, în spațiul parametrilor, este foarte lentă,
- pentru η inadecvat mare, deplasarea către optim, în spațiul parametrilor, se realizează cu oscilații (eventual cu pierderea stabilității algoritmului).

3° În cazul când algoritmul se încheie prin satisfacerea condiției $\chi 1$), acuratețea parametrilor furnizați $\boldsymbol{W}_f \in \mathbb{R}^{p \times n}$ şi $\boldsymbol{b}_f \in \mathbb{R}^p$ depinde de valoarea de prag ε stabilită pentru eroarea pătratică globală. Uzual, condiția ($\chi 1$) devine operantă atunci când poare fi satisfăcută relația (4.1.18), adică atunci când valoarea minimă a lui $J(\boldsymbol{v})$ este 0.

4° Algoritmul Widrow-Hoff, privit ca metodă de căutare a minimului funcției $J(\boldsymbol{v})$ (4.1.16'), nu se confruntă cu problema eşuării în minime locale, deoarece $J(\boldsymbol{v})$ este o formă pătratică definită pe spațiul parametrilor $\mathbb{R}^{p(n+1)}$ care posedă numai un punct de minim global egal cu 0, sau o infinitate de astfel de puncte (în accepțiunea discuției din paragraful 4.1). ■

4.3. Facilități software oferite de Neural Network Toolbox

Funcția newlind permite determinarea parametrilor unei rețele ADALINE prin rezolvarea sistemului (4.1.14) în sensul celor mai mici pătrate. Forma de apel este

```
net = newlind(P, T)
```

unde P este matricea R x Q a vectorilor prototip, iar T este matricea S x Q a vectorilor țintă (cu notațiile din paragraful 4.2.2, P reprezintă matricea $\boldsymbol{X} \in \mathbb{R}^{n \times N}$ iar T este matricea $\boldsymbol{Z} \in \mathbb{R}^{p \times N}$). Funcția newlind returnează rețeaua ADALINE cu un singur strat cu S neuroni. Parametrii rețelei sunt calculați pe baza relației (4.1.24).

Funcția newlin poate fi folosită pentru crearea unei rețele ADALINE cu un singur strat. Forma de apel este

net = newlin(PR, S, ID, LR)

unde PR este matricea R x 2 corespunzătoare valorilor minime şi maxime ale celor R componente ale vectorilor prototip, S reprezintă numărul de neuroni al rețelei create, ID este vectorul corespunzător întârzierilor semnalelor de intrare (valoare implicită nulă) şi LR reprezintă rata de învățare ce va fi utilizată în antrenarea rețelei prin aplicarea algoritmului Widrow-Hoff (având valoarea implicită 0.01). Funcția newlin returnează rețeaua ADALINE cu un singur strat cu S neuroni având toți parametrii nuli.

După ce a fost creată, o rețea poate fi antrenată utilizând funcția train care implementează algoritmul de antrenare a unei rețele perceptron conform parametrilor de antrenare aleşi la crearea rețelei. Implicit, pentru actualizarea la fiecare iterație a parametrilor rețelei este apelată funcția learnwh care implementează relațiile (4.2.9)-(4.2.10) din algoritmul Widrow-Hoff. Criteriul de performanță utilizat este MSE. Numărul maxim de epoci de antrenare a rețelei şi valoarea de prag pentru MSE sunt reprezentate de net.trainParam.epochs şi, respectiv, net.trainParam.goal.

Funcția maxlinlr, având forma de apel

lr = maxlinlr(P)

poate fi folosită pentru determinarea ratei de învățare maxime care asigură stabilitatea algoritmului de antrenare Widrow-Hoff. Această funcție primeşte ca argument matricea P a vectorilor prototip (respectiv, cu notațiile din paragraful 4.2.2, matricea $\boldsymbol{X} \in \mathbb{R}^{n \times N}$) şi implementează în MATLAB relația (4.1.36) sub forma $\eta_c = 1/\lambda_{\max}(\boldsymbol{X}\boldsymbol{X}^T)$ în cazul în care se doreşte antrenarea unei rețele ADALINE cu deplasare nulă pentru toți neuronii, sau sub forma $\eta_c = 1/\lambda_{\max}(\boldsymbol{X}_2\boldsymbol{X}_2^T)$, cu $\boldsymbol{X}_2 = \begin{bmatrix} \boldsymbol{X} \\ \mathbf{1}_{1\times N} \end{bmatrix}$ (unde $\mathbf{1}_{1\times N}$ notează vectorul linie cu N elemente egale cu 1) dacă se doreşte antrenarea unei rețele ADALINE cu deplasare nenulă.

Pentru simularea rețelei antrenate se poate utiliza funcția sim.

În cazul unui neuron cu o singură intrare şi funcție de activare liniară, pentru reprezentarea grafică tridimensională a suprafeței erorii pătratice globale (ca funcție de pondere şi deplasarea neuronului) se pot utiliza funcțiile errsurf şi plotes (apelate în această ordine).

Pentru aprofundarea noțiunilor prezentate în acest capitol se recomanda studierea problemelor cu soluții analitice 7.6 – 7.8 şi a problemelor cu soluții numerice 8.8 – 8.11.

Capitolul 5

APLICAȚII ALE REȚELELOR FEEDFORWARD MULTISTRAT ÎN PROBLEME DE APROXIMARE ŞI CLASIFICARE

Rețelele neuronale feed-forward multistrat cu noduri de intrare sigmoidale şi noduri de ieşire liniare pot fi utilizate pentru a realiza aproximarea unor funcții neliniare, definite cu ajutorul unei mulțimi de vectori prototip şi respectiv a unei mulțimi de vectori țintă. Numărul de intrări şi ieşiri ai rețelei sunt impuse de dimensiunea vectorilor țintă şi respectiv - dimensiunea vectorilor prototip. Numărul de neuroni din stratul de intrare (şi din stratul ascuns - în cazul rețelelor cu trei straturi) este stabilit de utilizator. Aceste arhitecturi neuronale sunt denumite şi rețele MLP (eng. *MultiLayer Perceptron*). Pentru studierea transferului intrare-ieşire pentru acest tip de rețele se recomandă utilizarea funcțiilor disponibile în pachetul ***Neural Network Toolbox***. Un accent deosebit se va pune pe vizualizarea grafică a progresului antrenării şi anume pe vizualizare grafică tridimensională a suprafeței de eroare şi a căutării minimului în cazul unui neuron cu o singură intrare şi funcție de activare sigmoidală, respectiv pe vizualizare grafică bidimensională a evoluției procesului de aproximare în cazul unei rețele cu două straturi.

5.1. Proprietatea de aproximare a rețelelor feed-forward cu două straturi

O rețea cu două straturi cu noduri sigmoidale în stratul de intrare şi liniare în stratul de ieşire realizează transferul intrare-ieşire prin funcția:

$$\boldsymbol{f}:\mathbb{R}^n \to \mathbb{R}^p, \tag{5.1.1-a}$$

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{\sigma}^2\left(\boldsymbol{W}^2\boldsymbol{\sigma}^1(\boldsymbol{W}^1\boldsymbol{x}+\boldsymbol{b}^1)+\boldsymbol{b}^2\right), \tag{5.1.1-b}$$

unde notațiile au următoarea semnificație, în conformitate cu arhitectura reprezentată grafic în figura 5.1.1:

$\boldsymbol{x} \in \mathbb{R}^n$ – vectorul de intrare;

$\boldsymbol{y} \in \mathbb{R}^p$ – vectorul de ieşire;

$\boldsymbol{W}^1 \in \mathbb{R}^{m \times n}$ – matricea ponderilor stratului de intrare;

$\boldsymbol{b}^1 \in \mathbb{R}^m$ – vectorul deplasărilor stratului de intrare;

$\boldsymbol{W}^2 \in \mathbb{R}^{p \times m}$ – matricea ponderilor stratului de ieşire;

$\boldsymbol{b}^2 \in \mathbb{R}^p$ – vectorul deplasărilor stratului de ieşire;

$\boldsymbol{\sigma}^1 : \mathbb{R}^m \to \mathbb{R}^m$, $\boldsymbol{\sigma}^1(\boldsymbol{u}^1) = \left[\sigma^1(u_1^1) \quad \dots \quad \sigma^1(u_m^1)\right]^T$ – funcția vectorială de activare a stratului de intrare, unde $\boldsymbol{u}^1 = \boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1 \overset{not}{=} \left[u_1^1 \dots u_m^1\right]^T$ şi σ^1 are expresia analitică:

$\sigma^1(u) = 1/(1+\mathrm{e}^u)$ în cazul sigmoidului unipolar şi, respectiv,

$\sigma^1(u) = (\mathrm{e}^u - \mathrm{e}^{-u})/(\mathrm{e}^u + \mathrm{e}^{-u})$ în cazul sigmoidului bipolar;

$\boldsymbol{\sigma}^2 : \mathbb{R}^p \to \mathbb{R}^p$, $\boldsymbol{\sigma}^2(\boldsymbol{u}^2) = \boldsymbol{u}^2$ (echivalent cu $\boldsymbol{\sigma}^2(\boldsymbol{u}^2) = \left[\sigma^2(u_1^2) \dots \sigma^2(u_p^2)\right]^T$ şi $\sigma^2(u) = u$) – funcția de activare (liniară) a stratului de ieşire, unde $\boldsymbol{u}^2 = \boldsymbol{W}^2\boldsymbol{x}^2 + \boldsymbol{b}^2 \overset{not}{=} \left[u_1^1 \dots u_p^1\right]^T \in \mathbb{R}^p$, $\boldsymbol{x}^2 \equiv \boldsymbol{y}^1 = \boldsymbol{\sigma}^1(\boldsymbol{u}^1)$.

Transferul intrare-ieşire realizat este parametrizat de ponderile $\boldsymbol{W}^1 \in \mathbb{R}^{m \times n}$, $\boldsymbol{W}^2 \in \mathbb{R}^{p \times m}$ şi deplasările $\boldsymbol{b}^1 \in \mathbb{R}^m$, $\boldsymbol{b}^2 \in \mathbb{R}^p$ rețelei.

Fie funcția:

$$\varphi : \mathbb{R}^n \to \mathbb{R}^p, \tag{5.1.2}$$

netedă pe o mulțime compactă $S \subseteq \mathbb{R}^n$.

Este demonstrat faptul că, dacă rețeaua neuronală posedă *un număr adecvat de noduri* m în stratul de intrare, atunci parametrii rețelei pot fi determinați de aşa manieră încât $\boldsymbol{f}$ definită prin (1) *să aproximeze* pe S funcția φ din (2) *cu o precizie impusă* (în sensul de „oricât de bună"). Într-o formulare matematică riguroasă, acest rezultat remarcabil poate fi scris sub forma:

$$\begin{aligned} &\forall \varepsilon > 0, \quad \exists m \in \mathbb{N} \text{ şi } \boldsymbol{W}^1 \in \mathbb{R}^{m \times n}, \boldsymbol{b}^1 \in \mathbb{R}^m, \boldsymbol{W}^2 \in \mathbb{R}^{p \times m}, \boldsymbol{b}^2 \in \mathbb{R}^p \text{ a.î.} \\ &\qquad \left\| \boldsymbol{f}(\boldsymbol{x}) - \varphi(\boldsymbol{x}) \right\| < \varepsilon, \quad \forall \boldsymbol{x} \in S. \end{aligned} \tag{5.1.3}$$

Observație:

Proprietatea de aproximare prezentată mai sus *nu furnizează informații concrete* privind alegerea numărului de noduri $m \in \mathbb{N}$ şi determinarea parametrilor rețelei $\boldsymbol{W}^1 \in \mathbb{R}^{m \times n}$, $\boldsymbol{b}^1 \in \mathbb{R}^m$, $\boldsymbol{W}^2 \in \mathbb{R}^{p \times m}$, $\boldsymbol{b}^2 \in \mathbb{R}^p$, ci *garantează numai existența acestor valori*. ■

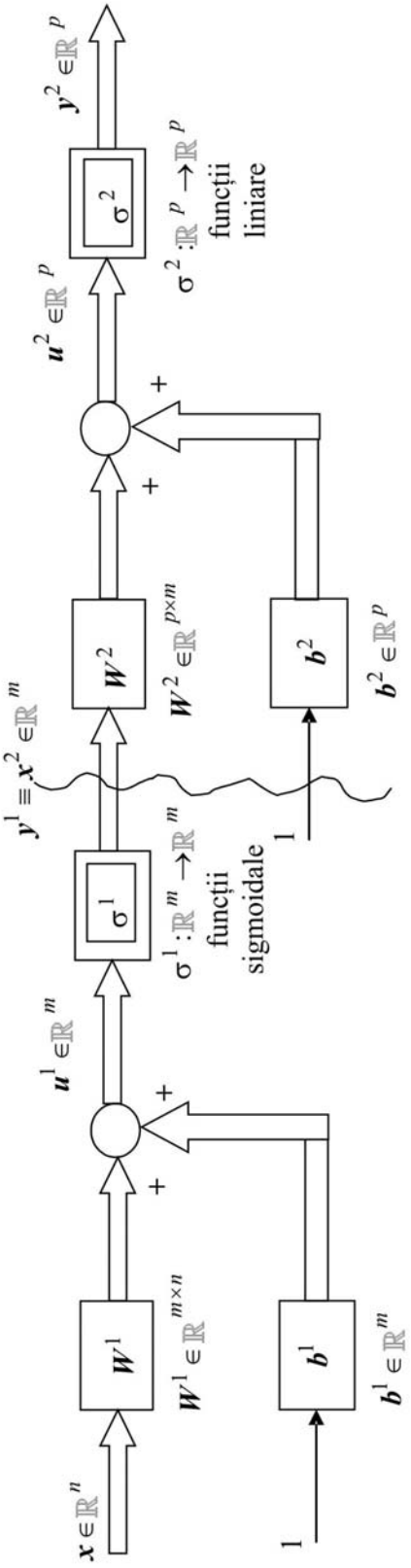


Figura 5.1.1. *Schema bloc a unei rețele neuronale cu două straturi, utilizată în probleme de aproximare neliniară*

Din punct de vedere practic, exploatarea rezultatului discutat anterior are loc în următorul context. Funcția φ din (5.1.2) este cunoscută într-un *număr finit N* de puncte $\boldsymbol{x}_i \in S \subseteq \mathbb{R}^n$, pentru care avem:

$$\varphi(\boldsymbol{x}_i) = \boldsymbol{z}_i\ ,\ \ i = 1,...,N\ . \tag{5.1.4}$$

La intrarea rețelei se prezintă *vectorii prototip*:

$$\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N \in \mathbb{R}^n, \tag{5.1.5}$$

pentru care se consideră *vectorii țintă*:

$$\boldsymbol{z}_1, \boldsymbol{z}_2, ..., \boldsymbol{z}_N \in \mathbb{R}^p \tag{5.1.6}$$

definiți conform (5.1.4). Notând prin $\boldsymbol{y}_i$, $i = 1,\ldots,N$, vectorii de ieşire ai MLP, adică:

$$\boldsymbol{y}_i = \boldsymbol{f}(\boldsymbol{x}_i) \in \mathbb{R}^p,\ \ i = 1,\ldots,N, \tag{5.1.7}$$

se definesc vectorii eroare prin:

$$\boldsymbol{e}_i\ =\ \boldsymbol{z}_i\ -\ \boldsymbol{y}_i\ \in\ \mathbb{R}^{\,p}\ ,\ \ i\ =\ 1,\ldots,\ N\ . \tag{5.1.8}$$

Considerând un număr de neuroni *m* în stratul de intrare al rețelei, vectorii de ieşire şi, implicit, vectorii de eroare, depind de parametrii rețelei $\boldsymbol{W}^1 \in \mathbb{R}^{m\times n}$, $\boldsymbol{b}^1 \in \mathbb{R}^m$, $\boldsymbol{W}^2 \in \mathbb{R}^{p\times m}$, $\boldsymbol{b}^2 \in \mathbb{R}^p$. Se defineşte funcția obiectiv MSE (eroarea pătratică medie):

$$J_{MSE}\left(\boldsymbol{W}^1, \boldsymbol{b}^1, \boldsymbol{W}^2, \boldsymbol{b}^2\right) = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{e}_i^T \boldsymbol{e}_i\ . \tag{5.1.9}$$

Pentru orice număr de neuroni *m* în stratul de intrare al rețelei, există valori ale parametrilor rețelei notate prin $\boldsymbol{W}^{1*} \in \mathbb{R}^{m\times n}$, $\boldsymbol{b}^{1*} \in \mathbb{R}^m$, $\boldsymbol{W}^{2*} \in \mathbb{R}^{p\times m}$, $\boldsymbol{b}^{2*} \in \mathbb{R}^p$, care asigură *minimizarea erorii pătratice medii*:

$$J_{MSE}\left(\boldsymbol{W}^{1*}, \boldsymbol{b}^{1*}, \boldsymbol{W}^{2*}, \boldsymbol{b}^{2*}\right) \le J_{MSE}\left(\boldsymbol{W}^1, \boldsymbol{b}^1, \boldsymbol{W}^2, \boldsymbol{b}^2\right). \tag{5.1.10}$$

Observație:

Parametrii $\boldsymbol{W}^{1*} \in \mathbb{R}^{m\times n}$, $\boldsymbol{b}^{1*} \in \mathbb{R}^m$, $\boldsymbol{W}^{2*} \in \mathbb{R}^{p\times m}$, $\boldsymbol{b}^{2*} \in \mathbb{R}^p$ din (5.1.9) nu asigură şi satisfacerea condiției (5.1.3) pe întreg compactul $S \subseteq \mathbb{R}^n$. În schimb, inegalitatea (5.1.9) posedă o semnificație numerică concretă, furnizând astfel un obiectiv consistent pentru antrenarea rețelei neuronale. ■

5.2. Utilizarea rețelelor feedforward cu două straturi

Utilizarea rețelelor neuronale feed-forward multistrat în problemele de aproximare neliniare se bazează pe antrenarea rețelelor prin algoritmul de propagare inversă. Denumirea algoritmului provine din faptul că pentru antrenare informațiile sunt procesate în sens invers, de la ieşirea rețelei, către intrare.

5.2.1. Obiectivul antrenării rețelei

Pornind de la *mulțimea vectorilor prototip* (5.1.5) şi *mulțimea vectorilor țintă* (5.1.6), prin antrenarea unei rețele cu m noduri în stratul de intrare trebuie să se determine parametrii acesteia $\boldsymbol{W}^{1*} \in \mathbb{R}^{m\times n}$, $\boldsymbol{b}^{1*} \in \mathbb{R}^{m}$, $\boldsymbol{W}^{2*} \in \mathbb{R}^{p\times m}$, $\boldsymbol{b}^{2*} \in \mathbb{R}^{p}$ care să asigure *minimizarea erorii pătratice globale*, adică satisfacerea inegalității (5.1.10). Cu aceste valori ale parametrilor, funcția $\boldsymbol{f}$ definită prin (5.1.1) realizează o aproximare (neliniară) in sensul celor mai mici pătrate a funcției φ, cunoscută prin intermediul eşantioanelor (5.1.4). Valoarea minimă obținută în (5.1.9) depinde (în general) de numărul de noduri m din stratul de intrare al rețelei şi deci m va influența calitatea aproximării realizate de rețea pentru funcția φ.

5.2.2. Algoritmul de antrenare prin propagare inversă

Algoritmul de antrenare prin *propagare inversă* (eng. *backpropagation*) se bazează pe *minimizarea erorii pătratice medii* (MSE) prin *metoda gradientului*.

Să notăm prin $\boldsymbol{v} \in \mathbb{R}^{m(n+1)+p(m+1)}$ vectorul parametrilor rețelei construit cu elementele matricelor $\boldsymbol{W}^1 \in \mathbb{R}^{m\times n}$, $\boldsymbol{W}^2 \in \mathbb{R}^{p\times m}$ şi ale vectorilor $\boldsymbol{b}^1 \in \mathbb{R}^{m}$, $\boldsymbol{b}^2 \in \mathbb{R}^{p}$, elemente ce sunt dispuse în vectorul $\boldsymbol{v}$ conform unei reguli prestabilite (de exemplu, întâi liniile matricei $\left[\boldsymbol{W}^1\ \boldsymbol{b}^1\right]$, urmând apoi liniile matricei $\left[\boldsymbol{W}^2\ \boldsymbol{b}^2\right]$).

Considerând vectorii eroare $\boldsymbol{e}_i$ definiți prin (5.1.8), ca funcții de parametrii rețelei (adică de elementele vectorului $\boldsymbol{v}$) se defineşte funcția obiectiv (5.1.9):

$$J(\boldsymbol{v}) = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{e}_i^{\,T}(\boldsymbol{v})\,\boldsymbol{e}_i(\boldsymbol{v}). \tag{5.2.1}$$

Minimizarea prin metoda gradientului a funcției $J(\boldsymbol{v})$ (care înseamnă satisfacerea condiției (5.1.10)) constă în modificarea vectorului parametrilor $\boldsymbol{v} \in \mathbb{R}^{m(n+1)+p(m+1)}$ la fiecare iterație, în conformitate cu formula:

$$\boldsymbol{v}_{\text{new}} = \boldsymbol{v}_{\text{old}} - \eta\,\nabla J(\boldsymbol{v}_{\text{old}}), \tag{5.2.2}$$

unde $\eta > 0$ notează *pasul metodei de optimizare*, iar $\nabla J(\boldsymbol{v}_{\text{old}}) = \left[\partial J/\partial \boldsymbol{v}\right]^T\Big|_{\boldsymbol{v}_{\text{old}}}$ notează vectorul gradient (normal la suprafețele de nivel constant ale lui J (5.2.1), care trec prin $\boldsymbol{v}_{\text{old}}$).

Printr-o exprimare adecvată a vectorului gradient, relația (5.2.2) poate fi formulată *direct în termenii parametrilor rețelei* $\boldsymbol{W}^1 \in \mathbb{R}^{m\times n}$, $\boldsymbol{b}^1 \in \mathbb{R}^{m}$, $\boldsymbol{W}^2 \in \mathbb{R}^{p\times m}$, $\boldsymbol{b}^2 \in \mathbb{R}^{p}$, furnizând astfel *mecanismul de efectuare a unei iterații* pentru *algoritmul de antrenare prin propagare inversă*. Termenul de "propagare inversă" se referă la maniera de calcul a valorilor curente ale vectorului gradient care se bazează pe o strategie de propagare a erorilor, definite prin (5.1.8), de la stratul de ieşire, înapoi către stratul de intrare.

Pentru algoritmul de antrenare se stabilesc, de către utilizator, un *număr maxim de iterații* sau *epoci* şi o valoare ε care se doreşte a fi atinsă în minimizarea funcției obiectiv (5.2.1) (adică *un prag pentru eroarea pătratică globală*). Atingerea unui prag *oarecare* de

eroare ε impus de utilizator *nu este întotdeauna posibilă*, deoarece valoarea minimă a lui $J(\boldsymbol{v})$ definită prin (5.2.1) poate fi strict pozitivă.

De asemenea, se fixează (de către utilizator) o valoare pentru pasul metodei de optimizare $\eta > 0$, care, în contextul problemei de optimizare, este referită sub denumirea de *rată de învățare* (eng. *learning rate*).

Se organizează vectorii prototip $\boldsymbol{x}_i \in \mathbb{R}^n$, $i = 1,\ldots,N$, şi cei țintă $\boldsymbol{z}_i \in \mathbb{R}^p$, $i = 1,\ldots,N$, sub forma matricelor:

$$\boldsymbol{X} = [\boldsymbol{x}_1 \ldots \boldsymbol{x}_N] \in \mathbb{R}^{n \times N}, \tag{5.2.3}$$

respectiv:

$$\boldsymbol{Z} = [\boldsymbol{z}_1 \ldots \boldsymbol{z}_N] \in \mathbb{R}^{p \times N}. \tag{5.2.4}$$

La *fiecare iterație* a algoritmului de propagare inversă se parcurg etapele descrise mai jos. Valorile parametrilor rețelei la începerea unei iterații sunt notate prin $\boldsymbol{W}^1_{\text{old}} \in \mathbb{R}^{m \times n}$, $\boldsymbol{W}^2_{\text{old}} \in \mathbb{R}^{p \times m}$ (pentru ponderi) şi, respectiv, $\boldsymbol{b}^1_{\text{old}} \in \mathbb{R}^m$, $\boldsymbol{b}^2_{\text{old}} \in \mathbb{R}^p$ (pentru deplasări).

Etapa I (Etapa de prezentare)

Se prezintă vectorii prototip $\boldsymbol{x}_i$, $i = 1,...,N$, la intrarea rețelei neurale, adică pentru valorile curente ale parametrilor rețelei ($\boldsymbol{W}^1_{\text{old}}$, $\boldsymbol{W}^2_{\text{old}}$, $\boldsymbol{b}^1_{\text{old}}$, $\boldsymbol{b}^2_{\text{old}}$) se calculează, conform (5.1.1-b), ieşirile rețelei, obținând:

$$\boldsymbol{y}_i = \boldsymbol{f}(\boldsymbol{x}_i) = \boldsymbol{\sigma}^2\left(\boldsymbol{W}^2_{\text{old}}\boldsymbol{\sigma}^1(\boldsymbol{W}^1_{\text{old}}\boldsymbol{x}_i + \boldsymbol{b}^1_{\text{old}}) + \boldsymbol{b}^2_{\text{old}}\right),\ i = 1,...,N, \tag{5.2.5}$$

care se organizează sub forma matricei:

$$\boldsymbol{Y} = [\boldsymbol{y}_1 \ldots \boldsymbol{y}_N] \in \mathbb{R}^{p \times N}. \tag{5.2.6}$$

Etapa II (Etapa de verificare)

Se calculează vectorii erorilor $\boldsymbol{e}_i$, $i = 1,...,N$, definiți prin (5.1.8) ca diferența dintre vectorii țintă $\boldsymbol{z}_i \in \mathbb{R}^p$ şi vectorii ieşirii la iterația curentă $\boldsymbol{y}_i \in \mathbb{R}^p$, construind matricea

$$\boldsymbol{E} = [\boldsymbol{e}_1 \ldots \boldsymbol{e}_N] = \boldsymbol{Z} - \boldsymbol{Y} \in \mathbb{R}^{p \times N}. \tag{5.2.7}$$

Algoritmul se opreşte dacă este îndeplinită una din următoarele două condiții:
($\chi 1$) eroarea pătratică medie este inferioară pragului ε stabilit la startarea algoritmului, adică:

$$J\left(\boldsymbol{W}^2_{\text{old}}, \boldsymbol{W}^1_{\text{old}}, \boldsymbol{b}^1_{\text{old}}, \boldsymbol{b}^2_{\text{old}}\right) = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{e}_i{}^T \boldsymbol{e}_i < \varepsilon; \tag{5.2.8}$$

sau:
($\chi 2$) a fost atins numărul maxim de iterații stabilit la startarea algoritmului.

La oprire, algoritmul va furniza valorile parametrilor rețelei rezultate din antrenare, care vor fi notate prin $\boldsymbol{W}_f^1 \in \mathbb{R}^{m\times n}$, $\boldsymbol{W}_f^2 \in \mathbb{R}^{p\times m}$ (pentru ponderi) şi, respectiv, $\boldsymbol{b}_f^1 \in \mathbb{R}^m$, $\boldsymbol{b}_f^2 \in \mathbb{R}^p$ (pentru deplasări).

Dacă nici una din condițiile ($\chi 1$) sau ($\chi 2$) nu este îndeplinită, se continuă cu etapa următoare a iterației curente.

Etapa III (Etapa de calcul a vectorilor delta prin propagare inversă)

Pentru valorile curente ale parametrilor rețelei $\boldsymbol{W}_{\text{old}}^1 \in \mathbb{R}^{m\times n}$, $\boldsymbol{W}_{\text{old}}^2 \in \mathbb{R}^{p\times m}$, $\boldsymbol{b}_{\text{old}}^1 \in \mathbb{R}^m$, $\boldsymbol{b}_{\text{old}}^2 \in \mathbb{R}^p$, pentru fiecare vector prototip $\boldsymbol{x}_i$, $i=1,\ldots,N$, se calculează *matricele Jacobian ale funcțiilor de activare* (vectoriale) ale celor două straturi, notate prin:

$$\boldsymbol{Q}_i^1 \in \mathbb{R}^{m\times m},\ \boldsymbol{Q}_i^2 \in \mathbb{R}^{p\times p},\ i=1,\ldots,N.$$

- pentru stratul de intrare:

$$\boldsymbol{Q}_i^1 = \operatorname{diag}\left\{q_i^1(1),\ldots,q_i^1(m)\right\}, \tag{5.2.9-a}$$

unde:

$$q_i^1(k) = (\sigma^1)(u_i^1(k)) = \frac{1}{(\operatorname{Inv}\sigma^1)(y_i^1(k))},\quad k=1,\ldots,m; \tag{5.2.9-b}$$

- pentru stratul de ieşire:

$$\boldsymbol{Q}_i^2 = \operatorname{diag}\left\{q_i^2(1),\ldots,q_i^2(m)\right\}, \tag{5.2.10-a}$$

unde:

$$q_i^2(k) = (\sigma^2)(u_i^2(k)) = \frac{1}{(\operatorname{Inv}\sigma^2)(y_i^2(k))} = 1,\quad k=1,\ldots,m. \tag{5.2.10-b}$$

În relațiile (5.2.9-b) şi (5.2.10-b) care definesc elementele matricelor Jacobian, notațiile $(\operatorname{Inv}\sigma^1)$ şi $(\operatorname{Inv}\sigma^2)$ au semnificația de inverse ale funcțiilor de activare σ^1 şi respectiv σ^2 corespunzătoare celor două straturi. Deoarece funcțiile de activare ale celui de al doilea strat sunt liniare, $\boldsymbol{Q}_i^2$ este *matricea unitate*, $i=1,\ldots,N$.

Cu ajutorul matricelor Jacobian $\boldsymbol{Q}_i^1 \in \mathbb{R}^{m\times m}$, $\boldsymbol{Q}_i^2 \in \mathbb{R}^{p\times p}$ şi a vectorilor erorilor $\boldsymbol{e}_i \in \mathbb{R}^p$, $i=1,\ldots,N$, $i=1,\ldots,N$, se definesc *vectorii delta* (prin procesarea informației de la ieşire, către intrare, procesare numită *propagare inversă*):

- pentru stratul de ieşire:

$$\boldsymbol{\delta}_i^2 = \boldsymbol{Q}_i^2\boldsymbol{e}_i = \boldsymbol{e}_i \in \mathbb{R}^p,\quad i=1,\ldots,N; \tag{5.2.11}$$

- pentru stratul de intrare:

$$\boldsymbol{\delta}_i^1 = \boldsymbol{Q}_i^1\left((\boldsymbol{W}^2)^T\boldsymbol{\delta}_i^2\right) \in \mathbb{R}^m,\quad i=1,\ldots,N. \tag{5.2.12}$$

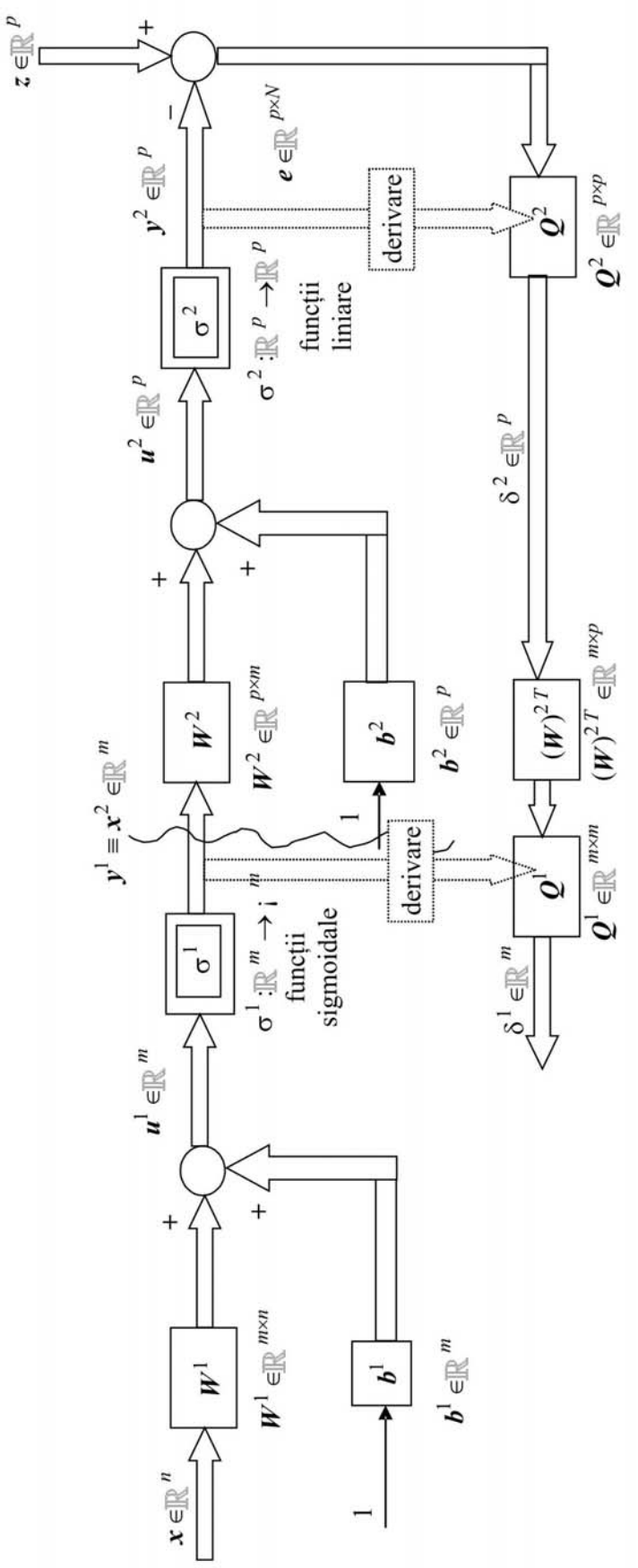


Figura 5.2.1. *Schema bloc a procesării în sens invers a informației în antrenarea unei rețele cu două straturi prin algoritmul propagării inverse - calculul vectorilor delta* δ^2 *și* δ^1

Astfel, suntem în măsură să construim *matricele delta* ale celor două straturi:

- pentru stratul de ieşire

$$\Delta^2 = [\boldsymbol{\delta}_i^2 .... \boldsymbol{\delta}_N^2] \in \mathbb{R}^{p \times N}, \tag{5.2.13}$$

- pentru stratul de intrare

$$\Delta^1 = [\boldsymbol{\delta}_i^1 .... \boldsymbol{\delta}_N^1] \in \mathbb{R}^{m \times N}. \tag{5.2.14}$$

O reprezentare sub forma de schemă bloc a modului de calcul al *vectorilor delta* prin propagare inversă într-o rețea cu două straturi este dată în figura 5.2.1.

Etapa IV (Etapa de actualizare a parametrilor)

Cu ajutorul matricelor delta se pot calcula *matricele de actualizare* a parametrilor aplicând *regula delta* pe fiecare strat:

- pentru stratul de ieşire
  – ponderi

$$\boldsymbol{D}_{\boldsymbol{W}^2} = \Delta^2 (\boldsymbol{X}^2)^T \in \mathbb{R}^{p \times m}, \tag{5.2.15-a}$$

  – deplasări

$$\boldsymbol{D}_{\boldsymbol{b}^2} = \Delta^2 \underbrace{[1 \ldots 1]^T}_{N \text{ elemente}} \in \mathbb{R}^p; \tag{5.2.15-b}$$

- pentru stratul de intrare
  – ponderi

$$\boldsymbol{D}_{\boldsymbol{W}^1} = \Delta^1 \boldsymbol{X}^T \in \mathbb{R}^{m \times n}, \tag{5.2.16-a}$$

  – deplasări

$$\boldsymbol{D}_{\boldsymbol{b}^1} = \Delta^1 \underbrace{[1 \ldots 1]^T}_{N \text{ elemente}} \in \mathbb{R}^m. \tag{5.2.16-b}$$

În relația (5.2.15-a), matricea $\boldsymbol{X}^2 \in \mathbb{R}^{m \times N}$ colectează vectorii $\boldsymbol{x}_i^2 = \boldsymbol{y}_i^1 \in \mathbb{R}^m$, $i = 1, \ldots, N$, care se prezintă la intrarea celui de-al doilea strat, adică:

$$\boldsymbol{X}^2 = [\boldsymbol{x}_1^2 \ldots \boldsymbol{x}_N^2] \in \mathbb{R}^{m \times N}. \tag{5.2.17}$$

Pe baza matricelor de actualizare se determină noile valori ale parametrilor rețelei:

- pentru stratul de ieşire
  – ponderi

$$\boldsymbol{W}_{\text{new}}^2 = \boldsymbol{W}_{\text{old}}^2 + \eta \boldsymbol{D}_{\boldsymbol{W}^2}, \tag{5.2.18-a}$$

  – deplasări

$$\boldsymbol{b}_{\text{new}}^2 = \boldsymbol{b}_{\text{old}}^2 + \eta \boldsymbol{D}_{\boldsymbol{b}^2}; \tag{5.2.18-b}$$

- pentru stratul de intrare
  – ponderi

$$\boldsymbol{W}_{\text{new}}^1 = \boldsymbol{W}_{\text{old}}^1 + \eta \boldsymbol{D}_{\boldsymbol{W}^1}, \tag{5.2.19-a}$$

– deplasări

$$\boldsymbol{b}^1_{\text{new}} = \boldsymbol{b}^1_{\text{old}} + \eta \boldsymbol{D}_{\boldsymbol{b}^1}. \tag{5.2.19-b}$$

Cu aceste calcule, iterația curentă se încheie şi se trece la o nouă iterație, efectuând atribuirile:

$$\begin{aligned} &\boldsymbol{W}^2_{\text{new}} \to \boldsymbol{W}^2_{\text{old}}, \quad \boldsymbol{b}^2_{\text{new}} \to \boldsymbol{b}^2_{\text{old}}, \\ &\boldsymbol{W}^1_{\text{new}} \to \boldsymbol{W}^1_{\text{old}}, \quad \boldsymbol{b}^1_{\text{new}} \to \boldsymbol{b}^1_{\text{old}}. \end{aligned} \tag{5.2.20}$$

5.2.3. Algoritmul de antrenare prin propagare inversă privit ca problemă standard de optimizare

În acest paragraf vom prezenta o demonstrație pentru egalitățile (5.2.15) şi (5.2.16), care furnizează matricele de actualizare a parametrilor rețelei la fiecare iterație a algoritmului de antrenare. În acest scop, vom porni de la forma iterației standard a metodei de gradient, exprimată prin (5.2.2).

Aplicarea iterațiilor de tip (5.2.2) pentru funcția obiectiv (5.2.1) permite ajustarea parametrilor rețelei în conformitate cu metoda gradientului care operează în spațiul $\mathbb{R}^{m(n+1)+p(m+1)}$. Vectorii acestui spațiu au forma:

$$\boldsymbol{v} = \left[\tilde{\boldsymbol{w}}^1_1 \ldots \tilde{\boldsymbol{w}}^1_m \ \tilde{\boldsymbol{w}}^2_1 \ldots \tilde{\boldsymbol{w}}^2_p \right]^T, \tag{5.2.21}$$

unde $\tilde{\boldsymbol{w}}^1_k$, $k = 1,...,m$, şi $\tilde{\boldsymbol{w}}^2_j$, $j = 1,...,p$, notează liniile matricelor parametrilor pentru *primul strat*:

$$\tilde{\boldsymbol{W}}^1 = \left[\boldsymbol{W}^1 \ \boldsymbol{b}^1 \right] = \begin{bmatrix} \tilde{\boldsymbol{w}}^1_1 \\ \cdots \\ \tilde{\boldsymbol{w}}^1_m \end{bmatrix} \in \mathbb{R}^{m \times (n+1)} \tag{5.2.22}$$

şi respectiv pentru al *doilea strat*:

$$\tilde{\boldsymbol{W}}^2 = \left[\boldsymbol{W}^2 \ \boldsymbol{b}^2 \right] = \begin{bmatrix} \tilde{\boldsymbol{w}}^2_1 \\ \cdots \\ \tilde{\boldsymbol{w}}^2_p \end{bmatrix} \in \mathbb{R}^{p \times (m+1)}. \tag{5.2.23}$$

Scriind iterația (5.2.2) pentru fiecare set de parametri $\tilde{\boldsymbol{w}}^1_k$, $k = 1,...,m$, în parte:

$$\tilde{\boldsymbol{w}}^1_{k\ \text{new}} = \tilde{\boldsymbol{w}}^1_{k\ \text{old}} - \eta \frac{\partial J}{\partial \tilde{\boldsymbol{w}}^1_k}, \ k = 1,\ldots,m, \tag{5.2.24}$$

şi asamblând cele m egalități vectoriale din (5.2.24) sub formă matriceală, obținem pentru *stratul de intrare*:

$$\tilde{\boldsymbol{W}}^1_{\text{new}} = \tilde{\boldsymbol{W}}^1_{\text{old}} - \eta \begin{bmatrix} \dfrac{\partial J}{\partial \tilde{\boldsymbol{w}}^1_1} \\ \cdots \\ \dfrac{\partial J}{\partial \tilde{\boldsymbol{w}}^1_m} \end{bmatrix}. \tag{5.2.25}$$

Egalitatea (5.2.25) este însă de aceeaşi formă cu (5.2.19), constatare ce arată că actualizarea parametrilor în stratul de intrare se realizează în *scrierea matriceală* de tip (5.2.19), prin intermediul matricei:

$$\boldsymbol{D}_{\tilde{\boldsymbol{W}}^1} = \begin{bmatrix} \boldsymbol{D}_{\boldsymbol{W}^1} & \boldsymbol{D}_{\boldsymbol{b}^1} \end{bmatrix} = -\begin{bmatrix} \dfrac{\partial J}{\partial \tilde{\boldsymbol{w}}^1_1} \\ \cdots \\ \dfrac{\partial J}{\partial \tilde{\boldsymbol{w}}^1_m} \end{bmatrix} \in \mathbb{R}^{m\times(n+1)}. \tag{5.2.26}$$

În mod analog se arată că actualizarea parametrilor în stratul de ieşire se realizează în *scrierea matriceală* de tip (5.2.18), prin intermediul matricei:

$$\boldsymbol{D}_{\tilde{\boldsymbol{W}}^2} = \begin{bmatrix} \boldsymbol{D}_{\boldsymbol{W}^2} & \boldsymbol{D}_{\boldsymbol{b}^2} \end{bmatrix} = -\begin{bmatrix} \dfrac{\partial J}{\partial \tilde{\boldsymbol{w}}^2_1} \\ \cdots \\ \dfrac{\partial J}{\partial \tilde{\boldsymbol{w}}^2_p} \end{bmatrix} \in \mathbb{R}^{p\times(m+1)}. \tag{5.2.27}$$

Pe de altă parte, scriind funcția obiectiv (5.2.1) sub forma:

$$J(\boldsymbol{v}) = \sum_{i=1}^{N} J_i, \quad J_{\text{i}} = \frac{1}{2}\boldsymbol{e}_i^T \boldsymbol{e}_i, \quad i = 1,\ldots,N, \tag{5.2.28}$$

vectorii linie $\dfrac{\partial J}{\partial \tilde{\boldsymbol{w}}^1_k}$, $k = 1,...,m$, şi $\dfrac{\partial J}{\partial \tilde{\boldsymbol{w}}^2_j}$, $j = 1,\ldots,p$, ce intră în structura matricelor $\boldsymbol{D}_{\tilde{\boldsymbol{W}}^1}$ din (5.2.26) şi respectiv $\boldsymbol{D}_{\tilde{\boldsymbol{W}}^2}$ din (5.2.27) pot fi exprimați drept:

$$\frac{\partial J}{\partial \tilde{\boldsymbol{w}}^1_k} = \sum_{i=1}^{N} \frac{\partial J_{\text{i}}}{\partial \tilde{\boldsymbol{w}}^1_k}, \quad k = 1,\ldots,m, \tag{5.2.29}$$

şi respectiv:

$$\frac{\partial J}{\partial \tilde{\boldsymbol{w}}^2_j} = \sum_{i=1}^{N} \frac{\partial J_{\text{i}}}{\partial \tilde{\boldsymbol{w}}^2_j}, \quad j = 1,\ldots,p. \tag{5.2.30}$$

Cu această observație, *matricele de actualizare a parametrilor* pot fi puse sub forma:

- pentru stratul de intrare:

$$\boldsymbol{D}_{\tilde{\boldsymbol{W}}^1} = \sum_{i=1}^{N} \boldsymbol{D}^i_{\tilde{\boldsymbol{W}}^1}, \tag{5.2.31}$$

cu:

$$\boldsymbol{D}^i_{\tilde{\boldsymbol{W}}^1} = -\begin{bmatrix} \dfrac{\partial J_i}{\partial \tilde{\boldsymbol{w}}^1_1} \\ \cdots \\ \dfrac{\partial J_i}{\partial \tilde{\boldsymbol{w}}^1_m} \end{bmatrix} \in \mathbb{R}^{m\times(n+1)}, \quad i=1,\ldots,N; \tag{5.2.32}$$

- pentru stratul de ieşire:

$$\boldsymbol{D}_{\tilde{\boldsymbol{W}}^2} = \sum_{i=1}^{N} \boldsymbol{D}^i_{\tilde{\boldsymbol{W}}^2}, \tag{5.2.33}$$

cu:

$$\boldsymbol{D}^i_{\tilde{\boldsymbol{W}}^2} = -\begin{bmatrix} \dfrac{\partial J_i}{\partial \tilde{\boldsymbol{w}}^2_1} \\ \cdots \\ \dfrac{\partial J_i}{\partial \tilde{\boldsymbol{w}}^2_p} \end{bmatrix} \in \mathbb{R}^{p\times(m+1)}, \quad i=1,\ldots,N \tag{5.2.34}$$

Aplicând regula derivării funcţiilor compuse pentru *stratul de ieşire* se obţine:

$$\frac{\partial J_i}{\partial \tilde{\boldsymbol{w}}^2_j} = \frac{\partial J_i}{\partial \boldsymbol{e}_i}\cdot\frac{\partial \boldsymbol{e}_i}{\partial \boldsymbol{y}_i}\cdot\frac{\partial \boldsymbol{y}_i}{\partial \boldsymbol{u}_i}\cdot\frac{\partial \boldsymbol{u}_i}{\partial \tilde{\boldsymbol{w}}^2_j} = -\left(\boldsymbol{\delta}^2_i\right)^T \begin{bmatrix} 0 \\ \cdots \\ \left(\tilde{\boldsymbol{x}}^2_i\right)^T \\ \cdots \\ 0 \end{bmatrix} \leftarrow j,\ j=1,...,p,\ i=1,...,N, \tag{5.2.35}$$

unde vectorii $\boldsymbol{\delta}^2_i$, $i=1,...,N$, satisfac egalitatea (5.2.11), adică reprezintă *vectorii delta ai stratului de ieşire*. Astfel putem scrie:

$$\boldsymbol{D}^i_{\tilde{\boldsymbol{W}}^2} = \begin{bmatrix} \boldsymbol{\delta}^2_i(1)\left(\tilde{\boldsymbol{x}}^2_i\right)^T \\ \cdots \\ \boldsymbol{\delta}^2_i(p)\left(\tilde{\boldsymbol{x}}^2_i\right)^T \end{bmatrix} = \boldsymbol{\delta}^2_i\left(\tilde{\boldsymbol{x}}^2_i\right)^T, \quad i=1,\ldots,N. \tag{5.2.36}$$

Înlocuind în (5.2.33) se obţine:

$$\boldsymbol{D}_{\tilde{\boldsymbol{W}}^2} = \sum_{i=1}^{N} \boldsymbol{\delta}^2_i\left(\tilde{\boldsymbol{x}}^2_i\right)^T = \left[\boldsymbol{\delta}^2_1 \ldots \boldsymbol{\delta}^2_N\right] \begin{bmatrix} \left(\tilde{\boldsymbol{x}}^2_1\right)^T \\ \cdots \\ \left(\tilde{\boldsymbol{x}}^2_N\right)^T \end{bmatrix}, \tag{5.2.37}$$

adică s-au demonstrat egalităţile (5.2.15) din *Etapa IV* a algoritmului de antrenare.

Aplicăm acum regula derivării funcţiilor compuse pentru *stratul de intrare* şi avem:

$$\frac{\partial J_i}{\partial \tilde{\boldsymbol{w}}_k^1}=\frac{\partial J_i}{\partial \boldsymbol{e}_i}\cdot\frac{\partial \boldsymbol{e}_i}{\partial \boldsymbol{y}_i}\cdot\frac{\partial \boldsymbol{y}_i}{\partial \boldsymbol{u}_i}\cdot\frac{\partial \boldsymbol{u}_i}{\partial \boldsymbol{y}_i'}\cdot\frac{\partial \boldsymbol{y}_i'}{\partial \boldsymbol{u}_i'}\cdot\frac{\partial \boldsymbol{u}_i'}{\partial \tilde{\boldsymbol{w}}_k^1}=-\left(\boldsymbol{\delta}_i^2\right)^T\boldsymbol{W}^2\boldsymbol{Q}_i^1\begin{bmatrix}0\\ \cdots\\ \tilde{\boldsymbol{x}}_i^T\\ \cdots\\ 0\end{bmatrix}=-\left(\boldsymbol{\delta}_i^1\right)^T\begin{bmatrix}0\\ \cdots\\ \tilde{\boldsymbol{x}}_i^T\\ \cdots\\ 0\end{bmatrix}\leftarrow k\,, \qquad (5.2.38)$$

$k=1,...,m$, $i=1,...,N$, unde vectorii $\boldsymbol{\delta}_i^1$, $i=1,...,N$, satisfac egalitatea (5.2.12), adică reprezintă *vectorii delta ai stratului de intrare*. Astfel putem scrie:

$$\boldsymbol{D}_{\tilde{\boldsymbol{W}}^1}^i=\begin{bmatrix}\delta_i^1(1)\tilde{\boldsymbol{x}}_i^T\\ \cdots\\ \delta_i^1(m)\tilde{\boldsymbol{x}}_i^T\end{bmatrix}=\boldsymbol{\delta}_i^1\tilde{\boldsymbol{x}}_i^T,\ i=1,...,N\,. \qquad (5.2.39)$$

Înlocuind în (5.2.31), obținem:

$$\boldsymbol{D}_{\tilde{\boldsymbol{W}}^1}=\sum_{i=1}^N\boldsymbol{\delta}_i^1\tilde{\boldsymbol{x}}_i^T=\left[\boldsymbol{\delta}_1^1\ldots\boldsymbol{\delta}_N^1\right]\begin{bmatrix}\tilde{\boldsymbol{x}}_1^T\\ \cdots\\ \tilde{\boldsymbol{x}}_N^T\end{bmatrix}, \qquad (5.2.40)$$

adică s-au demonstrat şi egalitățile (5.2.16) din *Etapa IV* a algoritmului de antrenare.

Observații:

1° Inițializarea parametrilor rețelei (adică $\boldsymbol{W}_{\text{old}}^1\in\mathbb{R}^{m\times n}$, $\boldsymbol{W}_{\text{old}}^2\in\mathbb{R}^{p\times m}$, $\boldsymbol{b}_{\text{old}}^1\in\mathbb{R}^m$, $\boldsymbol{b}_{\text{old}}^2\in\mathbb{R}^p$ la prima iterație a algoritmului) se face cu valori *subunitare arbitrare*; în acest mod se asigură senzitivitate maximă pentru funcțiile sigmoidale din primul strat al rețelei.

2° Convergența algoritmului de propagare inversă se realizează în condițiile specifice metodei gradientului (ca tehnică de optimizare), fiind puternic dependentă de valoarea ratei de învățare η şi anume:

- pentru η inadecvat mic, deplasarea către optim, în spațiul parametrilor, este foarte lentă,
- pentru η inadecvat mare, deplasarea către optim, în spațiul parametrilor, se realizează cu oscilații (eventual cu pierderea stabilității algoritmului).

3° În cazul când algoritmul se încheie prin satisfacerea condiției (χ1), acuratețea parametrilor furnizați $\boldsymbol{W}_f^1\in\mathbb{R}^{m\times n}$, $\boldsymbol{W}_f^2\in\mathbb{R}^{p\times m}$ (ponderi) şi $\boldsymbol{b}_f^1\in\mathbb{R}^m$, $\boldsymbol{b}_f^2\in\mathbb{R}^p$ (deplasări) depinde de valoarea pragului ε stabilit pentru eroarea pătratică globală. Valoarea finală a lui J (implicit alegerea valorii de prag ε) trebuie corelată cu magnitudinea valorilor țintă $\boldsymbol{z}_i$, $i=\overline{1,N}$. De exemplu, dacă $\boldsymbol{z}_i$ este de ordinul 10^6 în unitățile de măsură, eroarea poate fi aleasă de ordinul 10^4-10^3; nu vom utiliza drept criteriu de antrenare un prag de ordinul 10^{-2} pentru J deoarece acesta nu ar putea fi atins practic niciodată.

4° Suprafața erorii $J(\boldsymbol{v})$ definită prin (10) uzual posedă *minime locale* în care căutarea minimului poate eşua, dacă rata de învățare are o valoare prea mică. Astfel de minime locale nu pot fi părăsite decât prin schimbarea valorii ratei de învățare, sau prin restartarea algoritmului cu alte valori inițiale ale parametrilor.

5^0 Numărul de neuroni din stratul de intrare se alege în funcție de "gradul de neliniaritate" care se constată la funcția φ din examinarea eşantioanelor (5.1.4). În alegerea arhitecturii rețelei (adică a numărului de neuroni de pe primul strat) pot părea două arhitecturi care trebuie verificate şi eliminate:

a. Subaproximarea (eng. *underfitting*): care are loc pentru un număr prea mic de neuroni. În cazul $n = p = 1$, graficul realizat de rețea nu prezintă un număr suficient de puncte de inflexiune. Pentru a asigura calitatea aproximării este suficient la J să fie foarte mic (nu va fi de aceeaşi valoare ca atunci când trec prin vecinătatea tuturor punctelor).
b. Supraaproximarea (eng. *overfitting*): se alege un număr mare de neuroni în cazul funcției scalare $n = p = 1$. Numărul excesiv de neuroni creează prea multe puncte de inflexiune, care nu ar putea exista pentru un fenomen real.

Alegerea adecvată a numărului de neuroni se bazează pe experiență şi, eventual, pe teste de factură încercare-eroare (eng. *trial and error*) efectuate pe problema concretă. ■

5.2.4. Strategii de creştere a performanțelor antrenării prin propagare inversă

Pentru îmbunătățirea convergenței algoritmului de antrenare se utilizează modificări ale iterațiilor standard de tipul (5.2.18) şi (5.2.19), modificări ce au drept efect creşterea vitezei de căutare şi, totodată, fac posibilă evitarea situațiilor patologice de eşuare în minime locale (Sima, Varga, 1986). Astfel de modificări au drept fundament teoretic diferite maniere de a interveni în iterația standard a metodei de gradient (5.2.2).

În scopul prezentării strategiilor de creştere a performanțelor antrenării prin propagare inversă, vom rescrie iterațiile (5.2.18) şi (5.2.19) în maniera compactă oferită de notațiile matriceale (5.2.26) şi (5.2.27) adică:

- pentru stratul de ieşire:

$$\tilde{\boldsymbol{W}}^2_{\text{new}} = \tilde{\boldsymbol{W}}^2_{\text{old}} + \eta \boldsymbol{D}_{\tilde{\boldsymbol{W}}^2}; \tag{5.2.41}$$

- pentru stratul de intrare:

$$\tilde{\boldsymbol{W}}^1_{\text{new}} = \tilde{\boldsymbol{W}}^1_{\text{old}} + \eta \boldsymbol{D}_{\tilde{\boldsymbol{W}}^1}. \tag{5.2.42}$$

Considerând iterația curentă ca fiind cea de-a k-a în procesul de antrenare, putem reformula (5.2.41) şi (5.2.42) într-o manieră generală:

$$\tilde{\boldsymbol{W}}^2(k+1) = \tilde{\boldsymbol{W}}^2(k) + \tilde{\boldsymbol{M}}^2(k), \tag{5.2.43}$$

$$\tilde{\boldsymbol{W}}^1(k+1) = \tilde{\boldsymbol{W}}^1(k) + \tilde{\boldsymbol{M}}^1(k), \tag{5.2.44}$$

unde matricele $\tilde{\boldsymbol{M}}^2(k)$ şi $\tilde{\boldsymbol{M}}^1(k)$ definesc schimbările parametrilor impuse de procesul de căutare la iterația curentă:

$$\tilde{\boldsymbol{M}}^2(k) = \eta \boldsymbol{D}_{\tilde{\boldsymbol{W}}^2}(k), \tag{5.2.45}$$

$$\tilde{\boldsymbol{M}}^1(k) = \eta \boldsymbol{D}_{\tilde{\boldsymbol{W}}^1}(k). \tag{5.2.46}$$

În relațiile (5.2.45) şi (5.2.46) matricele $\boldsymbol{D}_{\tilde{\boldsymbol{W}}^2}(k)$ şi $\boldsymbol{D}_{\tilde{\boldsymbol{W}}^1}(k)$ sunt date de (5.2.15) şi respectiv (5.2.16) calculate cu vectorii delta de la iterația *k*, adică:

$$\boldsymbol{D}_{\tilde{\boldsymbol{W}}^2}(k) = \boldsymbol{\Delta}^2(k)\left(\tilde{\boldsymbol{X}}^2\right)^T, \tag{5.2.47}$$

şi, respectiv:

$$\boldsymbol{D}_{\tilde{\boldsymbol{W}}^1}(k) = \boldsymbol{\Delta}^1(k)\tilde{\boldsymbol{X}}^T. \tag{5.2.48}$$

Strategiile de creştere a performanțelor antrenării prin propagare inversă se bazează pe schimbări ale modului de calcul dat de (5.2.45) şi (5.2.46) pentru matricele $\tilde{\boldsymbol{M}}^2(\mathrm{k})$ şi $\tilde{\boldsymbol{M}}^1(\mathrm{k})$, ce operează în iterația *k* descrisă prin egalitățile (5.2.43) şi (5.2.44).

A. Propagarea inversă cu moment

Introducerea *momentului* revine la a înlocui (5.2.45) şi (5.2.46) cu următorul mod de calcul al matricelor $\tilde{\boldsymbol{M}}^2(k)$ şi $\tilde{\boldsymbol{M}}^1(k)$:

$$\tilde{\boldsymbol{M}}^2(k) = m\tilde{\boldsymbol{M}}^2(k-1) + (1-m)\eta\boldsymbol{D}_{\tilde{\boldsymbol{W}}^2}(k), \tag{5.2.49}$$

şi respectiv:

$$\tilde{\boldsymbol{M}}^1(k) = m\tilde{\boldsymbol{M}}^1(k-1) + (1-m)\eta\boldsymbol{D}_{\tilde{\boldsymbol{W}}^1}(k), \tag{5.2.50}$$

unde momentul m are o valoare subunitară apropiată de 1.

Prin aceasta, direcția de căutare curentă în procesul de optimizare este exprimată drept o *combinație liniară* între direcția gradientului la pasul curent (dată de termenii de pe poziția a doua a sumelor din (5.2.49) şi (5.2.50)) şi direcția de căutare la pasul precedent (dată de termenii de pe prima poziție a sumelor din (5.2.49) şi (5.2.50)). Cu alte cuvinte, căutarea se realizează memorând, de la un pas la altul, vechea direcție, de care se va ține seama (ponderat prin *m*) în stabilirea noii direcții.

B. Propagarea inversă cu rată de învățare adaptivă

Utilizarea ratei de învățare adaptive revine la a înlocui, în (5.2.45) şi (5.2.46), factorul constant η printr-o valoare $\eta(k)$, ajustabilă de la o iterație la alta, adică (5.2.45) şi (5.2.46) se transformă în:

$$\tilde{\boldsymbol{M}}^2(k) = \eta(k)\boldsymbol{D}_{\tilde{\boldsymbol{W}}^2}(k), \tag{5.2.51}$$

şi, respectiv:

$$\tilde{\boldsymbol{M}}^1(k) = \eta(k)\boldsymbol{D}_{\tilde{\boldsymbol{W}}^1}(k). \tag{5.2.52}$$

Prin aceasta, $\eta(k)$ poate fi crescut atunci când, prin ajustarea parametrilor, valoarea funcției obiectiv *J* din (5.2.1) scade semnificativ. Astfel, antrenarea se poate realiza cu o viteză mai mare, atâta timp cât direcția de căutare este favorabilă. Când căutarea eşuează în descreşterea lui *J* din (5.2.1), rata $\eta(k)$ urmează a fi micşorată.

C. Propagarea inversă cu moment şi rată de învăţare adaptivă

Prin această strategie se combină avantajele celor două procedee prezentate anterior. Astfel, din (5.2.49) şi (5.2.51) va rezulta, pentru stratul de ieşire, forma tipică a iteraţiei k sub forma:

$$\tilde{\boldsymbol{M}}^2(k) = m\tilde{\boldsymbol{M}}^2(k-1) + (1-m)\eta(k)\boldsymbol{D}_{\tilde{\boldsymbol{W}}^2}(k). \tag{5.2.53}$$

Analog din (5.2.50) şi (5.2.52) va rezulta forma tipică a iteraţiei k pentru ajustarea parametrilor de pe stratul de intrare:

$$\tilde{\boldsymbol{M}}^1(k) = m\tilde{\boldsymbol{M}}^1(k-1) + (1-m)\eta(k)\boldsymbol{D}_{\tilde{\boldsymbol{W}}^1}(k). \tag{5.2.54}$$

5.2.5. Alţi algoritmi de antrenare pentru reţele multistrat

După cum am precizat anterior, antrenarea reţelei neurale constă în rezolvarea unei probleme de minimizare corespunzătoare unei funcţii obiectiv de tip eroare pătratică medie (având expresia din (5.1.9)). Am mai insistat asupra faptului că ceea ce, în limbajul reţelelore neurale, este cunoscut sub denumirea de "propagare inversă" reprezintă un algoritm de minimizare de tip gradient cu descreştere maximă (steepest descent), dacă utilizăm exprimările mai generale specifice tehnicilor de optimizare. Altfel spus, propagarea inversă este un algoritm de tip gradient cu descreştere maximă, aranjat într-o formulare matematică specială, asociată tipului particular de aplicaţii care descriu funcţionarea reţelelor neurale (şi, implicit, care contribuie la definirea funcţiei obiectiv).

Acest mod de înţelegere a algoritmului de antrenare al reţelei neurale lasă loc unei întrebări fireşti. Pentru antrenarea unei reţele neurale pot fi utilizate şi alte metode de minimizare (împrumutate din "recuzita" tehnicilor de optimizare)?

Răspunsul este afirmativ. În prezent, există algoritmi de antrenare ai reţelelor neurale care au fost obţinuţi (adaptaţi) din metodele de otimizare cele mai eficiente (ca de exemplu metoda Levenberg-Marquardt, metodele bazate pe direcţii conjugate etc.). În general vorbind, metodele amintite prezintă performanţe numerice mult mai bune decât gradientul cu descreştere maximă, motiv pentru care utilizarea lor în antrenarea reţelelor neurale este mai avantajoasă decât propagarea inversă. Cu toate acestea, atragem atenţia că propagarea inversă reprezintă un subiect tratat de orice manual / monografie referitoare la reţele neuronale, întrucât deţine o "semnificaţie istorică" pentru dezvoltarea domeniului, fiind strâns legată de începuturile utilizării reţelelor multistrat.

Pentru a aprofunda problematica metodelor de optimizare implementate drept algoritmi de antrenare supervizată a reţelelor neurale, cititorul este invitat să parcurgă Anexa A a prezentei cărţi.

5.3. Utilizarea reţelelor feedforward cu trei straturi

Reţelele feedforward cu trei straturi (primele două cu noduri sigmoidale şi cel de ieşire cu noduri liniare) posedă proprietatea de aproximare discutată în paragraful 5.1. Uzual se face apel la astfel de reţele atunci când funcţia ce trebuie aproximată φ din (5.1.2) prezintă moduri de variaţie mai complicate, puse în evidenţă de eşantioanele (5.1.4).

Antrenarea se realizează prin metoda propagării inverse organizată pe cele trei straturi prin generalizarea algoritmului prezentat în paragraful 5.2.2 aplicând *regula delta* succesiv celor trei straturi. În acest caz, vor trebui calculați *vectorii şi matricele delta* pentru toate cele trei straturi, într-o manieră analogă cu (5.2.15) – (5.2.19).

5.4. Utilizarea rețelelor feedforward în probleme de clasificare

După cum a fost prezentat în capitolul 3, rețelele cu funcții de activare treaptă au proprietatea de a separa spațiul ieşirilor prin frontiere de tip hiperplane. Datorită acestei proprietăți, rețelele cu funcție de activare treaptă pot fi utilizate în probleme de clasificare. Un asemenea clasificator poate fi utilizat însă numai pentru clase liniar separabile. În plus, o rețea neurală de tip perceptron cu un strat nu este capabilă să realizeze o clasificare pe problema simplă de tip XOR (care necesită o rețea cu două straturi, după cum se arată în problema 7.5). Detalii privind problematica clasificării sunt accesibile în anexa B.

Aceste deficiențe pot fi depăşite dacă se utilizează rețele neurale cu funcții de activare de tip sigmoidal având o arhitectură cu două straturi. Pentru utilizarea unei astfel de arhitecturi, ca şi în cazul rețelelor cu funcții de activare de tip treaptă, se parcurg două etape:

(i) antrenarea rețelei neurale pe baza unui lot reprezentativ, capabil să dea informații relevante despre clase;
(ii) utilizarea rețelei antrenate drept clasificator.

În continuare vom considera rețele neurale cu funcții de activare de tip sigmoid unipolar, dar în acelaşi măsură pot fi utilizate şi noduri bipolare. Să presupunem că proiectăm o aplicație pentru a realiza clasificarea punctelor dintr-un domeniu din $\mathbb{R}^n$ în s clase.

O primă variantă este aceea de a considera o rețea cu n intrări şi s ieşiri. Clasei $\mathscr{C}_j$ îi asociem vectorul $\underline{\boldsymbol{y}}_j = [0 \ldots \underset{j}{1} \ldots 0]^T \in \mathbb{R}^s$, pentru $j = 1, 2, ..., s$. Întrucât ieşirile rețelei sunt date de funcții sigmoidale, luând valori în intervalul deschis $(0, 1)$ şi doar tinzând asimptotic către capetele acestuia, în vectorii țintă nu pot fi utilizate valorile binare 0 şi 1. Pentru ca neuronii din stratul de ieşire să furnizeze valori foarte apropiate fie de 0 fie de 1, ar trebui ca ponderile acestora să ia valori foarte mari. Din acest motiv se preferă relaxarea formei binare cu 0 şi 1 a vectorilor țintă, utilizând în locul lor 0 valorile ε şi, respectiv, $1-\varepsilon$, cu $\varepsilon > 0$ suficient de mic (uzual ε se ia de ordinul sutimilor). Astfel, clasei $\mathscr{C}_j$ îi asociem vectorul $\tilde{\boldsymbol{y}}_j = [\varepsilon \ldots \underset{j}{1-\varepsilon} \ldots \varepsilon]^T \in \mathbb{R}^s$, pentru $j = 1, 2, ..., s$. Antrenarea rețelei neurale se va realiza pe baza perechilor $(\boldsymbol{x}^i, \boldsymbol{z}^i)$, $i = \overline{1, N}$, cu $\boldsymbol{x}^i \in \mathbb{R}^n$ şi $\boldsymbol{z}^i \in \mathbb{R}^s$, $\boldsymbol{z}^i = \tilde{\boldsymbol{y}}_j$ dacă $\boldsymbol{x}^i \in \mathscr{C}_j$. Prin perechile $(\boldsymbol{x}^i, \boldsymbol{z}^i)$, $i = \overline{1, N}$, definim o problemă de tip aproximare, contextul teoretic dezvoltat de Cybenko ne asigură succesul antrenării (dacă se consideră un număr suficient de neuroni pe primul strat).

Dacă numărul de clase este o putere a lui 2, de forma $s = 2^p$, se poate utiliza o a doua variantă, similară celei prezentate în capitolul 3. Se consideră o rețea cu n intrări şi p ieşiri. Clasei $\mathscr{C}_j$ i se asociază un vector $\underline{\boldsymbol{y}}_j \in \mathbb{R}^p$, $j = 1, \ldots, 2^p$, reprezentând codul binar al indicelui

j al clasei, apoi în locul valorilor 0 şi 1 se utilizează ε şi, respectiv, $1-\varepsilon$, cu $\varepsilon > 0$ suficient de mic, construind vectorul $\tilde{\boldsymbol{y}}_j \in \mathbb{R}^p$. Antrenarea reţelei neurale se va realiza pe baza perechilor $(\boldsymbol{x}^i, \boldsymbol{z}^i)$, $i = \overline{1,N}$, cu $\boldsymbol{x}^i \in \mathbb{R}^n$ şi $\boldsymbol{z}^i \in \mathbb{R}^p$, $\boldsymbol{z}^i = \tilde{\boldsymbol{y}}_j$ dacă $\boldsymbol{x}^i \in \mathscr{C}_j$.

În cazul în care nodurile reţelei neurale au noduri bipolare, vectorii ţintă se formează cu elementele $-1+\varepsilon$ şi $1-\varepsilon$.

Practica a arătat că stratul de ieşire al reţelei poate avea şi noduri de tip liniar. În aceste condiţii apartenenţa la o clasă în etapa de învăţare se va realiza cu vectori ţintă formaţi cu valori de 0 şi 1 ferme (vectorii notaţi în paragrafele anterioare cu $\underline{\boldsymbol{y}}_j$). Datorită utilizării funcţiei de activare liniară pe ultimul strat, ieşirile reţelei pot avea şi valori mai mici decât 0 sau mai mari decât 1.

Indiferent de tipul funcţiilor de activare utilizate pe stratul de ieşire al reţelei neurale utilizată pentru clasificare, decizia privind apartenenţa unui vector prototip la o anumită clasă va fi luată pe baza distanţei dintre ieşirea reţelei neurale şi vectorii ţintă corespunzători claselor considerate. Un vector prototip va fi atribuit clasei al cărei vector prototip este situate la distanţa cea mai mică de ieşirea reţelei neurale utilizate drept clasificator.

La fel ca şi în probleme de aproximare, în faza de antrenare a reţelei în scopul recunoaşterii de trăsături sau forme trebuie să fie prezentate reţelei toate elementele semnificative din punct de vedere al aplicaţiei. Reţeaua va trebui "să vadă" eşantioanele reprezentative pentru toate clasele care apar în aplicaţii (numărul de clase şi semnificaţia claselor este o problemă care precede antrenarea reţelei şi trebuie complet soluţionată înainte de momentul începerii antrenării). Adică reţeaua nu este implicată în a decide câte clase distincte sunt.

Valorile concrete pe care le furnizează clasificatorul pe ieşirea reţelei neurale, pot fi utilizate pentru a informa asupra „ calităţii clasificării", (altfel spus, un fel de garanţie pe care o oferă reţeaua că separă (atribuie) corect vectorii de intrare pe clase). Distanţe mari faţă de ε şi 1-ε (în cazul ieşirilor sigmoidale) sau 0, 1 (în cazul ieşirilor liniare) atrag atenţia asupra unor elemente clasificate cu o doză mare de risc (de incertitudine) în cazul că în etapa de învăţare, reţeaua „nu a văzut" eşantioane apropiate ca trăsături cu aceste elemente. Astfel de elemente necesită o altă procedură de clasificare pentru reducerea incertitudinii sau, dacă apariţia lor este frecventă, atrage atenţia asupra necesităţii reantrenării reţelei (de aşa manieră încât reţeaua să aibă cunoştinţă despre trăsăturile aferente elementelor respective).

5.5. Facilităţi software oferite de Neural Network Toolbox

Funcţia newff creează o reţea feed-forward. Numărul de straturi şi de neuroni din stratul de intrare şi din straturile ascunse, precum şi funcţia de activare a neuronilor fiecărui strat sunt specificate de către utilizator în apelul funcţiei. La crearea reţelei utilizatorul trebuie să furnizeze şi rutina ce va fi utilizată pentru antrenarea acesteia precum şi parametrii de antrenare doriţi.

Pentru inițializarea cu valori arbitrare a parametrilor unei rețele (ponderi şi deplasări) se poate utiliza funcția rands. De asemenea, pentru inițializarea parametrilor se poate apela funcția init.

În cazul unui neuron cu o singură intrare, pentru reprezentarea grafică tridimensională a suprafeței erorii pătratice globale (ca funcție de ponderea şi deplasarea neuronului) se pot utiliza funcțiile errsurf şi plotes (apelate în această ordine).

Funcția train antrenează rețeaua conform rutinei de antrenare specificate la creare şi a parametrilor de antrenare aleşi.

Algoritmul de antrenare prin propagare inversă descris mai sus este implementat de rutina 'traingd' (se bazează pe metoda *gradient descent* de minimizare a unei funcții – algoritm lent). Dintre parametrii asociați acestui algoritm amintim net.trainParam.lr (valoarea ratei de învățare utilizată în antrenare), net.trainParam.epochs (numărul de epoci de antrenare), net.trainParam.goal (valoarea de prag impusă pentru criteriul de performanță minimizat net.performFcn) şi net.trainParam.show (numărul de epoci de antrenare după care se prezintă grafic progresul antrenării).

Două variante ale acestui algoritm sunt implementate în funcțiile 'traingdm' (*gradient descent with momentum* – în general, mai rapid decât traingd) şi 'traingdx' (*gradient descent with adaptive learning rate* – de asemenea, mai rapid decât traingd). Metodele de minimizare bazate pe direcții de căutare conjugate gradientului sunt implementate prin funcțiile de antrenare 'traincgf' (metoda Fletcher-Reeves – cel mai mic necesare de memorie), 'traincgp' (metoda Polak-Ribiére – convergență mai rapidă în unele cazuri), 'traincgb' (metoda Powell-Beale – convergență rapidă în general), 'trainscg' (*scaled conjugate gradient* – algoritm foarte bun pentru probleme generale).

Un algoritm de antrenare mai rapid pentru rețele de dimensiuni moderate este algoritmul Levenberg-Marquardt, implementat prin rutina 'trainlm' (prezintă facilități de reducere a memoriei utilizate în cazul când setul de antrenare are lungime mare). Funcția 'trainbr' (*Bayesian regularization*) implementează o variantă modificată a acestui algoritm.

Algoritmul de antrenare de tip quasi-Newton bazat pe metoda Broyden-Fletcher-Goldfarb-Shanno este implementat prin funcția 'trainbfgs' (necesită stocarea aproximării matricei Hessian şi efectuează mai multe calcule la fiecare iterație decât algoritmii de tip gradient descent). Funcția 'trainoss' (metoda *one step secant*) realizează un compromis între metodele de tip gradient conjugat şi cele de tip quasi-Newton.

Pentru simularea rețelei antrenate se poate folosi funcția sim.

Pentru aprofundarea noțiunilor prezentate în acest capitol se recomanda studierea problemelor cu soluții analitice 7.9 – 7.16 şi a problemelor cu soluții numerice 8.12 – 8.16.

Capitolul 6

APLICAȚII ALE REȚELELOR FEEDFORWARD MULTISTRAT ÎN IDENTIFICAREA ŞI CONTROLUL SISTEMELOR DINAMICE

6.1. Identificarea sistemelor dinamice utilizând rețele neurale

Identificarea sistemelor dinamice bazată pe modele neurale s-a conturat ca direcție de cercetare la începutul anilor ’90, drept urmare a investigațiilor matematice asupra proprietăților de aproximare ale rețelelor neuronale. Evoluția noului domeniu a fost impulsionată de contribuțiile remarcabile ale unor nume de prestigiu în Automatică, ca, de exemplu, K. Narendra, K. Parthasarathy, P.J. Gawthorp, T.J. McAvoy, L. Ljung. Pe de altă parte, firmele producătoare de software tehnico-ştiințific au demarat dezvoltarea de facilități specifice rețelelor neuronale, astfel încât apariția în 1992 a primei versiuni a ***Neural Network Toolbox*** încorporată în mediul MATLAB 4.2 a avut un impact major asupra interesului acordat de către automatişti acestei direcții de cercetare.

Modelele generale de identificare parametrică a sistemelor dinamice (liniare sau neliniare) se pot implementa, atât hard cât şi soft, cu ajutorul rețelelor neurale artificiale. Parametrii modelului neural sunt reprezentați de ponderile sinaptice ale neuronilor şi deplasările acestora. Teoria generală privind identificarea sistemelor se poate aplica direct pentru obținerea modelului neural ținând cont de echivalențele:

structura modelului = arhitectura rețelei neurale
estimare = antrenare
validare = generalizare
set de date de estimare = set de date de antrenare
set de date de validare = set de date de generalizare

6.1.1. Structura modelelor neurale

Considerăm un sistem dinamic cu o intrare şi o ieşire (figura 6.1.1) pentru care se măsoară semnalele de intrare u şi de ieşire y la momentele discrete de timp $t_k = kT$, (unde T reprezintă perioada de eşantionare a semnalelor, $k \in \mathbb{N}$), notând $u[k] = u(kT)$, $y[k] = y(kT)$, $k = \overline{1, N}$.

Figura 6.1.1. *Sistem dinamic SISO*

În problema identificării sistemelor dinamice, unde neliniaritatea care trebuie modelată depinde de timp, problema care se pune este cea a reprezentării explicite a timpului în structura reţelei neurale. Pentru ca o reţea neurală să fie dinamică, ea trebuie să posede o memorie. Ideea de la care s-a pornit la începutul anilor 1990 a fost cea de a se utiliza reţele neurale statice împreună cu un sistem extern de întârziere a semnalelor de intrare în reţea, obţinând aşa numitele *reţele neurale cu dinamică externă*.

Un interes practic deosebit îl prezintă obţinerea modelelor neurale de tip ARX (NNARX) în timp discret, cu perioadă de eşantionare dată:

$$\hat{y}[k] = f\left(y[k-1], y[k-2], ..., y[k-n], u[k-d], u[k-d-1], ..., u[k-d-m]\right), \quad (6.1.1)$$

unde $d, n \in \mathbb{N}^*$, $m \in \mathbb{N}$, şi:

$$f : \mathbb{R}^{n+m+1} \to \mathbb{R} \quad (6.1.2)$$

este o aplicaţie (liniară sau neliniară, netedă) ce descrie transferul intrare-ieşire al unei reţele neurale statice cu topologie adecvată. Această funcţie poate fi reprezentată cu ajutorul unei reţele neurale statice cu propagare înainte a semnalului cu funcţii de activare liniare (ADALINE) sau neliniare (MLP). Reţeaua are ca intrări valorile de la momentele anterioare de timp ale intrării şi ieşirii procesului. Valorile semnalelor de intrare şi de ieşire la momentele anterioare de timp formează vectorul *regresorilor* care este notat $\varphi[k] = \left[y[k-1], y[k-2], ..., y[k-n], u[k-d], u[k-d-1], ..., u[k-d-m]\right]^T$ şi se obţine utilizând un sistem de întârziere (*Tapped Delay Line*) plasat în afara reţelei neurale propriu-zise.

Configuraţia prezentată în figura 6.1.2 este cunoscută şi sub denumirea de *modelul serie-paralel* (SPM) deoarece în raport cu intrarea $u[k]$ reţeaua neurală este conectată paralel cu procesul, iar în raport cu ieşirea $y[k]$ conectarea este în serie. Eroarea de predicţie $e[k]$, diferenţa dintre ieşirea reală a procesului $y[k]$ şi ieşirea estimată cu ajutorul reţelei neurale $\hat{y}[k]$, este utilizată pentru antrenarea reţelei aplicând algoritmi de antrenare uzuali, independenţi de timp. În această figură, z^{-1} reprezintă operatorul de întârziere cu un pas, $z^{-1}u[k] = u[k-1]$, iar ZOH notează extrapolatorul de ordin zero (*Zero Order Hold*) utilizat pentru discretizarea, cu perioada de eşantionare dorită, a unui proces continuu.

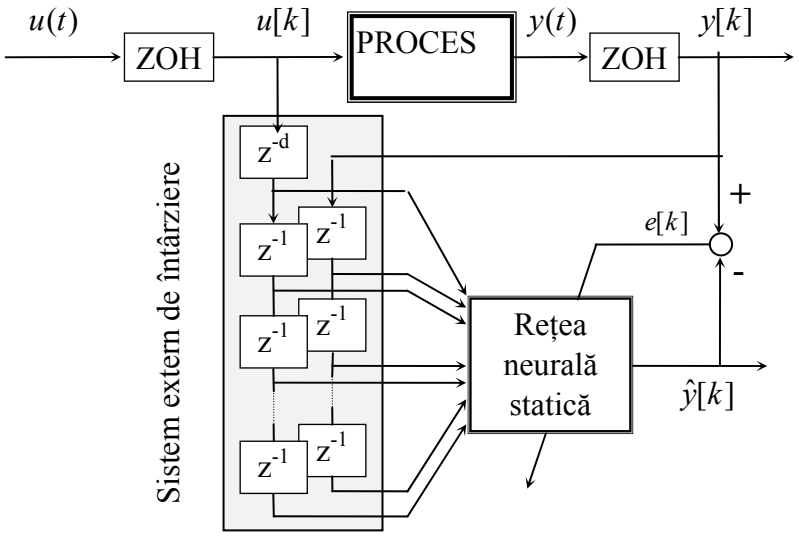


Figura 6.1.2. *Modelul neural de tip ARX (NNARX) (modelul serie-paralel)*

După antrenarea reţelei neurale, modelul identificat poate fi folosit independent de proces, aducând la intrarea reţelei ieşirea acesteia, printr-o conexiune inversă externă. O astfel de configuraţie poartă numele de *modelul paralel* şi este utilizată pentru simularea proceselor dinamice. Configuraţia poate fi utilizată şi în faza de antrenare a reţelei neurale, modelul obţinut în acest fel fiind de fapt modelul neural de tip *output error* (NNOE) (figura 6.1.3)

$$\hat{y}[k] = f\left(\hat{y}[k-1], \hat{y}[k-2], ..., \hat{y}[k-n], u[k-d], u[k-d-1], ..., u[k-d-m]\right). \quad (6.1.3)$$

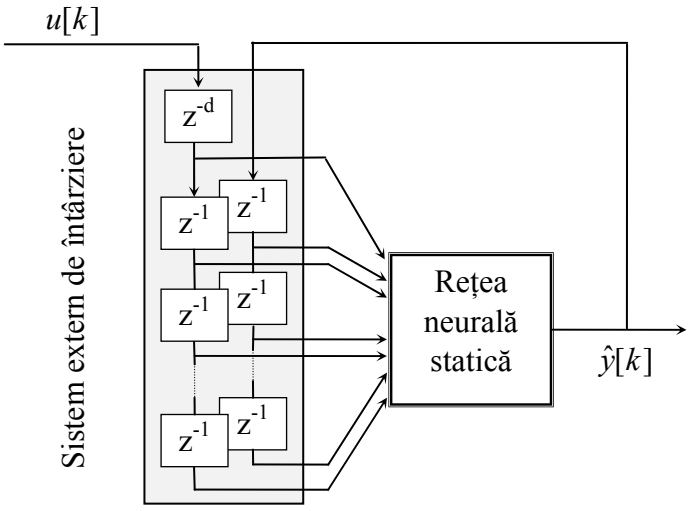


Figura 6.1.3. *Modelul neural OE* (NNOE) (*modelul paralel*)

În mod asemănător se pot obţine şi modelele neurale de tip ARMAX (NNARMAX) sau Box-Jenkins (NNBJ). Dezavantajele modelelor recursive de tip NNOE, NNARMAX sau NNBJ sunt posibila instabilitate a modelului şi necesitatea unui efort suplimentar pentru a calcula gradientul ce intervine în procedura iterativă de antrenare a reţelei.

Pentru reprezentarea modelelor neliniare de tip NNARX cu o singură intrare şi o singură ieşire Narendra şi Parthasarathy (1990) au introdus patru modele. Modelele propuse de ei sunt descrise de ecuațiile:

Modelul 1: $$\hat{y}[k+1]=\sum_{i=0}^{n-1}\alpha_i y[k-i]+g\left(u[k],u[k-1],...,u[k-m+1]\right), \tag{6.1.4}$$

Modelul 2: $$\hat{y}[k+1]=f\left(y[k],y[k-1],...,y[k-n+1]\right)+\sum_{i=0}^{m-1}\beta_i u[k-i], \tag{6.1.5}$$

Modelul 3: $$\hat{y}[k+1]=f\left(y[k],y[k-1],...,y[k-n+1]\right)+g\left(u[k],u[k-1],...,u[k-m+1]\right), \tag{6.1.6}$$

Modelul 4: $$\hat{y}[k+1]=f\left(y[k],y[k-1],...,y[k-n+1],u[k],u[k-1],...,u[k-m+1]\right), \tag{6.1.7}$$

Se presupune că funcțiile *f* şi *g* sunt diferențiabile în raport cu toate argumentele lor. Aceste funcții pot fi aproximate cu ajutorul unor rețele neurale statice. Alegerea modelului adecvat depinde de informația despre procesul considerat disponibilă a priori. În modelele 1 (figura 6.1.4) şi 2 (figura 6.1.5) se presupune că neliniaritatea procesului implică numai intrarea, respectiv numai ieşirea.

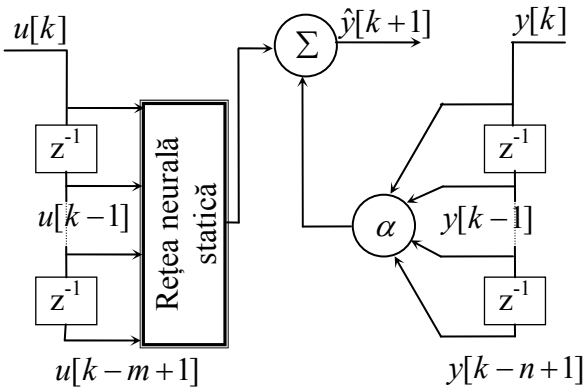


Figura 6.1.4. *Reprezentarea modelului 1.*

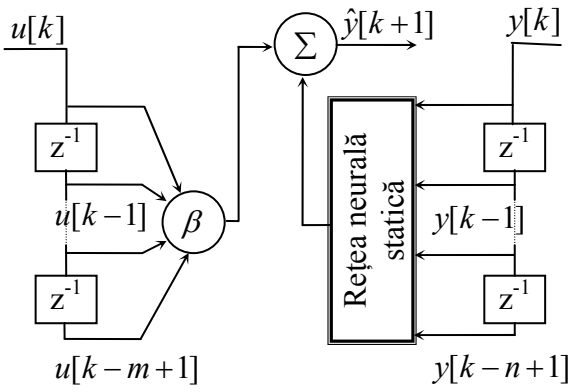


Figura 6.1.5. *Reprezentarea modelului 2.*

Modelul 3 (figura 6.1.6) rezultă ca suma a două funcții neliniare distincte care depind numai de semnalul de intrare, respectiv ieşire. Modelul 4 (figura 6.1.7), cel mai general, reprezintă proiecția neliniară atât a intrărilor cât şi a ieşirilor procesului de identificat.

Modelele de tip NNARX pot fi utilizate pentru probleme simple, dar utilizarea lor nu este avantajoasă pentru probleme ce implică dependența semnalului de ieşire al sistemului de un număr mare de valori trecute ale semnalului de intrare. Pentru modelarea unor asemenea sisteme se recomandă utilizarea modelelor recursive.

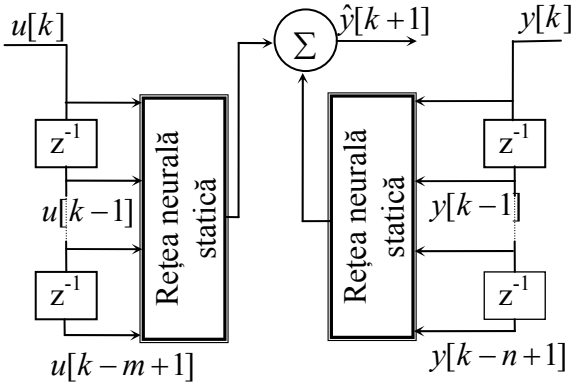


Figura 6.1.6. *Reprezentarea modelului 3*

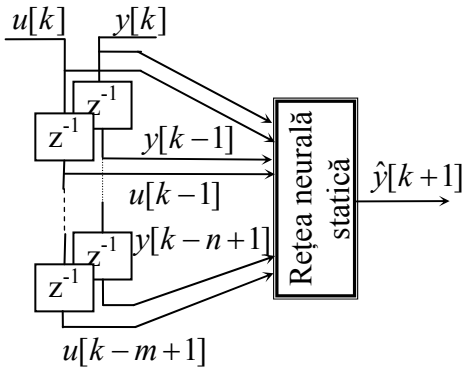


Figura 6.1.7. *Reprezentarea modelului 4*

O altă posibilitate de modelare a unui sistem dinamic neliniar revine la a construi în prima etapă un model liniar al sistemului. Reziduurile obţinute ca diferenţă între ieşirile actuale ale sistemului şi ieşirile estimate pe baza modelului liniar vor conţine evident neliniarităţile nemodelate. Pentru a modela aceste neliniarităţi se construieşte apoi un model neural având ca intrări intrările sistemului şi reziduurile.

6.1.2. Estimarea modelelor neurale

Aplicaţia f din relaţia (6.1.2) este parametrizată de ponderile şi deplasările reţelei. Indiferent de topologia reţelei, identificarea pe baza aşa-numitei scheme serie-paralel (figura 6.1.2) asigură stabilitatea algoritmului iterativ de minimizare a funcţiilor obiectiv definite cu ajutorul erorii $e[k] = y[k] - \hat{y}[k-1])$.

Formularea concretă a funcţiilor obiectiv este corelată cu modul de obţinere şi exploatare a datelor experimentale corespunzătoare procesului. În acest sens, literatura evidenţiază două categorii de metode de antrenare a reţelei neuronale din schema serie-paralel:

- pe lot (*batch*) sau *off-line*, care utilizează funcţii obiectiv de forma (până la o constantă multiplicativă):

$$J = \frac{1}{N}\sum_{k=1}^{N} e^2[k] \le \varepsilon\,, \tag{6.1.8}$$

  unde N notează numărul total de elemente disponibile în lot şi ε este valoarea de prag dorită;

- pe eşantioane (*pattern*) sau *on-line*, care utilizează funcţii obiectiv de forma (până la o constantă multiplicativă):

$$J(i) = \frac{1}{N_E}\sum_{k=i}^{i+N_E-1} e^2[k] \le \varepsilon,\quad i \in \mathbb{N}\,, \tag{6.1.9}$$

  unde N_E notează numărul de eşantioane conţinute în fereastra mobilă de selectare a elementelor, i este indicele iteraţiei curente şi ε este valoarea de prag dorită.

Minimizarea operează în spațiul parametrilor rețelei (ponderi şi deplasări) şi poate fi condusă prin tehnicile clasice de optimizare fără restricții, adaptate la specificul organizării matriceal-vectoriale a parametrilor aplicației *f*. În limbajul propriu rețelelor neuronale, determinarea parametrilor funcției *f* ca rezultat al minimizării funcțiilor obiectiv (5.6) sau (5.7) (estimarea modelului parametric) este referită drept proces de antrenare sau învățare.

Observație:

Determinarea modelului neural urmează paşii prezentați în capitolul 2: alegerea regresorilor modelului, a tipului neuronilor din structura rețelei neurale, alegerea numărului de neuroni din stratul intern şi apoi determinarea ponderilor rețelei (adică determinarea parametrilor modelului). Principiul care se aplică este de a porni de la simplu la complex (lama lui Occam): se încearcă întâi cu un număr redus de regresori şi de neuroni în stratul de intrare şi apoi se creşte progresiv numărul neuronilor până la un număr rezonabil. Dacă nu se determină un model suficient de bun se creşte numărul regresorilor şi se încearcă, din nou, valori crescute progresiv ale numărului de neuroni. Dacă nici în acest caz nu se determină un model corespunzător, se alege un alt tip de model pentru sistemul respectiv. ■

6.2. Controlul predictiv al sistemelor dinamice neliniare utilizând modele neurale

Controlul predictiv este de acum un standard larg utilizat în industrie şi un număr foarte mare de algoritmi au fost prezentați în literatura de specialitate ca de exemplu: *Generalized Predictive Control* – GPC, *Self - Adaptive Control* – EPSAC şi *Unified Predictive Control* – UPC. Marele avantaj al folosirii controlului predictiv provine din utilizarea de către regulator a unui model exact al procesului real, ceea ce duce la îmbunătățirea calității controlului. Scopul algoritmilor de control predictiv bazați pe model (eng. *Model Based Predictive Control* – MBPC) îl constituie găsirea unei secvențe de intrare ce va fi aplicată procesului controlat astfel încât ieşirea sa să urmărească cu eroare cât mai mică o traiectorie dorită (referința) pe durata unui orizont de timp prestabilit (Lazăr, 1999).

Principiul de bază al algoritmilor de control predictiv constă în prezicerea ieşirii procesului, pe baza unui model (obținerea predictorilor), şi pe minimizarea unei anumite funcții de cost. Diferențele dintre diverşii algoritmi sunt date de modelul folosit de controller şi de forma funcției de cost. În cazul sistemelor liniare, majoritatea modelelor dinamice utilizate în controlul predictiv se obțin prin teste aplicate proceselor sau prin metode de identificare. Pentru controlul predictiv neliniar testarea şi identificarea proceselor devin mult mai dificile. Datorită proprietății rețelelor neurale de a fi aproximatori generali, utilizarea unui model neuronal pentru obținerea predictorilor prezintă un foarte mare interes.

O reprezentare grafică a strategiei de control predictiv este prezentată în figura 6.2.1. La fiecare moment de eşantionare k se determină strategia optimă de control, fiind calculate valorile viitoare ale intrării $u[k+1]$, $u[k+2]$, …, $u[k+N_u]$ (N_u – orizontul de control), astfel

încât ieşirea procesului y să urmărească cu eroare cât mai mică referinţa r pe intervalul de predicţie $[N_1, N_2]$ (N_1 – orizontul minim de predicţie, N_2 – orizontul de predicţie).

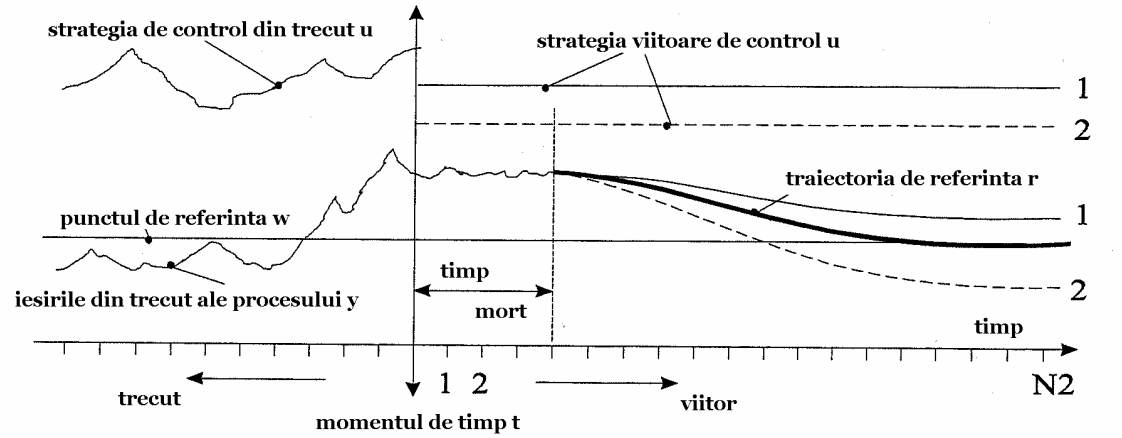


Figura 6.2.1. *Strategia controlului predictiv bazat pe model.*

Valorile viitoare ale ieşirii procesului sunt determinate folosind un model neural, iar pe baza erorilor între ieşirile prezise şi referinţă este calculată o funcţie de cost. Această funcţie de cost urmează a fi minimizată cu scopul obţinerii unei strategii de control optime ce vor fi aplicate procesului. Una din formele posibile ale funcţiei de cost, folosită în majoritatea algoritmilor de control predictiv, este dată de relaţia:

$$J = \sum_{i=N_1}^{N_2} \left(r[k+i] - y[k+i]\right)^2 + \lambda \sum_{i=1}^{N_u} \left(u[k+i] - u[k+i-1]\right)^2 . \qquad (6.2.1)$$

De multe ori, în practică, ponderea λ asupra comenzii are valoarea 0, iar orizontul minim de predicţie (N_1) ia valoarea 1.

Funcţia de cost J urmează a fi minimizată la orice moment de timp k în funcţie de vectorul $\boldsymbol{u} = \left[u[k+1], u[k+2], \ ..., u[k+N_u]\right]^T$ ce conţine valorile viitoare ale intrării.

Modelul neural folosit în calculul ieşirilor viitoare ale procesului este un model de tip ARX de forma (6.1.1). Pentru vectorii conţinând intrările, respectiv ieşirile viitoare ale procesului vom folosi următoarele notaţii:

$$\begin{aligned} \boldsymbol{u}[k+i-d] &= \left[u[k+i-d], u[k+i-d-1], \ ..., u[k+i-d-m]\right]^T , \\ \boldsymbol{y}[k+i-1] &= \left[y[k+i-1], y[k+i-2], \ ..., y[k+i-n]\right]^T , \end{aligned} \qquad (6.2.2)$$

unde $i = N_1, ..., N_2$, reprezintă ordinul predictorului, k este momentul curent de timp, iar d, m şi n reprezintă parametrii consideraţi pentru modelul NNARX antrenat: d - timpul mort, m şi n ordinele de întârziere ale intrării, respectiv ieşirii. Aplicând vectorii (6.2.2) modelului neural, se obţine predictorul de ordin i al ieşirii, $y[k+i]$, după cum rezultă şi din figura 6.2.2.

În calculul predictorilor ieşirii, daca orizontul de predicţie este mai mare decât orizontul de control sumat cu timpul mort considerat în modelul procesului ($N_2 > N_u + d$), după iteraţia $N_u + d$, intrarea va fi considerată constantă, având valoarea $u[k+N_u]$.

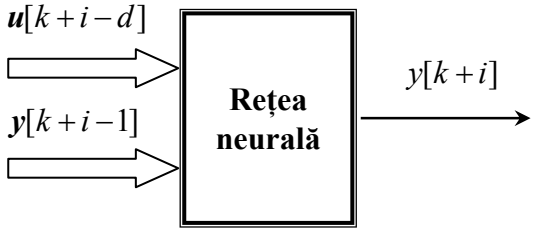


Figura 6.2.2. *Calculul predictorilor neurali.*

Algoritmul iterativ de control predictiv poate fi uşor înțeles prin descrierea paşilor ce sunt parcurşi la orice moment de timp (iterație) *k*:

- în iterația anterioară (momentul de timp $k-1$), prin minimizarea funcției de cost, a fost obținut vectorul comenzii $\boldsymbol{u}_{\text{old}} = \left[u[k], u[k+1],\ ...,\ u[k+N_u-1]\right]^T$;
- primul element al acestui vector, $u[k]$, reprezintă intrarea procesului la momentul de timp curent;
- sunt calculați predictorii ieşirii $y[k+i]$ de ordin $i = N_1, ..., N_2$, folosind ca intrări ale modelului neural perechile de vectori $\boldsymbol{u}[k+i-d]$ şi $\boldsymbol{y}[k+i-1]$;
- se calculează funcția de cost J la pasul curent, după formula (6.2.1);
- noua secvență optimă de control se obține prin minimizarea funcției de cost J în raport cu vectorul comenzii $\boldsymbol{u}_{\text{new}} = \left[u[k+1], u[k+2],\ ...,\ u[k+N_u]\right]^T$.

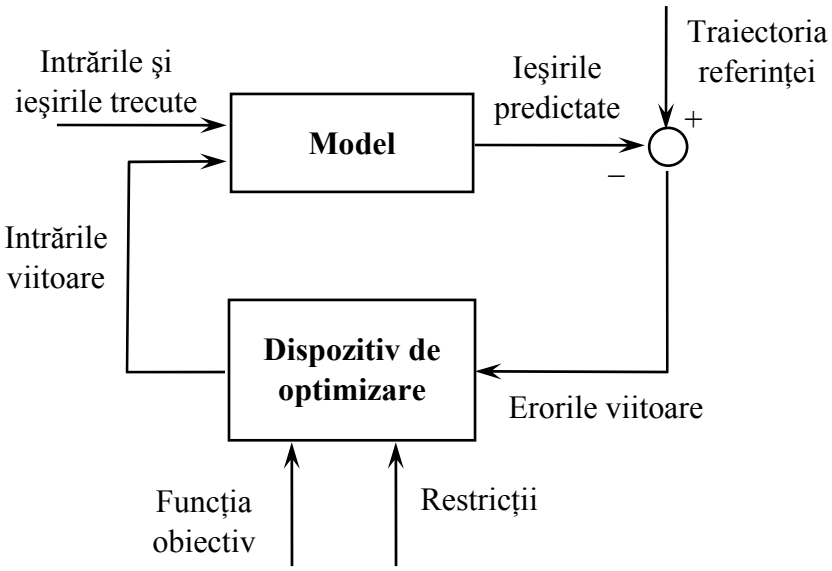


Figura 6.2.3: *Structura de bază a unui controller predictiv*

La momentul de timp următor (*k*+1), primul element din vectorul $\boldsymbol{u}_{\text{new}}$, $u[k+1]$, este aplicat la intrarea procesului şi algoritmul de control predictiv continuă cu iterația *k*+1. La startarea algoritmului, utilizatorul trebuie să aleagă valoarea inițială $u[0]$ a intrării procesului.

Intrările ce urmează a fi aplicate procesului real pot fi supuse unor restricţii ce decurg din natura fizică a acestuia. Unii algoritmi de control predictiv permit ca aceste restricţii să fie luate în consideraţie la minimizarea funcţiei de cost.

Având în vedere principiile MBPC prezentate, structura unui bloc de control ce implementează un astfel de algoritm poate fi prezentată ca în figura 6.2.3:

Viteza de calcul a unui astfel de controller trebuie să fie suficient de ridicată, pentru a permite efectuarea paşilor corespunzători unei iteraţii a algoritmului de control într-un timp mai mic decât perioada de eşantionare aleasă pentru interfaţarea cu procesul real.

6.3. Blocuri Simulink pentru aplicaţii ale reţelelor neurale în identificarea şi conducerea sistemelor

Figura 6.3.1 prezintă o bibliotecă de blocuri Simulink pentru identificarea, simularea şi controlul predictiv al sistemelor pe baza modelelor neurale dezvoltată în cadrul Catedrei de Automatică şi Informatică Aplicată de la Facultatea de Automatică şi Calculatoare din Iaşi (Kloetzer *et al.*, 2001), (Kloetzer, Păstrăvanu, 2004).

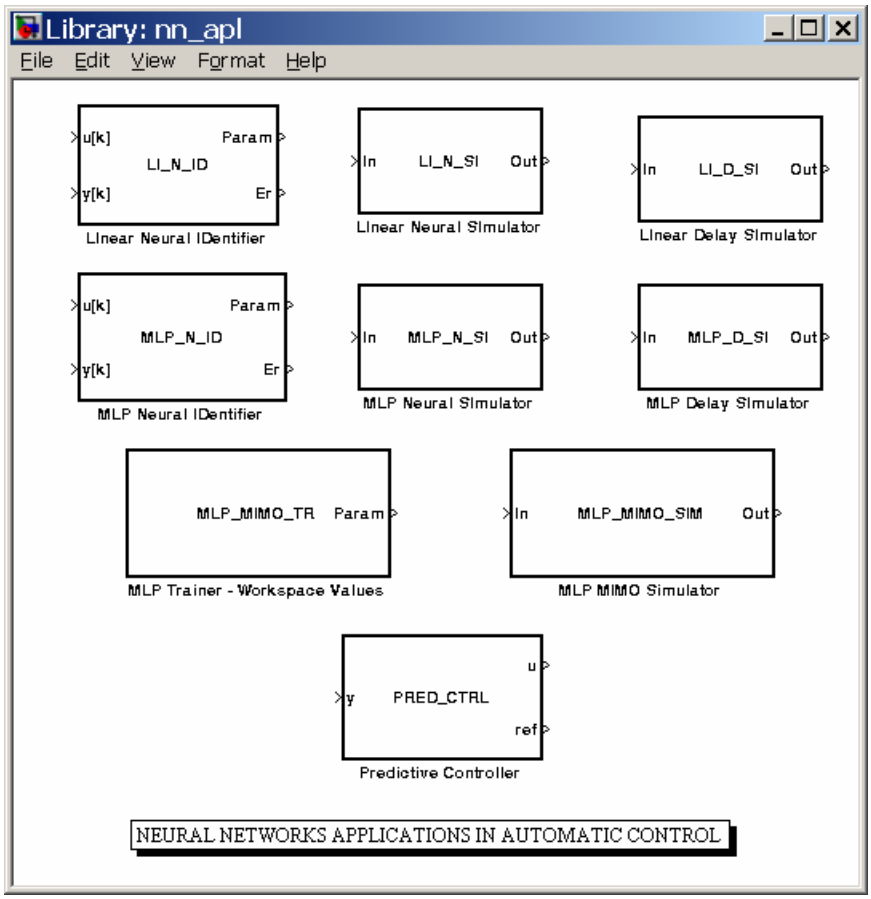


Figura 6.3.1. *Bibliotecă de blocuri Simulink pentru identificarea, simularea şi controlul predictiv al sistemelor pe baza modelelor neurale.*

Dezvoltarea acestei biblioteci de blocuri Simulink pentru identificarea, simularea şi controlul predictiv al sistemelor pe baza modelelor neurale a fost motivată de dorința de a uşura descrierea procesului supus identificării (aşa cum se va vedea în continuare), lăsând utilizatorului, în acelaşi timp, flexibilitate în organizarea simulării. Mai mult chiar, posibilitatea executării simulărilor în timp real, prin utilizarea Real-Time Workshops, extinde aria de aplicabilitate a procedurilor de identificare la procese reale, interfațate adecvat prin cartele de achiziție.

Biblioteca Simulink din figura 6.3.1 constă din blocuri ce permit identificarea şi simularea modelelor neurale cu o singură intrare şi o singură ieşire (SISO) de forma (6.1.1) şi anume:

(*i*) pentru funcția *f* liniară:
- blocul ***LI_N_ID*** (*LInear Neural IDentifier*) folosit pentru antrenarea unei rețele ADALINE,
- blocul ***LI_N_SI*** (*LInear Neural SImulator*) folosit simularea transferului intrare-ieşire a unui model neural cu topologie ADALINE, identificat anterior,
- blocul ***LI_D_SI*** (*LInear Delay SImulator*) folosit pentru simularea transferului intrare-ieşire a unui model neural de tip ADALINE, cu timp mort, identificat anterior;

(*ii*) pentru funcția *f* neliniară:
- blocul ***MLP_N_ID*** (*MLP Neural IDentifier*) serveşte la antrenarea rețelelor de tip MLP,
- blocul ***MLP_N_SI*** (*MLP Neural SImulator*) serveşte pentru simularea transferului intrare-ieşire a unui model neural cu topologie MLP, identificat anterior,
- blocul ***MLP_D_SI*** (*MLP Delay SImulator*) utilizat pentru simularea transferului intrare-ieşire a unui model neural cu topologie MLP, cu timp mort, identificat anterior.

Pentru identificarea şi simularea modelelor neurale neliniare cu mai multe intrări şi ieşiri (MIMO), această bibliotecă Simulink mai include:
- blocul ***MLP_MIMO_TR*** (*MLP MIMO Trainer*) serveşte la antrenarea rețelelor de tip MLP utilizând matrice din spațiul de lucru Matlab,
- blocul ***MLP_MIMO_SI*** (*MLP MIMO SImulator*) serveşte pentru simularea transferului intrare-ieşire a unui model neural cu topologie MLP, identificat anterior, cu ajutorul blocului ***MLP_MIMO_TR***.

În plus, blocul ***PRED_CTRL*** permite implementarea unui controller predictiv pentru un sistem SISO utilizând un model neural, identificat anterior cu folosind blocul ***MLP_N_ID***.

Casetele de dialog ale blocurilor ***LI_N_ID*** şi ***MLP_N_ID*** pentru identificarea modelelor SISO sunt prezentate în figurile 6.3.2 şi, respectiv, 6.3.4. Aceste blocuri permit atât antrenarea pe lot (*off-line*), în concordanță cu (6.1.8), cât şi antrenare pe eşantioane (*on-line*), în concordanță cu (6.1.9), cu un număr de eşantioane N_E specificat de utilizator în caseta de dialog. Utilizatorul poate seta valoarea de prag pentru eroarea pătratică medie utilizată în antrenare şi modul în care se formează vectorii regresorilor.

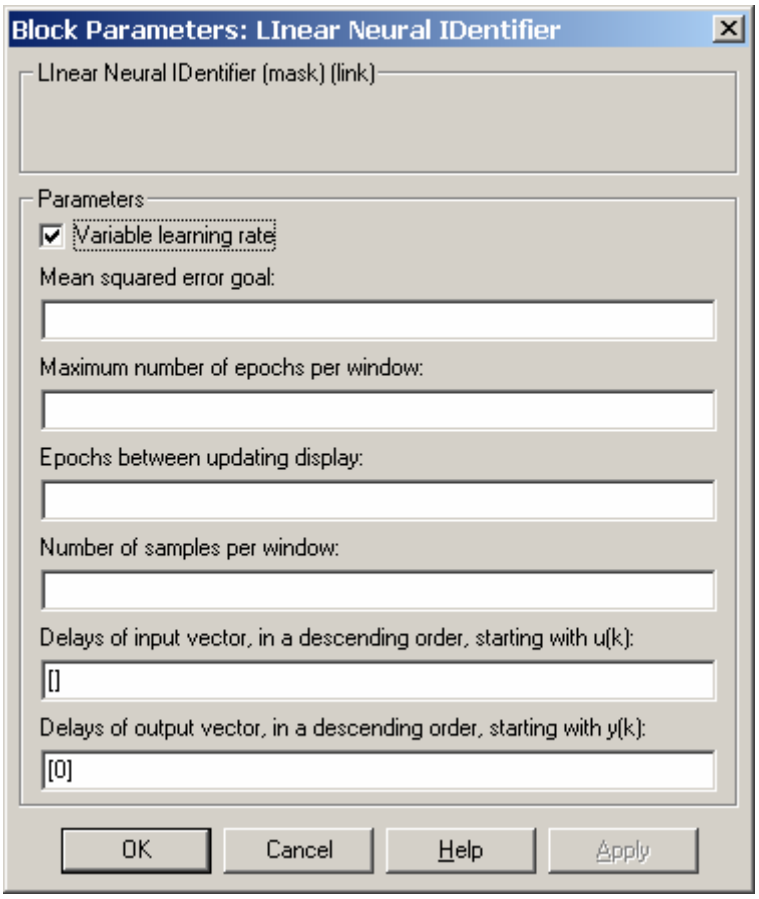


Figura 6.3.2. *Caseta de dialog a blocului* ***LIN_N_ID***.

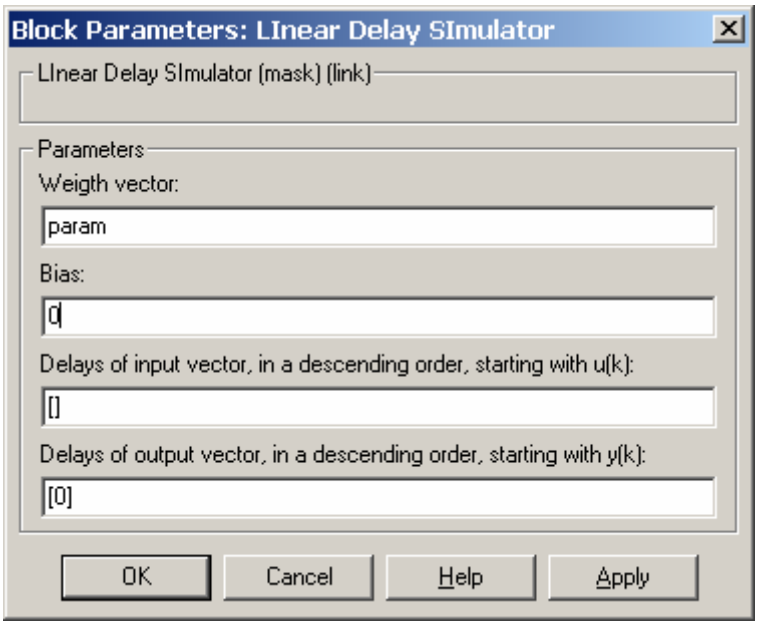


Figura 6.3.3. *Caseta de dialog a blocului* ***LIN_D_SI***.

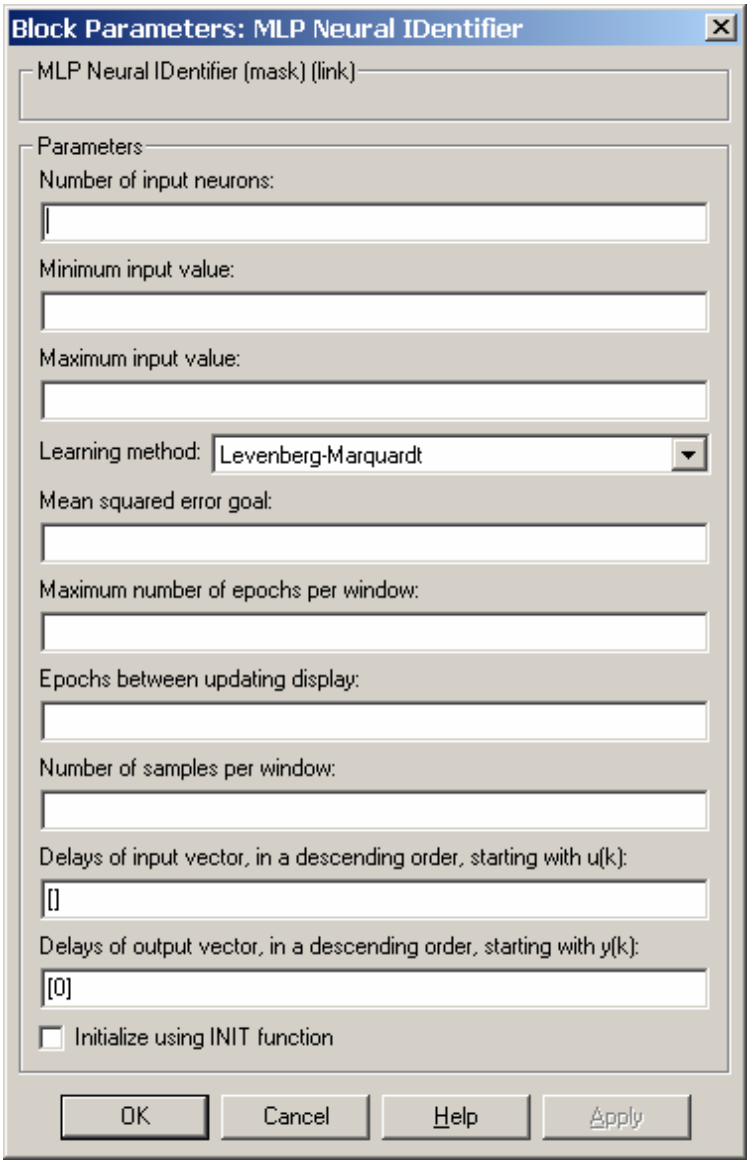


Figura 6.3.4. *Caseta de dialog a blocului* ***MLP_N_ID***.

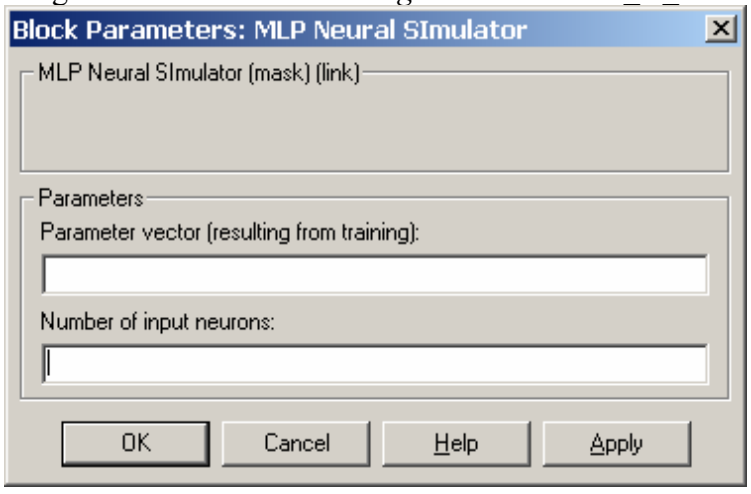


Figura 6.3.5. *Caseta de dialog a blocului* ***MLP_N_SI***.

Casetele de dialog ale blocurilor ***LIN_D_SI*** şi ***MLP_N_SI***, utilizate în simulare, sunt prezentate în figurile 6.3.3 şi, respectiv, 6.3.5. Parametrii reţelelor pot fi setaţi manual sau pot fi încărcaţi automat din spaţiul de lucru al Matlab-ului.

În găsirea regresorilor optimi pentru vectorii de intrare şi ieşire este recomandată o strategie *bottom-up*, în sensul că arhitectura reţelei va fi iniţializată cu un număr redus de regresori. Pentru blocul de tip MLP se va porni cu un număr redus de neuroni pe stratul de intrare. Aceste valori vor incrementate succesiv până la obţinerea rezultatului dorit.

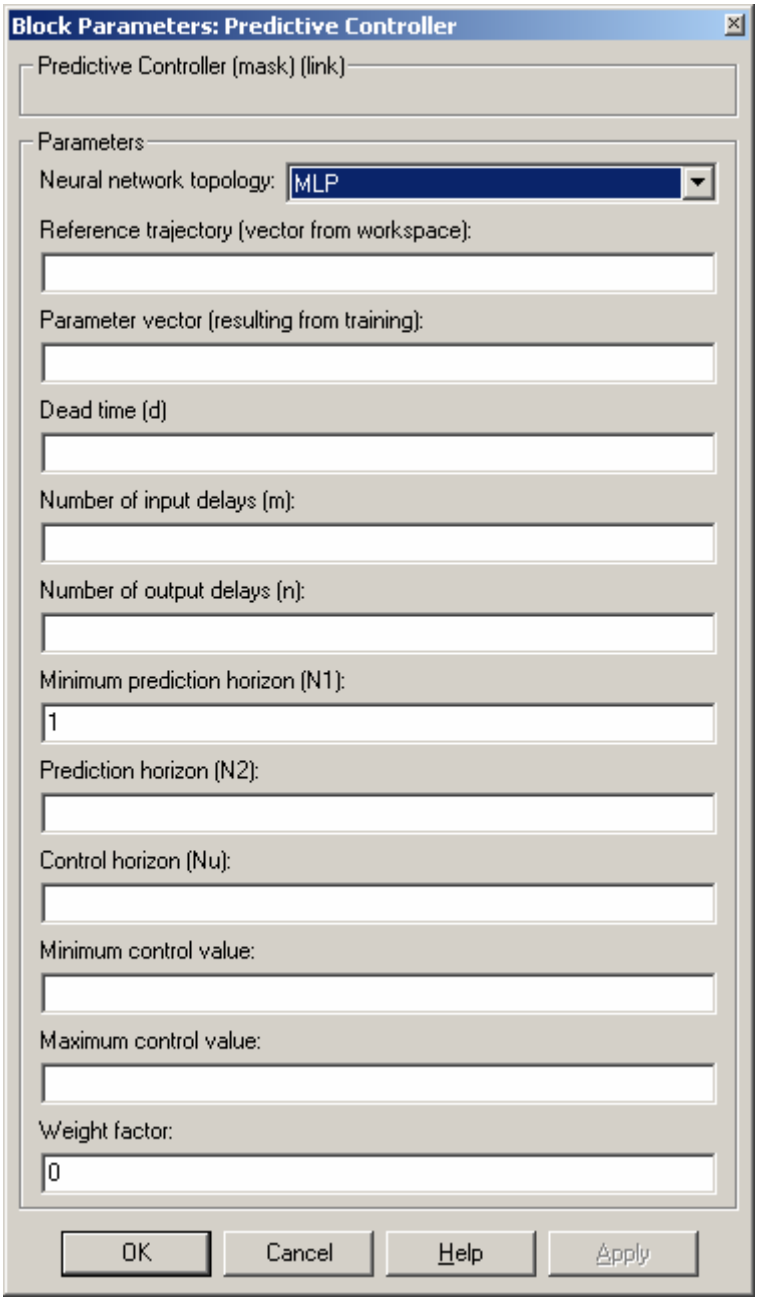


Figura 6.3.6. *Caseta de dialog a blocului* ***PRED_CTRL***.

Blocul ***PRED_CTRL*** poate fi folosit pentru controlul proceselor neliniare ce au drept model neuronal o rețea de tip MLP. Caseta de dialog a acestui bloc este prezentată în figura 6.3.6, parametrii ce pot fi aleşi de utilizator privind atât rețeaua neurală (presupusă deja antrenată), cât şi algoritmul de control predictiv. Vectorul ce conține parametrii modelului neuronal are aceeaşi organizare ca şi vectorul obținut în urma identificării folosind blocurile ***MLP_N_ID***. Valorile întârzierilor intrării şi ieşirii, precum şi valoarea timpului mort, trebuie să fie aceleaşi cu cele considerate la crearea şi antrenarea modelului NNARX.

Parametrilor corespunzători algoritmului de control predictiv ce pot fi setați de către utilizator sunt în concordanță cu notațiile utilizate în paragraful 6.2. Traiectoria referinței este presupusă a fi cunoscută apriori, fiind specificată sub forma unui vector.

Ieşirea 'u' a blocului de control furnizează valoarea intrării de control la fiecare pas de simulare, iar ieşirea 'ref' poate fi utilă pentru a aprecia modul în care ieşirea procesului urmăreşte referința dorită pe durata simulării.

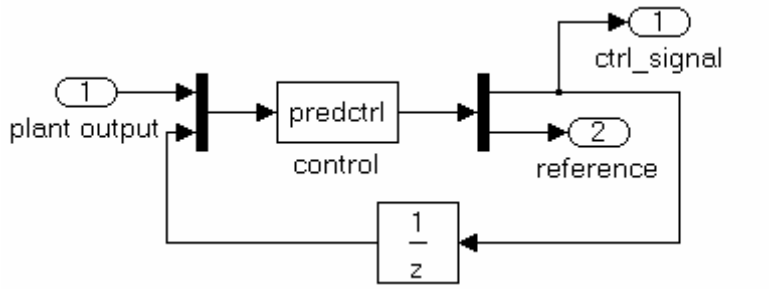


Figura 6.3.7. *Structura intrernă a blocului* ***PRED_CTRL***.

În structura internă a blocului ***PRED_CTRL*** (figura 6.3.7) intră *funcția-S* denumită ***predctrl*** care este responsabilă atât de organizarea datelor interne pe baza parametrilor specifici modelului neuronal şi algoritmului de control, cât şi de minimizarea funcției de cost J (6.2.1). La fiecare pas de simulare (iterație a algoritmului de control predictiv) această funcție apelează o rutină ***cost.m***, care realizează calculul predictorilor neuronali şi al funcției de cost. Pentru crearea modelului neuronal s-au folosit funcțiile specifice din *Neural Network Toolbox*, iar pentru minimizarea funcției de cost s-a făcut apel la funcția *fmincon* din *Optimal Toolbox*, care permite impunerea unor limite (restricții) pentru variabila în raport cu care se efectuează minimizarea. La fiecare iterație a algoritmului este memorat vectorul obținut prin minimizarea lui J, iar acest vector va constitui punctul de start al minimizării la iterația următoare.

Conectarea blocului într-o schemă Simulink în vederea simulării controlului unui oarecare proces dinamic neliniar (pentru care se dispune de un model neuronal) este ilustrată în figura 6.3.8. În timpul simulării, poate fi urmărită evoluția intrării de control şi urmărirea referinței de către ieşirea procesului.

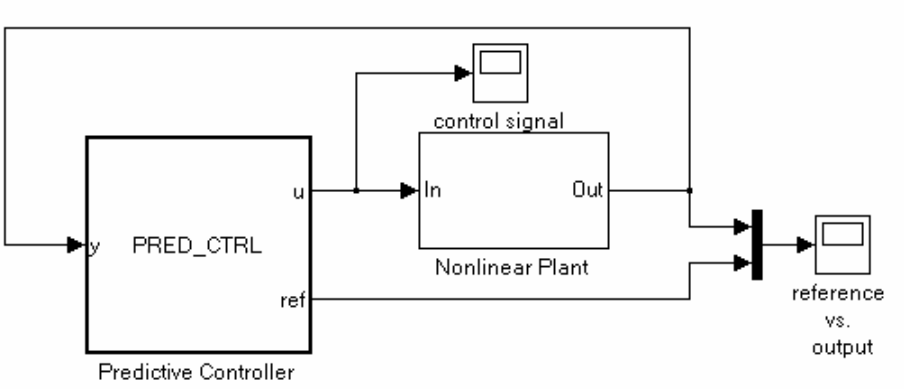


Figura 6.3.8. *Modul de conectare a blocului* ***PRED_CTRL*** *într-o schemă Simulink.*

6.4. Studii de caz

6.4.1. Exemple ilustrative pentru construcţia modelelor

Deşi utilizarea modelelor neuronale este atractivă îndeosebi în cazul neliniar, luăm în discuţie, pentru început, construcţia unui model liniar, deoarece acesta permite comparaţii relevante (de natură parametrică) cu reprezentările standard de tip funcţie de transfer, discretă în timp. Al doilea exemplu ilustrează construcţia unui model neliniar pentru care discuţia privind validitatea se limitează doar la analiza comparativă a răspunsurilor obţinute de la proces şi respectiv model, în condiţiile aceloraşi semnale de test aplicate la intrare. În nici unul dintre cazuri nu abordăm problematica validării modelelor cu ajutorul testelor statistice.

Pentru ambele exemple, funcţionarea procesului supus identificării este simulată în mediul Simulink, antrenarea reţelelor nefăcând însă uz de cunoaşterea ordinului modelelor de simulare (ipoteză absolut firească în desfăşurarea aplicaţiilor reale). Totodată, pentru fiecare din exemple au fost considerate, separat, două situaţii ce urmează a fi detaliate mai jos:

(a) ieşirea procesului nu este afectată de perturbaţii;

(b) ieşirea procesului este afectată aditiv de perturbaţii de tip Gaussian care pot atinge 1% din amplitudinea maximă a semnalului util (corespunzând, de exemplu, zgomotului de conversie al unui convertor A/D cu rezoluţie scăzută).

A. Construcţia unui model liniar

În figura 6.4.1 se prezintă schema Simulink (aferentă situaţiei (*b*) menţionate la începutul secţiunii curente) utilizată pentru construirea unui model liniar de forma (6.1.1) cu perioada de eşantionare $T = 1$ secundă, cu ajutorul blocului ***LI_N_ID***. Procesul ce face obiectul identificării este descris în Simulink prin funcţia de transfer continuă în timp

$$G(s) = \frac{400e^{-15s}}{300s^2 + 40s + 1}, \tag{6.4.1}$$

procesul fiind excitat cu o sursă de zgomot alb de bandă limitată, adecvat aleasă. Singurele informații presupuse cunoscute apriori sunt comportarea liniară a procesului şi valoarea timpul mort (identificabil, de exemplu, de pe un răspuns la semnal treaptă), adică $d = 15$ în (6.4.1) pentru $T = 1$ secundă.

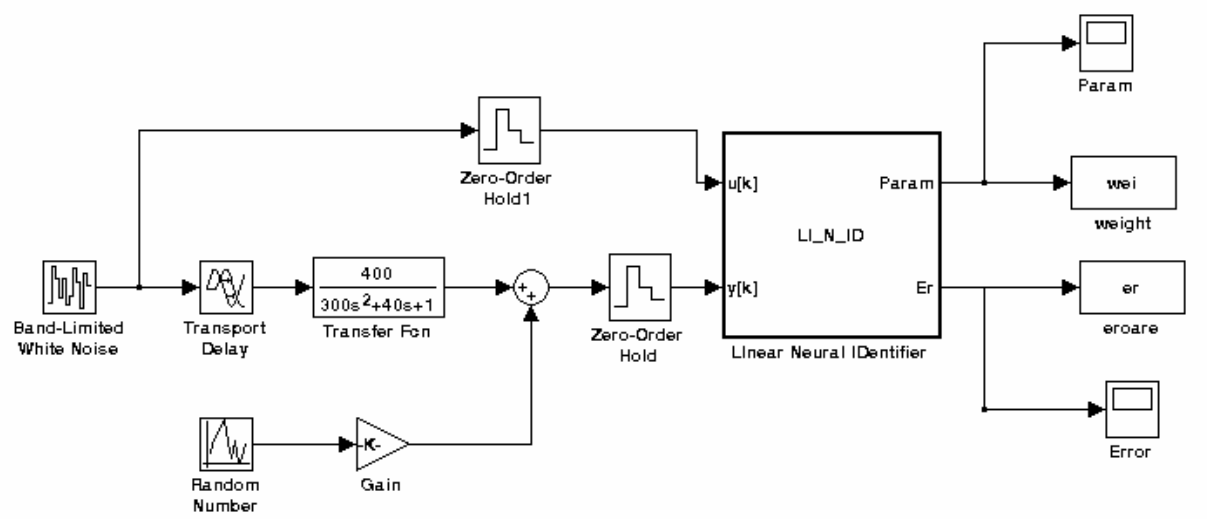


Figura 6.4.1. *Schema Simulink de identificare a modelului liniar corespunzător exemplului din secțiunea A.*

Selectăm drept complet edificatoare doar o parte din rezultate, prezentate în tabelul 6.4.1 pe care le exprimăm (dată fiind liniaritatea modelului) sub formă de funcții de transfer discrete (ale căror coeficienți sunt, până la un semn cunoscut, chiar parametrii furnizați de blocul ***LI_N_ID***). Această exprimare facilitează analiza calității modelului provenit din identificare, prin comparație directă cu discretizată cu T=1 secundă a lui $G(s)$, obținută prin metoda Euler predictor

$$\tilde{G}(z^{-1}) = z^{-15}\,\frac{0.6378z^{-1} + 0.6101z^{-2}}{1 - 1.872z^{-1} + 0.8752z^{-2}}\,. \qquad (6.4.2)$$

Toate cazurile comentate se referă la aceleaşi condiții de antrenare on-line cu N_E=50 eşantioane în funcția obiectiv J din (6.1.9), maxim 100 epoci/fereastră şi rată de învățare variabilă. Rezultatele din tabelul 6.4.1 evidențiază rolul jucat de valoarea obținută în antrenare pentru funcția obiectiv J în aprecierea convergenței parametrilor rețelei şi, totodată, în evaluarea calității modelului identificat.

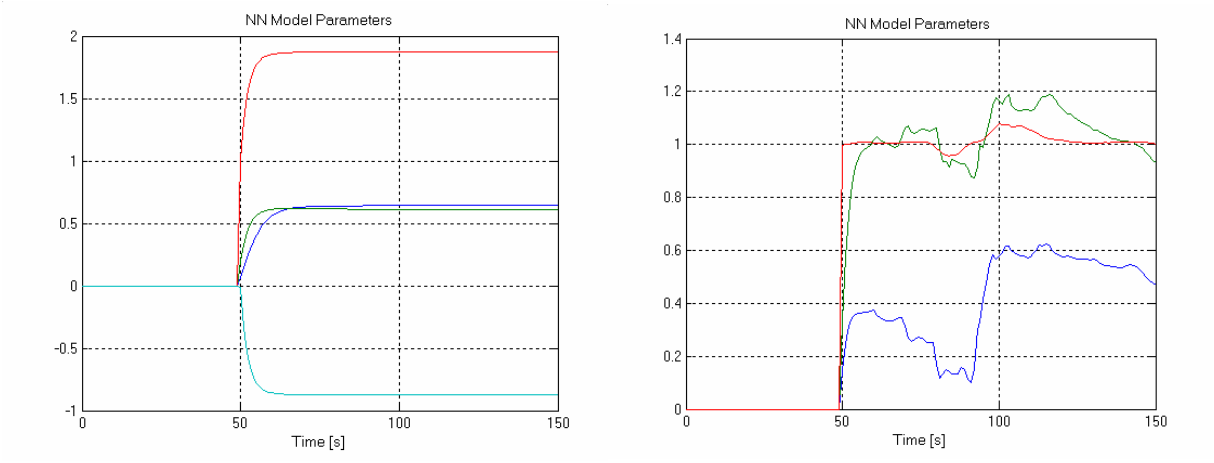
Figurile 6.4.2.a şi 6.4.2.b furnizează reprezentările grafice ale evoluțiilor parametrilor rețelei pentru o durată de 150 de secunde corespunzătoare cazului (*b*) pentru n=2 m=2 şi, respectiv, pentru n=2 şi m=1. Aceste reprezentări grafice ilustrează totodată şi capacitatea blocurilor Simulink utilizate de a furniza on-line informații cantitative ce caracterizează progresul antrenării. Mai subliniem ca elemente importante în construcția modelului neuronal

faptul că în toate situaţiile de convergenţă a parametrilor coeficientul termenului în z^0 al numărătorului (în exprimarea sub formă de funcţie de transfer) rezultă numeric nul (raportat la precizia de calcul).

Tabelul 6.4.1. *Prezentare sintetică a rezultatelor antrenării pentru opt teste de identificare selectate ca relevante pentru secţiunea 6.4.1.*

				Cazul (a) - Ieşire neperturbată
d	*n*	*m*	*J*(150)	Exprimare sub formă de funcţie de transfer
15	2	1	10^{-2}	Parametrii nu converg la valori staţionare
15	2	2	10^{-9}	$z^{-15}\dfrac{0.6377z^{-1}+0.6102z^{-2}}{1-1.872z^{-1}+0.8751z^{-2}}$
15	3	2	10^{-2}	Parametrii nu converg la valori staţionare
15	3	3	10^{-9}	$z^{-15}\dfrac{0.6378z^{-1}+0.9392z^{-2}+0.3148z^{-3}}{1-1.356z^{-1}-0.0907^{-2}+0.4515z^{-3}}=$ $=z^{-15}\dfrac{0.6378(\mathbf{z+0.5160})(z+0.9566)}{(\mathbf{z+0.5159})(z-0.9665)(z-0.9054)}$
				Cazul (b) - Ieşire perturbată
d	*n*	*m*	*J*(150)	Exprimare sub formă de funcţie de transfer
15	2	1	10^{-2}	Parametrii nu converg la valori staţionare
15	2	2	10^{-5}	$z^{-15}\dfrac{0.637z^{-1}+0.6158z^{-2}}{1-1.8711z^{-1}+0.8742z^{-2}}$
15	3	2	10^{-2}	Parametrii nu converg la valori staţionare
15	3	3	10^{-5}	$z^{-15}\dfrac{0.6378z^{-1}+0.9448z^{-2}+0.3144z^{-3}}{1-1.3552z^{-1}-0.0922^{-2}+0.4521z^{-3}}=$ $=z^{-15}\dfrac{0.6378(\mathbf{z+0.5048})(z+0.9766)}{(\mathbf{z+0.5167})(z-0.9043)(z-0.9676)}$

Redundanţa în stabilirea numărului de regresori conduce la apariţia perechilor pol-zerou cu valori numerice identice (raportate la precizia de calcul), fapt ilustrat în tabelul 6.4.1 pentru $n=m=3$ în ambele cazuri (*a*) şi (*b*).



a b

Figura 6.4.2. *Evoluția parametrilor rețelei neuronale pentru exemplul din secțiunea 6.4.1.A corespunzând unui interval de 150 secunde:* a) *Cazul* $n=2$ *şi* $m=2$ *(parametrii converg),* b) *Cazul* $n=2$ *şi* $m=1$ *(parametrii nu converg).*

B. Construcția unui model neliniar

Exemplul prezentat în continuare ilustrează utilizarea blocului ***MLP_N_ID*** în identificarea unui model neliniar de tipul (6.1.1) cu perioada de eşantionare $T=1$ secundă. Procesul ce urmează a fi identificat este simulat în Simulink pe baza următoarei ecuații cu diferențe (recomandată în literatură ca exemplu relevant pentru identificare neurală (Liu, *et al.*, 1998)):

$$y(k)=2.5\frac{y(k-2)y(k-1)}{1+y^2(k-1)+y^2(k-2)}+0.3\cos\big(0.5\big[y(k-1)+y(k-2)\big]\big)+1.2u(k-1)\,. \quad (6.4.3)$$

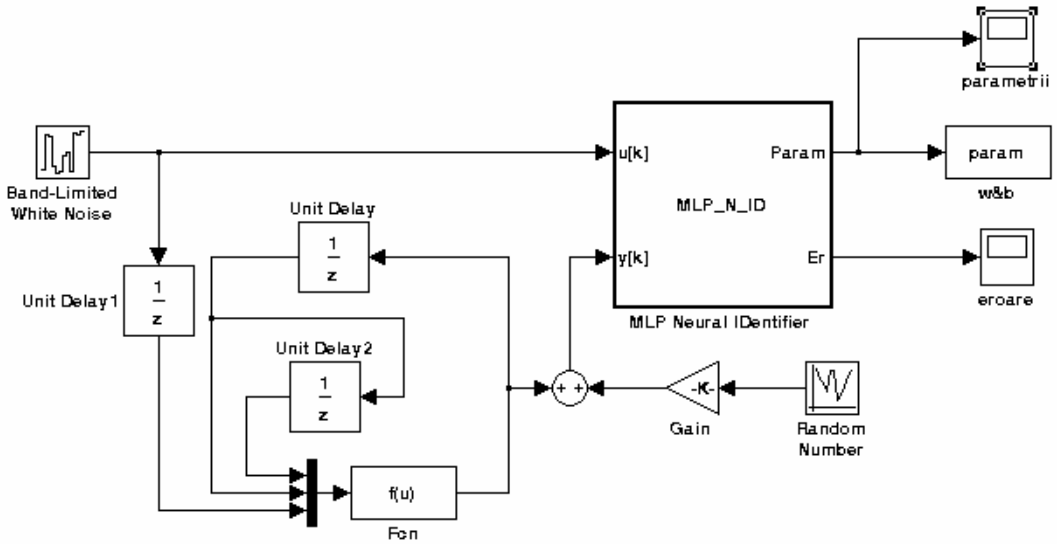
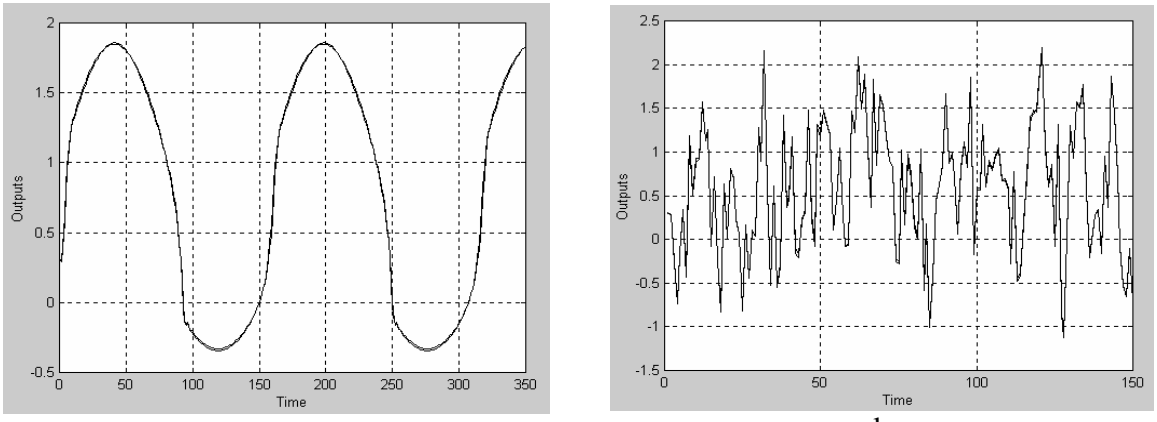


Figura 6.4.3. *Schema Simulink de identificare a modelului neliniar corespunzător exemplului din secțiunea B.*

Figura 6.4.3 prezintă diagrama Simulink utilizată pentru identificare. Semnalul de intrare este de tip „zgomot alb" adecvat ales. Toate cazurile ce vor fi comentate (extrase dintr-un set amplu de teste) se referă la aceleaşi condiţii de antrenare on-line cu $N_E = 200$ eşantioane în funcţia obiectiv J din (6.1.9), utilizând algoritmul Levenberg-Marquardt. Criteriile de oprire a antrenării au fost fie atingerea unui număr maxim de 100 epoci/fereastră sau o acurateţe de aproximare $\varepsilon=10^{-9}$ pentru funcţia obiectiv (6.1.9). Stratul de intrare conţine 10 neuroni sigmoidali. Rezultatele obţinute pentru $d = 0$, $m \geq 1$, $n \geq 2$, în (6.1.1) sunt comparabile, din punct de vedere al acurateţei, pentru cazurile (*a*) şi (*b*) având valori de ordinul 10^{-5} pentru $J(250)$ în cazul (*a*) şi respectiv 10^{-3} în cazul (*b*). Cazurile cu *d*, *m*, *n* ce nu îndeplinesc condiţiile menţionate anterior pot fi uşor separate ele furnizând valori pentru $J(250)$ de ordinul $10^{-1} - 10^{-2}$.

Pentru a ilustra calitatea modelului neural rezultat din identificare, figura 6.4.4 prezintă grafice comparative pentru răspunsul procesului şi a modelului pentru o intrare sinusoidală cu amplitudinea 0.7 (figura 6.4.4.a) şi un semnal de intrare aleator, uniform distribuit în intervalul [-1,1] (figura 6.4.4.b). Parametrii modelului neural sunt cei obţinuţi în situaţia (*b*) cu *d*=0, *m*=1, *n*=2 (adică numărul minim de regresori în (6.1.1) capabili să asigure valori favorabile pentru $J(250)$). Semnalele de test utilizate nu au fost „văzute" în timpul identificării. Folosirea numărului exact de regresori prezenţi în ecuaţia (6.4.3) (*d*=1, *m*=0, *n*=2) nu a adus îmbunătăţiri vizibile în acurateţea de aproximare a modelului obţinut în comparaţie cu o serie de teste nedetaliate în această lucrare.



a b

Figura 6.4.4: *Grafice comparative cu răspunsurile procesului (linie continuă) şi modelului neural MLP (linie întreruptă): a) intrare sinusoidală de amplitudine* 0.7*; b) semnal aleator uniform distribuit în* [-1,1].

În general vorbind, testele efectuate au arătat că modelele neurale nu sunt afectate semnificativ de folosirea regresorilor suplimentari în comparaţie cu folosirea numărului exact de regresori specificat anterior. Totuşi, redundanţa creşte complexitatea modelului neural fără nici un beneficiu real, de aceea e preferabil de căutat un model apropiat de cel minimal.

6.4.2. Exemplu de utilizare a controlului predictiv bazat pe model neural

Modelul neliniar identificat în secțiunea precedentă, corespunzător sistemului descris de ecuația cu diferențe (6.4.2) este utilizat în continuare pentru a proiecta un controller predictiv pentru acest sistem utilizând blocul ***PRED_CTRL***. Schema Simulink utilizată este prezentată în figura 6.4.5. Parametrii utilizați de algoritmul de control predictiv au următoarele valori: orizontul minim de predicție $N_1 = 1$, orizontul de predicție $N_2 = 3$, orizontul de control $N_u = 2$,factorul de ponderare $\lambda = 0$ şi valoarea inițială a intrării $u_{\text{init}} = 0$.

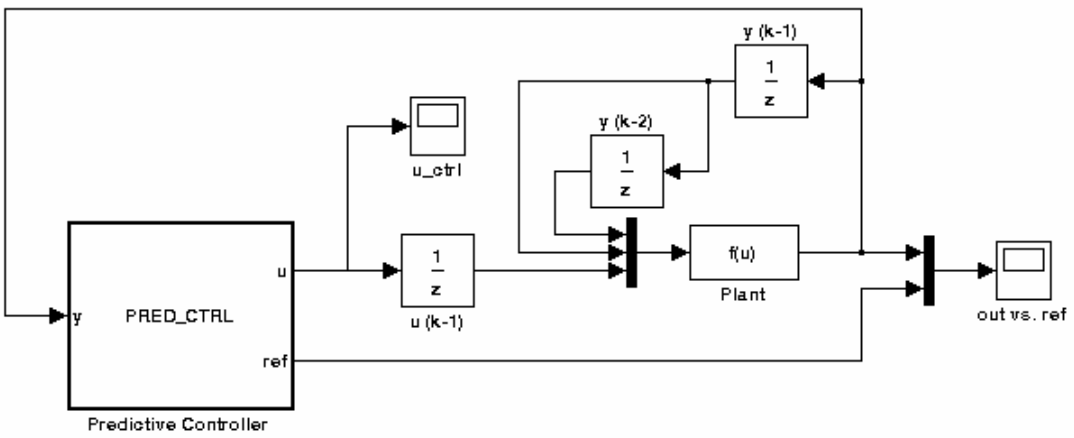


Figura 6.4.5. *Modelul Simulink utilizat pentru controlul predictiv al sistemului neliniar* (6.4.2).

Figura 6.4.6 ilustrează urmărirea unei referințe sinusoidale, cu amplitudine 1, fază 0, offset 0,2 şi frecvență 0,005 Hz, iar figura 6.4.6 urmărirea unei referințe de formă unghiulară, cu valori cuprinse în intervalul [-1,1].

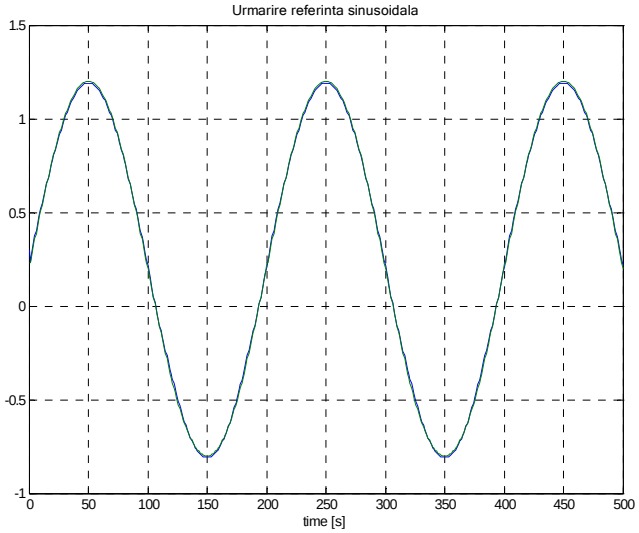


Figura 6.4.6. *Grafice comparative ale unei referințe sinusoidale şi a ieşirii sistemului neliniar controlat de un regulator predictiv*.

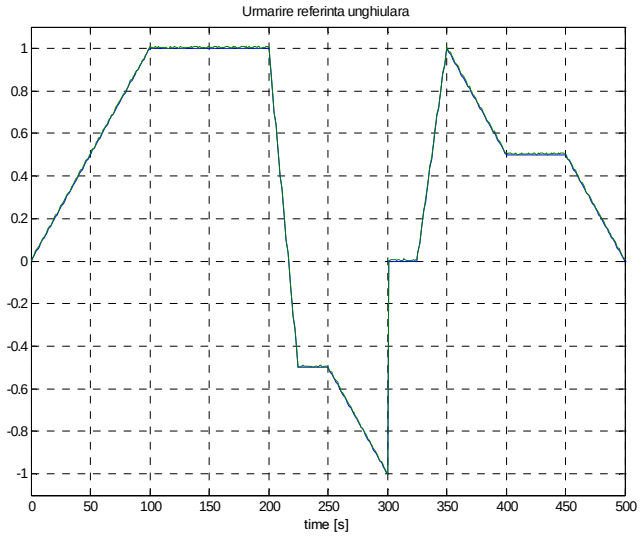


Figura 6.4.7. *Grafice comparative ale unei referințe unghiulare şi a ieşirii sistemului neliniar controlat de un regulator predictiv*.

Pentru aprofundarea noțiunilor prezentate în acest capitol se recomanda studierea problemelor cu soluții analitice 7.17 – 7.20 şi a problemelor cu soluții numerice 8.17, 8.18.

Capitolul 7

PROBLEME CU SOLUȚII ANALITICE

Problema 7.1.

Se consideră un neuron cu o singură intrare, cu funcția de activare liniară. La intrarea neuronului se aplică semnalul $x(t)=\cos\omega t$. Să se determine parametrii neuronului în cazul când la ieşire se obțin semnalele:

a) $y(t)=\cos^2\frac{\omega t}{2}$; b) $y(t)=-\cos^2\frac{\omega t}{2}$;

c) $y(t)=\sin^2\frac{\omega t}{2}$; d) $y(t)=-\sin^2\frac{\omega t}{2}$.

Indicaţii şi răspunsuri:

Cu notațiile din figura 2.1.1, pentru un neuron cu ponderea w, deplasarea b şi funcție de activare liniară, $\sigma(u)=u$, dependența ieşirii y a neuronului de intrarea x a acestuia este descrisă de:

$$y(x)=\sigma(u(x))=wx+b,\ \forall x\in\mathbb{R}.$$

a) Relația precedentă devine:

$$y(t)=wx(t)+b,\forall t\in\mathbb{R}\ \Leftrightarrow\ \cos^2\frac{\omega t}{2}=w\cos\omega t+b,\forall t\in\mathbb{R}.$$

Ținând cont că $\cos\omega t=2\cos^2(\omega t/2)-1$, rezultă:

$$\cos^2\frac{\omega t}{2}=w\left[2\cos^2\frac{\omega t}{2}-1\right]+b,\forall t\in\mathbb{R}\ \Rightarrow\begin{cases}2w=1\\-w+b=0\end{cases}\Rightarrow\begin{cases}w=\frac{1}{2}\\b=\frac{1}{2}\end{cases}$$

b) Similar, se ajunge la:

$$-\cos^2\frac{\omega t}{2}=w\left[2\cos^2\frac{\omega t}{2}-1\right]+b,\forall t\in\mathbb{R}\ \Rightarrow\begin{cases}2w=-1\\-w+b=0\end{cases}\Rightarrow\begin{cases}w=-\dfrac{1}{2}\\b=-\dfrac{1}{2}\end{cases}$$

c) Utilizând relația $y(t)=\sin^2\left(\omega t/2\right)=1-\cos^2\left(\omega t/2\right)$, rezultă:

$$1-\cos^2\frac{\omega t}{2}=w\left[2\cos^2\frac{\omega t}{2}-1\right]+b,\forall t\in\mathbb{R}\ \Rightarrow\begin{cases}2w=-1\\-w+b=1\end{cases}\Rightarrow\begin{cases}w=-\dfrac{1}{2}\\b=\dfrac{1}{2}\end{cases}$$

d) Analog cazului c) se obține:

$$-1+\cos^2\frac{\omega t}{2}=w\left[2\cos^2\frac{\omega t}{2}-1\right]+b,\forall t\in\mathbb{R}\ \Rightarrow\begin{cases}2w=1\\-w+b=-1\end{cases}\Rightarrow\begin{cases}w=\dfrac{1}{2}\\b=-\dfrac{1}{2}\end{cases}$$

Reprezentările grafice cerute în enunțul problemei se obțin imediat.

Problema 7.2.

Se consideră un neuron cu o singură intrare şi funcția de activare treaptă unipolară, la intrarea la intrarea căruia se aplică semnalul $x(t)=\sin 2\pi t,\quad t>0$. Să se exprime analitic şi să se reprezinte grafic semnalul de ieşire $y(t)$ în următoarele cazuri:

(*i*)	$w=1,\ b=0;$	(*ii*)	$w=-1,\ b=0;$
(*iii*)	$w=1,\ b=1;$	(*iv*)	$w=-1,\ b=1;$
(*v*)	$w=1,\ b=-1;$	(*vi*)	$w=-1,\ b=1.$

Indicații şi răspunsuri:

Ieşirea neuronului este dată de $y(t)=\sigma\left(wx(t)+b\right)$, unde σ este funcția de activare de tip treaptă unipolară:

$$\sigma(u)=\begin{cases}0;\,u<0,\\1;\quad u\geq 0.\end{cases}$$

Se obține astfel

$$y(t)=\begin{cases}0,\ \text{dacă}\quad w\sin 2\pi t+b<0,\\1,\ \text{dacă}\quad w\sin 2\pi t+b\geq 0.\end{cases}$$

(*i*) $w=1$ şi $b=0\ \Rightarrow y(t)=\sigma\left(\sin 2\pi t\right)=\begin{cases}1,\quad t\in[0,1/2]+k,\,k\in\mathbb{Z},\\0,\ \text{în rest;}\end{cases}$

(*ii*) $w=-1$ şi $b=0\Rightarrow y(t)=\sigma\left(-\sin 2\pi t\right)=\begin{cases}1,\quad t\in[1/2,1]+k,\,k\in\mathbb{Z},\\0,\ \text{în rest;}\end{cases}$

(*iii*) $w=1$ şi $b=1\Rightarrow y(t)=\sigma[\sin 2\pi t+1]=1,\forall t\in\mathbb{R}$;

(*iv*) $w=-1$ şi $b=1\Rightarrow y(t)=\sigma[-\sin 2\pi t+1]=1,\forall t\in\mathbb{R}$;

$(v)\ \ w=1\ \text{şi}\ \ b=-1 \Rightarrow y(t)=\sigma[\sin 2\pi t-1]=\begin{cases}1,\ t=k+1/4,\ k\in\mathbb{Z},\\ 0,\ \text{în rest;}\end{cases}$;

$(vi)\ \ w=-1\ \text{şi}\ \ b=-1 \Rightarrow y(t)=\sigma[-\sin 2\pi t-1]=\begin{cases}1,\ t=k+3/4,\ k\in\mathbb{Z},\\ 0,\ \text{în rest.}\end{cases}$

Reprezentările grafice cerute în enunțul problemei se obțin imediat.

Problema 7.3.

Se consideră un neuron cu două intrări şi funcția de activare de tip treaptă bipolară. Vectorii de la intrarea neuronului sunt de forma: $\boldsymbol{x}=\begin{bmatrix}x_1 & x_2\end{bmatrix}^T\in\mathbb{R}^2$.

Planul (x_1,x_2) poate fi separat în două semiplane în funcție de valorile ieşirii y a rețelei astfel: într-unul din aceste semiplane $y=-1$, iar în celălalt $y=1$.

Să se determine parametrii neuronului astfel încât să aibă loc următoarele separări:

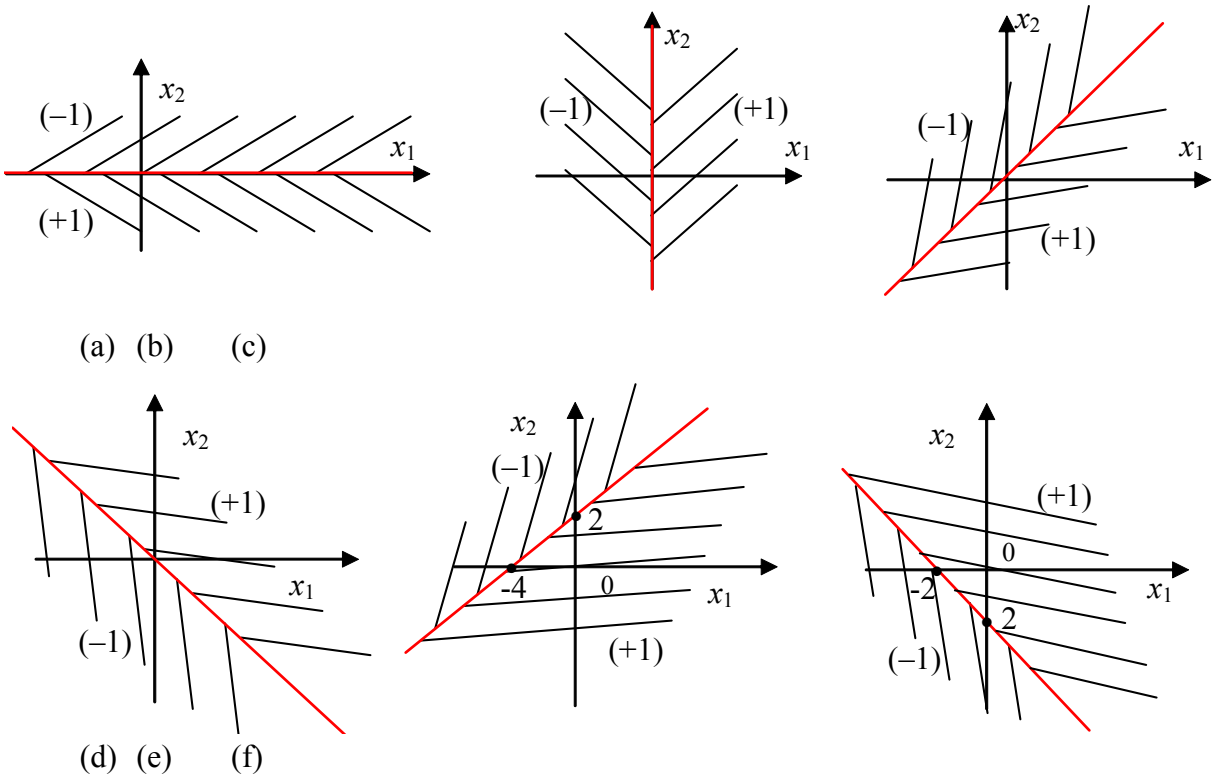


Figura 7.3.1. *Separarea planului* (x_1,x_2) *în regiuni.*

Indicații şi răspunsuri:

Expresia funcției de activare de tip treaptă bipolară este:

$$\sigma(u)=\begin{cases}-1,\ u<0,\\ 1\ ,\ u\geq 0.\end{cases}$$

Cu notațiile din figura 3.1.1, vectorul ponderilor $\boldsymbol{w} = [w_1 \; w_2] \in \mathbb{R}^{1\times 2}$ şi deplasarea $b \in \mathbb{R}$, pentru neuronul cu două intrări se obține ieşirea:

$$y(x_1, x_2) = -1 \;\Leftrightarrow\; w_1 x_1 + w_2 x_2 + b < 0,$$
$$y(x_1, x_2) = +1 \;\Leftrightarrow\; w_1 x_1 + w_2 x_2 + b \geq 0.$$

Ecuația dreptei care separă planul (x_1, x_2) în cele două semiplane în funcție de valorile ieşirii y a rețelei este

$$w_1 x_1 + w_2 x_2 + b = 0\,. \qquad \text{(S)}$$

(a) Ecuația dreptei de separație este $x_2 = 0$, $\forall x_1 \in \mathbb{R}$. Înlocuind în (S) se obține $w_1 x_1 + b = 0$, $\forall x_1 \in \mathbb{R}$, de unde rezultă $w_1 = 0$, $b = 0$, astfel că $u = w_2 x_2$. Din figura (a) se observă că pentru $x_2 > 0$ ieşirea neuronului are valoarea $y = -1$, care corespunde lui $u < 0$, echivalent cu $w_2 x_2 < 0$. Rezultă că separarea planului (x_1, x_2) în modul prezentat în figura (a) poate fi realizată de un neuron pentru care ponderea w_2 are o valoare negativă arbitrară $w_2 < 0$.

(b) Analog rezultă că $w_2 = 0$, $b = 0$. Se observă că $x_1 < 0$ pentru $u < 0$, de unde se obține $w_1 > 0$.

(c) Ecuația dreptei de separație este $x_2 = x_1$, $x_1 \in \mathbb{R}$. Înlocuim pe x_1, x_2 în expresia lui y şi obținem: $w_1 x_1 + w_2 x_2 + b = 0, (\forall)\, x_1 = x_2 = x \in R \Rightarrow (w_1 + w_2)x + b = 0, (\forall)\, x \in R$. De aici rezultă că $w_1 = -w_2\,; b = 0$. Pentru $x_1 = 0$ şi $x_2 > 0$ avem $y = -1$, de unde rezultă că $w_2 x_2 < 0$, deci $w_2 < 0$.

(d) Ecuația dreptei de separație este $x_2 = -x_1$, $x_1 \in \mathbb{R}$. Similar cazului (c) se obține $w_1 = w_2 > 0$, $b = 0$.

(e) $x_2 = \frac{1}{2} x_1 + 2$, $x_1 \in \mathbb{R}$, ceea ce conduce la $w_1 = -\frac{1}{2} w_2$, $b = -2w_2$. Pentru $x_1 = x_2 = 0$ ieşirea neuronului trebuie să fie +1, astfel că $w_2 < 0$.

(f) Ecuația dreptei de separație este $x_2 = -x_1 - 2$, $x_1 \in \mathbb{R}$, de unde rezultă $w_1 = w_2$ şi $b = 2w_2$. Pentru $x_1 = x_2 = 0$ ieşirea neuronului trebuie să fie +1, astfel că $w_2 > 0$.

Problema 7.4.

Se consideră un perceptron cu o singură intrare şi funcția de activare unipolară. La intrarea acestuia se aplică vectorii prototip: $x^1 = 1$, $x^2 = 3$, $x^3 = 0$.

Se definesc vectorii eroare: $e^i = z^i - y^i, i = 1,2,3$, pentru vectorii țintă z^i, $i = 1,2,3$, asociați celor 3 vectori prototip.

a) Să se discute în planul parametrilor rețelei (b – abscisă, w – ordonată) valorile luate de erorile e^i, $i=1,2,3$ şi să se precizeze regiunea pentru care se realizează $e^1=0$, $e^2=0$, $e^3=0$.

Se vor avea în vedere următoarele două seturi de valori ale vectorilor țintă:

(i) $z^1=0,\ z^2=1,\ z^3=0$;

(ii) $z^1=0,\ z^2=1,\ z^3=1$.

b) Să se explice diferențele dintre rezultatele obținute în cazurile a) şi b) de la punctul I.

Indicaţii şi răspunsuri:

I) Erorile sunt date de $e^i=z^i-\sigma(wx^i+b)$, $i=\overline{1,3}$, cu $\sigma(u)=\begin{cases}0, & u<0,\\ 1, & u\ge 0.\end{cases}$

a) Pentru vectori țintă: $z^1=0,\ z^2=1,\ z^3=0$, se obține

$$e^1=\begin{cases}0, & w+b<0,\\ -1, & w+b\ge 0;\end{cases}\quad e^2=\begin{cases}0, & 3w+b\ge 0,\\ 1, & 3w+b<0;\end{cases}\quad e^3=\begin{cases}0, & b<0,\ w\in\mathbb{R},\\ -1, & b\ge 0,\ w\in\mathbb{R}.\end{cases}$$

Regiunea pentru care se realizează $e^1=0$, $e^2=0$, $e^3=0$ este prezentată în figura 7.4.1.

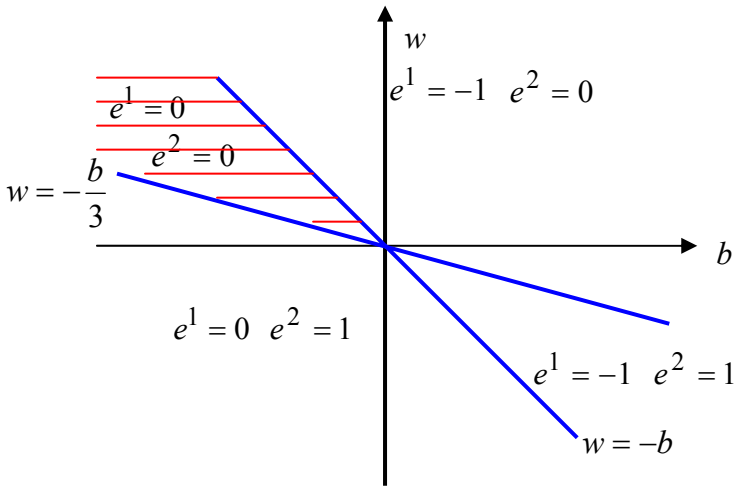


Figura 7.4.1. *Reprezentarea grafică a regiunii din planul parametrilor pentru care* $e^1=0$, $e^2=0$, $e^3=0$ *pentru vectorii țintă din cazul (a).*

b) Pentru vectori țintă: $z^1=0,\ z^2=1,\ z^3=1$, rezultă

$$e^1=\begin{cases}0, & w+b<0,\\ -1, & w+b\ge 0;\end{cases}\quad e^2=\begin{cases}0, & 3w+b\ge 0,\\ 1, & 3w+b<0;\end{cases}\quad e^3=\begin{cases}0, & b\ge 0,\ w\in\mathbb{R},\\ 1, & b<0,\ w\in\mathbb{R}.\end{cases}$$

II. Diferențele dintre cele două cazuri sunt datorate modificării vectorului țintă z^3.

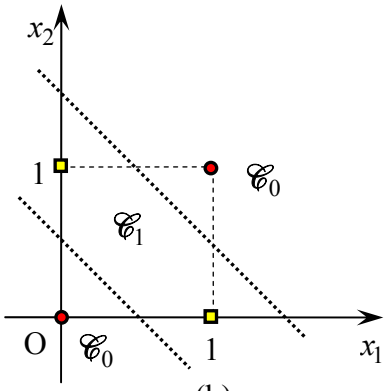
Problema 7.5.

Se consideră funcția logică XOR având tabela de adevăr şi reprezentarea grafică din figura 7.5.1. Funcția stabileşte apartenența vectorilor $\boldsymbol{x} = [x_1 \;\; x_2]^T$, $x_1, x_2 \in \{0, 1\}$, la două clase definite prin $\mathscr{C}_0 = \{(x_1, x_2) \in P \mid x_1 \oplus x_2 = 0\}$ şi $\mathscr{C}_1 = \{(x_1, x_2) \in P \mid x_1 \oplus x_2 = 1\}$.

Să se arate că cele două clase pot fi separate cu ajutorul unei rețele perceptron cu 2 straturi, având 2 neuroni în stratul de intrare.

x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

(a)



(b)

Figura 7.5.1. (a) *Tabela de adevăr şi* (b) *reprezentarea grafică a funcției logice XOR.*

Indicații şi răspunsuri:

Se utilizează exprimarea funcției XOR sub forma echivalentă $x_1 \oplus x_2 = (x_1 \wedge \overline{x_2}) \vee (\overline{x_1} \wedge x_2)$.

Pe stratul de intrare plasăm 2 neuroni care implementează funcțiile logice $x_1 \wedge \overline{x_2}$ respectiv $\overline{x_1} \wedge x_2$. Stratul de ieşire va conține un singur neuron cu 2 intrări ce implementează funcția OR. Ținând cont de chestiunile prezentate în exemplul 3.1.1 se obțin următoarele valori:

– pentru neuronul 1 de pe stratul de intrare: $w_{1,1}^1 = 1$, $w_{1,2}^1 = -1$, $b_1^1 = -0.5$;

– pentru neuronul 2 de pe stratul de intrare: $w_{2,1}^1 = -1$, $w_{2,2}^1 = 1$, $b_2^1 = -0.5$;

– pentru neuronul de pe stratul de ieşire: $w_1^2 = w_2^2 = 1$, $b^2 = -0.5$.

Problema 7.6.

În planul $\mathbb{R}^2$ se consideră următoarele două perechi de puncte (x_i, z_i), $i = 1,2$: (0,0), (1,2).

a) Să se determine ecuația unei drepte de forma $y = wx + b$ care trece prin punctele (x_i, z_i), $i = \overline{1,2}$.

b) Se consideră un neuron liniar cu o singură intrare unde se prezintă valorile x_1, x_2. Neuronul este conectat cu un comparator (ca în figură) care furnizează valorile: $e_1 = z_1 - f(x_1)$, $e_2 = z_2 - f(x_2)$, atunci când pe intrarea cu (+) a comparatorului se prezintă z_1, respectiv z_2.

Se consideră eroarea pătratică globală $J = e_1^2 + e_2^2$, ca funcţie de care depinde de parametrii neuronului w şi b.

(i) Să se reprezinte grafic suprafaţa $J(w, b)$.

(ii) Să se arate că această suprafaţă posedă un singur punct de minim global şi să se precizeze valorile w^*, b^* corespunzătoare acestui punct.

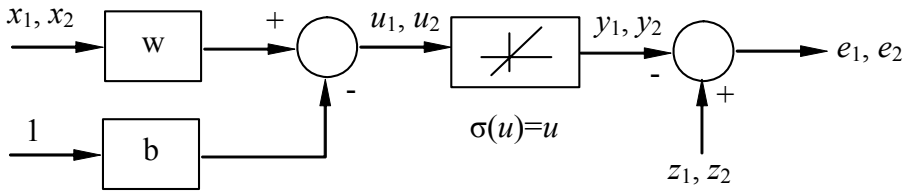


Figura 7.6.1. *Schema bloc pentru generarea semnalelor de eroare din problema 7.5.*

Indicaţii şi răspunsuri:

a) Ecuaţia dreptei de forma $y = wx + b$ ce trece prin punctele date se determină din relaţiile:

$$\begin{cases} 0 = 0 * w + b \\ 2 = w + b \end{cases} \Rightarrow \begin{cases} w = 2 \\ b = 0 \end{cases}$$

b) Pentru neuronul liniar cu o intrare, $y(x_i) = wx_i + b$, $e_i = z_i - (x_i w - b)$, $i = 1, 2$. Eroarea pătratică globală este $J(w,b) = e_1^2 + e_2^2 = b^2 + (2 - w + b)^2 = 2b^2 + 4 + w^2 - 4w + 4b - 2wb$.

(ii) Punctele de extrem ale funcţiei J se determină egalând cu 0 derivatele parţiale ale acesteia în raport cu cei doi parametri, $\frac{\partial J}{\partial w} = 2w - 4 - 2b$ şi $\frac{\partial J}{\partial b} = 4b + 4 - 2w$, ceea ce conduce la sistemul

$$\begin{cases} 2w - 4 - 2b = 0 \\ 4b + 4 - 2w = 0 \end{cases} \Leftrightarrow \begin{cases} 2w - 2b = 4 \\ -2w + 4b = -4 \end{cases} \Rightarrow \begin{cases} b^* = 0 \\ w^* = 2 \end{cases}$$

Matricea Hessian a funcţiei J în punctul $b^* = 0$, $w^* = 2$, este

$$H = \begin{bmatrix} \frac{\partial^2 J}{\partial w^2} & \frac{\partial^2 J}{\partial w \partial b} \\ \frac{\partial^2 J}{\partial b \partial w} & \frac{\partial^2 J}{\partial b^2} \end{bmatrix} = \begin{bmatrix} 2 & -2 \\ -2 & 4 \end{bmatrix}.$$

Deoarece valorile proprii ale lui H sunt $\lambda_{1,2} = 3 \pm \sqrt{5} > 0$, matricea H este pozitiv definită. Rezultă că punctul $b^* = 0$, $w^* = 2$ este punct de minim global al funcţiei J.

Problema 7.7.

Se consideră amplificatorul operaţional din figură, unde $x_1, x_2, \ldots, x_n$ sunt semnale de intrare, iar y este semnal de ieşire.

Să se modeleze funcționarea amplificatorului utilizând o rețea neuronală cu funcții de activare liniare (se va dimensiona rețeaua şi se vor preciza valorile parametrilor).

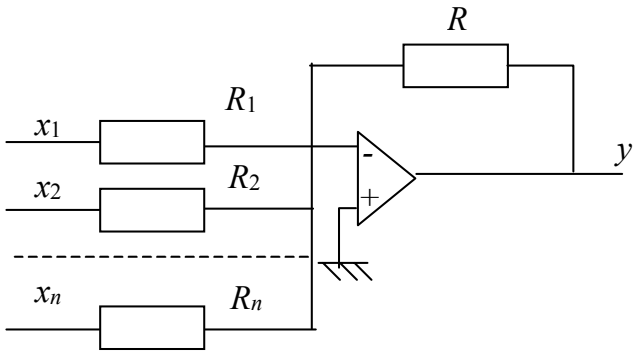


Figura 7.7.1. *Schema bloc a unui amplificator operațional cu n intrări.*

Indicații şi răspunsuri:

Funcționarea amplificatorului operațional este descrisă de ecuația:

$$\sum_{i=1}^{n} \frac{x_i}{R_i} = -\frac{y}{R},$$

unde x_i, $i = 1,n$, şi y reprezintă tensiunea de la intrarea i, $(i = 1,n)$ şi, respectiv, ieşirea amplificatorului operațional.

Legătura dintre ieşirea y a unui neuron liniar şi cele n intrări ale sale este:

$$y = \sum_{i=1}^{n} x_i w_i + b\,.$$

O rețea neuronală formată dintr-un neuron liniar cu n intrări cu parametrii $w_i = -\frac{R}{R_i}$ şi $b = 0$ va modela funcționarea amplificatorului operațional.

Problema 7.8.

Se consideră un neuron cu o singură intrare şi funcția de activare liniară. Pentru ce valori ale parametrilor neuronului ieşirea acestuia aproximează, în sensul celor mai mici pătrate, funcția φ cunoscută în următoarele 3 puncte: $\varphi(-1) = -1$, $\varphi(0) = 0$, $\varphi(1) = 2$

Indicații şi răspunsuri:

Notăm $x_1 = -1$, $x_2 = 0$, $x_3 = 1$, şi $z_i = \varphi(x_i)$, $i = 1,2,3$.

Pentru neuronul liniar cu parametrii w şi b, notăm $y_i = \sigma(wx_i + b)$, $e_i = z_i - y_i$, $i = 1,2,3$.

Parametrii neuronului vor fi determinați astfel încât sa fie minimizată eroarea pătratică:

$$J(w,b) = \sum_{i=1}^{3} e_i^{\,2} = (-1 + w - b)^2 + (-b)^2 + (2 - w - b)^2 = 2w^2 + 3b^2 - 6w - 2b + 5\,.$$

Punctul de minim al funcției se obține ca soluție a sistemului de ecuații:

$$\begin{cases} \dfrac{\partial J}{\partial w} = 0 \\ \dfrac{\partial J}{\partial b} = 0 \end{cases} \Rightarrow \begin{cases} 4w - 6 = 0 \\ 6b - 2 = 0 \end{cases} \Rightarrow \begin{cases} w = 3/2 \\ b = 1/3 \end{cases}$$

Se calculează matricea hessian şi se observă că este pozitiv definită, deci J are un unic punct de minim (global).

Problema 7.9.

Se consideră un neuron cu o singură intrare şi funcția de activare σ derivabilă. La intrarea neuronului se aplică $x \in \mathbb{R}$, iar $y \in \mathbb{R}$ este comparat cu $z \in \mathbb{R}$ dat (fixat), furnizând eroarea $e = z - y$ conform figurii. Se consideră funcția: $J = \dfrac{1}{2}e^2$.

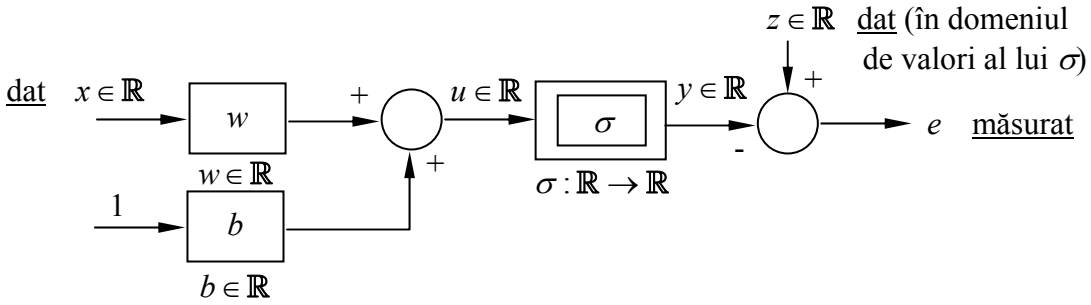


Figura 7.9.1. *Schema bloc pentru generarea semnalelor de eroare din problema 7.9.*

a) Să se exprime $\dfrac{\partial J}{\partial w}$ şi $\dfrac{\partial J}{\partial b}$ în funcție de e, x şi $\sigma'(u)$.

b) Să se exprime $\sigma'(u)$ ca o funcție de argument y.

c) Folosind rezultatele obținute la subpunctele a) şi b) să se exprime $\dfrac{\partial J}{\partial w}$ şi $\dfrac{\partial J}{\partial b}$ ca depinzând de e, x şi valoarea unei funcții de argument y.

Indicații şi răspunsuri:

a) $J(w,b) = \frac{1}{2}e^2 = \frac{1}{2}\left[z - \sigma(wx+b)\right]^2 \Rightarrow \begin{array}{l} \dfrac{\partial J}{\partial w} = \dfrac{\partial J}{\partial e}\dfrac{\partial e}{\partial \sigma}\dfrac{\partial \sigma}{\partial u}\dfrac{\partial u}{\partial w} = -e\sigma'(u)x \\ \dfrac{\partial J}{\partial b} = \dfrac{\partial J}{\partial e}\dfrac{\partial e}{\partial \sigma}\dfrac{\partial \sigma}{\partial u}\dfrac{\partial u}{\partial b} = -e\sigma'(u) \end{array}$

b) $\left.\begin{array}{l} \dfrac{\partial \sigma}{\partial y} = \dfrac{\partial \sigma}{\partial u}\cdot\dfrac{\partial u}{\partial y} \\ \dfrac{\partial \sigma}{\partial y} = 1 \end{array}\right\} \Rightarrow \dfrac{\partial \sigma}{\partial u} = \dfrac{1}{\dfrac{\partial u}{\partial y}} \Leftrightarrow \sigma'(u) = \dfrac{1}{(Inv\sigma)'(y)}$

c) Notând $f(y)=\dfrac{1}{(Inv\sigma)'(y)}$, rezultă $\dfrac{\partial J}{\partial w}=-ef(y)x$, $\dfrac{\partial J}{\partial b}=-ef(y)$

Problema 7.10.

Se consideră funcția $\sigma:\mathbb{R}\to\mathbb{R}$, $y=\sigma(u)$. Să se exprime $\sigma'(u)$ ca o funcție de y pentru următoarele situații:

a) $\sigma(u)=\text{logsig}(u)$,

b) $\sigma(u)=\text{tansig}(u)$.

Indicații și răspunsuri:

a) $\sigma(u)=\dfrac{1}{1+e^{-u}}=y \;\Rightarrow\; \sigma'(u)=\dfrac{1}{(Inv\sigma)'(y)}$, cu $Inv\sigma(y)=\ln\left(\dfrac{y}{1-y}\right) \Rightarrow \sigma'(u)=y(1-y)$

b) $\sigma(u)=\dfrac{e^{u}-e^{-u}}{e^{u}+e^{-u}}=y \;\Rightarrow\; \sigma'(u)=\dfrac{1}{(Inv\sigma)'(y)}$, cu $Inv\sigma(y)=\frac{1}{2}\ln\left(\dfrac{1+y}{1-y}\right) \Rightarrow \sigma'(u)=1-y^2$

Problema 7.11.

a) Se consideră $F:\mathbb{R}^n\to\mathbb{R}$, $F(\boldsymbol{x})=\dfrac{1}{2}\boldsymbol{x}^T\boldsymbol{x}$. Să se demonstreze că: $\dfrac{\partial F}{\partial \boldsymbol{x}}=\boldsymbol{x}^T$.

b) Se consideră funcția $F:\mathbb{R}^n\to\mathbb{R}^n$, $F(\boldsymbol{x})=\boldsymbol{a}-\boldsymbol{x}$, cu $\boldsymbol{a}\in\mathbb{R}^n$ un vector constant. Să se demonstreze că: $\dfrac{\partial F}{\partial \boldsymbol{x}}=-\boldsymbol{I}_n$, unde $\boldsymbol{I}_n$ notează matricea unitate de ordinul n.

c) Se consideră funcția $f:\mathbb{R}^n\to\mathbb{R}^n$, $f(\boldsymbol{x})=\boldsymbol{A}\boldsymbol{x}$, cu $\boldsymbol{A}=\begin{bmatrix}\boldsymbol{a}_1\\ \vdots\\ \boldsymbol{a}_n\end{bmatrix}$, $\boldsymbol{a}_k\in\mathbb{R}^{1\times n}$, $k=\overline{1,n}$. Să se calculeze matricele Jacobian $\left[\dfrac{\partial f}{\partial \boldsymbol{a}_k}\right]$, $k=\overline{1,n}$.

Indicații și răspunsuri:

a) $F(\boldsymbol{x})=\dfrac{1}{2}\sum_{i=1}^{n}x_i^{\,2}$ cu $\boldsymbol{x}=[x_1\ x_2\ \ldots\ x_n]^T$.

$\dfrac{\partial F}{\partial x_i}=x_i,\ i=\overline{1,n},\ \Rightarrow\ \dfrac{\partial F}{\partial \boldsymbol{x}}=\left[\dfrac{\partial F}{\partial x_1}\ \dfrac{\partial F}{\partial x_2}\cdots\dfrac{\partial F}{\partial x_n}\right]=[x_1\ x_2\cdots x_n]=\boldsymbol{x}^T$

b) $F(\boldsymbol{x})=\begin{bmatrix}a_1\\a_2\\\vdots\\a_n\end{bmatrix}-\begin{bmatrix}x_1\\x_2\\\vdots\\x_n\end{bmatrix}=\begin{bmatrix}a_1-x_1\\a_2-x_2\\\vdots\\a_n-x_n\end{bmatrix}=\begin{bmatrix}f_1(\boldsymbol{x})\\f_2(\boldsymbol{x})\\\vdots\\f_n(\boldsymbol{x})\end{bmatrix}$; cu $f_i(\boldsymbol{x})=a_i-x_i,\ i=\overline{1,n}$.

$$\frac{\partial F}{\partial x}=\begin{bmatrix}\frac{\partial f_1}{\partial x_1}&\frac{\partial f_1}{\partial x_2}&\cdots&\frac{\partial f_1}{\partial x_n}\\\frac{\partial f_2}{\partial x_1}&\frac{\partial f_2}{\partial x_2}&\cdots&\frac{\partial f_2}{\partial x_n}\\\vdots&\vdots&&\vdots\\\frac{\partial f_n}{\partial x_1}&\frac{\partial f_n}{\partial x_2}&\cdots&\frac{\partial f_n}{\partial x_n}\end{bmatrix}=\begin{bmatrix}-1&0&\cdots&0\\0&-1&\cdots&0\\\vdots&\vdots&&\vdots\\0&0&\cdots&-1\end{bmatrix}=-\boldsymbol{I}_n$$

c) $f(\boldsymbol{x})=\boldsymbol{A}\boldsymbol{x},\ f(\boldsymbol{x})=\begin{bmatrix}\boldsymbol{a}_1\\\boldsymbol{a}_2\\\vdots\\\boldsymbol{a}_n\end{bmatrix}\boldsymbol{x},\quad \boldsymbol{a}_k\in R^{1xn},\quad k=\overline{1,n}\quad\Rightarrow f(\boldsymbol{x})=\begin{bmatrix}\boldsymbol{a}_1\boldsymbol{x}\\\boldsymbol{a}_2\boldsymbol{x}\\\vdots\\\boldsymbol{a}_n\boldsymbol{x}\end{bmatrix}=\begin{bmatrix}f_1(\boldsymbol{x})\\f_2(\boldsymbol{x})\\\vdots\\f_n(\boldsymbol{x})\end{bmatrix}$

Cu notațiile $\boldsymbol{a}_k=[a_k^1\quad a_k^2\ \ldots\ a_k^n],\ k=\overline{1,n},\ \boldsymbol{x}=[x_1\ x_2\ \cdots\ x_n]^T$; se obține:

$$\frac{\partial f}{\partial \boldsymbol{a}_k}=\begin{bmatrix}\frac{\partial f_1}{\partial a_k^1}&\frac{\partial f_1}{\partial a_k^2}&\cdots&\frac{\partial f_1}{\partial a_k^n}\\\frac{\partial f_2}{\partial a_k^1}&\frac{\partial f_2}{\partial a_k^2}&\cdots&\frac{\partial f_2}{\partial a_k^n}\\\\\frac{\partial f_k}{\partial a_k^1}&\frac{\partial f_k}{\partial a_k^2}&\cdots&\frac{\partial f_k}{\partial a_k^n}\\\\\frac{\partial f_n}{\partial a_k^1}&\frac{\partial f_n}{\partial a_k^2}&\cdots&\frac{\partial f_n}{\partial a_k^n}\end{bmatrix}=\begin{bmatrix}0&0&\cdots&0\\0&0&\cdots&0\\\vdots&\vdots&&\vdots\\x_1&x_2&\cdots&x_n\\\vdots&\vdots&&\vdots\\0&0&\cdots&0\end{bmatrix}=\begin{bmatrix}\boldsymbol{O}_{(k-1)xn}\\\boldsymbol{x}^T\\\boldsymbol{O}_{(n-k)xn}\end{bmatrix},\ k=\overline{1,n}.$$

Problema 7.12.

Se consideră un neuron liniar cu o singură intrare şi un neuron tansigmoidal cu o singură intrare, ambii având deplasări nule. Intrarea neuronilor este comună, notată *x*, iar ieşirile lor sunt conectate la un comparator, furnizând *e*(*x*), ca în figură:

Să se discute soluțiile ecuației $e(x)=0$ pentru $w_1\in\mathbb{R}$, $w_2\in\mathbb{R}$ (soluțiile nu se vor calcula efectiv!). În planul parametrilor (w_1, w_2) – cu w_1 abscisă şi w_2 ordonată – se vor delimita regiunile care conțin numere diferite de soluții distincte.

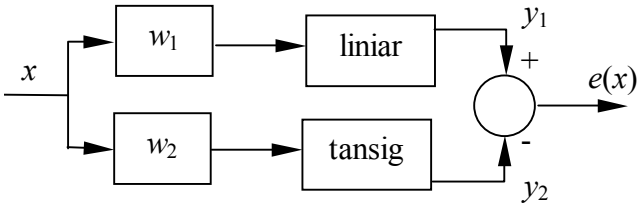


Figura 7.12.1. *Schema bloc a generării semnalelor de eroare din problema 7.12.*

Indicații și răspunsuri:

$$e(x)=y_1(x)-y_2(x)=w_1\cdot x-w_2\cdot\frac{e^x-e^{-x}}{e^x+e^{-x}}\ \Rightarrow\ e'(x)=w_1-w_2\cdot\frac{4}{(e^x+e^{-x})^2}$$

Pentru a discuta soluțiile ecuației studiem semnul lui $e'(x)$.

Cazul 1. Dacă $\frac{w_1}{w_2}<0$ atunci $e'(x)$ are semn constant pe $\mathbb{R}$.

a) Dacă $w_1>0$ și $w_2<0$ atunci $e'(x)>0$ și $e(x)$ este strict crescătoare pe $\mathbb{R}$, de la $-\infty$ la $+\infty$. Ecuația $e(x)=0$ are o singur soluție.

b) Dacă $w_1<0$ și $w_2>0$ atunci $e'(x)<0$ și $e(x)$ este strict descrescătoare pe $\mathbb{R}$, de la $+\infty$ la $-\infty$. Ecuația $e(x)=0$ are o singur soluție..

Cazul 2. Dacă $\frac{w_2}{w_1}>0$ atunci ecuația $e'(x)=0$ are soluții, ceea ce arată că funcția $e(x)$ are puncte de extrem (de minim sau de maxim).

Din $e'(x)=0$ se obține $w_1=\frac{4\cdot w_2}{(e^x+e^{-x})^2}\ \Leftrightarrow\ (e^x+e^{-x})^2=\frac{4\cdot w_2}{w_1}$.

Rezultă $e^x+e^{-x}=2\sqrt{\frac{w_2}{w_1}}$ sau $e^x+e^{-x}=-2\sqrt{\frac{w_2}{w_1}}$.

a) $e^x+e^{-x}=2\sqrt{\frac{w_2}{w_1}}$

Notăm $e^x=t$ și obținem relația $t+\frac{1}{t}=2\sqrt{\frac{w_2}{w_1}}\ \Rightarrow\ t^2-2t\sqrt{\frac{w_2}{w_1}}+1=0$. Se rezolvă ecuația de gradul 2 in t. După ce se determină cele două rădăcini ale acesteia se revine la $e^x=t$.

b) $e^x+e^{-x}=-2\sqrt{\frac{w_2}{w_1}}$

Notăm $e^x=t$ și obținem relația $t+\frac{1}{t}=-2t\sqrt{\frac{w_2}{w_1}}\ \Rightarrow\ t^2+2t\sqrt{\frac{w_2}{w_1}}+1=0$. Se rezolvă ecuația de gradul 2 in t. După ce se determină cele două rădăcini ale acesteia se revine la $e^x=t$.

Problema 7.13.

La intrarea unui neuron tansigmoidal se aplică vectorul prototip $x \in \mathbb{R}$ pentru care vectorul țintă este $z = 1$. Se calculează vectorul delta δ aferent perechii (x, 1) şi se obține $\delta = 1$. Să se determine parametrii neuronului care conduc la această soluție.

Indicaţii şi răspunsuri:

Fie funcție de activare: $\sigma(u) = \dfrac{e^u - e^{-u}}{e^u + e^{-u}}$. Vectorul delta aferent perechii $(x,1)$ este $\delta = 1$.

Eroarea este $e = z - y = z - \sigma(u) = \dfrac{2e^{-u}}{e^u + e^{-u}}$ şi $\delta = qe = \sigma'(u)e$, cu $\sigma'(u) = \dfrac{4}{(e^u + e^{-u})^2}$ $\Rightarrow$

$\delta = \dfrac{8e^{-u}}{(e^u + e^{-u})^3}$. Din $\delta = 1$, rezultă $8e^{-u} = (e^u + e^{-u})^3$, care se rezolvă în raport cu e^u.

Problema 7.14.

Se consideră următoarele trei puncte de forma $\left(x^i, z^i\right) \in \mathbb{R}^2$, $i = \overline{1,3}$: $(-1,-1/2)$, $(0,0)$, $(1,1/2)$.

a. Să se arate că există $\omega, \varphi \in \mathbb{R}$ astfel încât funcția $g(x) = \sin(\omega x + \varphi)$ trece prin cele trei puncte date.

b. Să se arate că există ponderea w şi deplasarea b pentru un neuron liniar, $w, b \in \mathbb{R}$, astfel încât funcția sa intrare-ieşire trece prin cele trei puncte date.

c. Să se arate că există ponderea w şi deplasarea b pentru un neuron tansigmoidal, $w, b \in \mathbb{R}$, astfel încât funcția sa intrare-ieşire trece prin cele trei puncte date.

Indicaţii şi răspunsuri:

a) Se ajunge la sistemul de ecuații $\begin{cases} \sin(-\omega + \phi) = -1/2, \\ \sin(\phi) = 0 \;\Rightarrow\; \phi = k\pi, k \in \mathbb{Z} \\ \sin(\omega + \phi) = 1/2 \end{cases}$

Ținând cont de simetria şi periodicitatea funcției sinus se obțin două soluții:

$$\omega = \arcsin\left(\frac{1}{2}\right) \Rightarrow \omega = \frac{\pi}{6}, \text{ şi } \omega = \arcsin\left(\frac{1}{2}\right) \Rightarrow \omega = \frac{\pi}{6}.$$

b) În cazul unui neuron liniar se ajunge la sistemul de ecuații $\begin{cases} -1/2 = -w + b \\ 0 = b \\ 1/2 = w + b \end{cases} \Rightarrow \begin{cases} b = 0 \\ w = 1/2 \end{cases}$

c) Pentru un neuron tansigmoidal avem: $u = wx + b$, $\sigma(u) = \dfrac{e^u - e^{-u}}{e^u + e^{-u}}$. Cu notația $e^u = a > 0$, rezultă:

$$-1/2=\frac{a-1/a}{a+1/a}\Rightarrow -1/2=\frac{a^2-1}{a^2+1}\Rightarrow -a^2-1=2a^2-2\Rightarrow 3a^2-1=0\Rightarrow a=\frac{1}{\sqrt{3}}$$

$$1/2=\frac{a-1/a}{a+1/a}\Rightarrow 1/2=\frac{a^2-1}{a^2+1}\Rightarrow a^2+1=2a^2-2\Rightarrow a^2-3=0\Rightarrow a=\sqrt{3}$$

$$0=\frac{a-1/a}{a+1/a}\Rightarrow 0=a^2-1\Rightarrow a=1$$

În final se obține $\begin{cases}-\ln(\sqrt{3})=-w+b\\ \ln(\sqrt{3})=w+b\\ \ln(1)=0=b\end{cases}\Rightarrow\begin{cases}w=\ln(\sqrt{3})\\ b=0\end{cases}$

Problema 7.15.

Se consideră trei puncte de forma $\left(x^i,z^i\right)\in\mathbb{R}^2$, $i=\overline{1,3}$: $(-1,-1/2)$, $(0,0)$, $(1,\ 3/4)$.

a.(i). Să se arate că nu există $\omega,\varphi\in\mathbb{R}$ astfel încât funcția $g(x)=\sin(\omega x+\varphi)$ să treacă prin cele trei puncte date.

(ii). Se caută funcția $g(x)$ care minimizează $J(\omega,\varphi)=\frac{1}{2}\sum_{i=1}^{3}\left[g(x^i)-z^i\right]^2$. Să se scrie iterația generală a algoritmului de gradient care minimizează criteriul $J(\omega,\varphi)$, sub forma: $\begin{bmatrix}\omega_{\text{new}}\\ \varphi_{\text{new}}\end{bmatrix}=F\left(\begin{bmatrix}\omega_{\text{old}}\\ \varphi_{\text{old}}\end{bmatrix}\right)$.

b.(i). Să se arate că nu există ponderea w şi deplasarea b pentru un neuron liniar, $w,b\in\mathbb{R}$, astfel încât funcția sa intrare-ieşire $f(x)$ să treacă prin cele trei puncte date.

(ii). Se caută $f(x)$ care minimizează $J(w,b)=\frac{1}{2}\sum_{i=1}^{3}\left[f(x^i)-z^i\right]^2$. Să se scrie iterația generală a algoritmului de antrenare a neuronului.

c. Se reia punctul b pentru un neuron tansigmoidal.

Indicaţii şi răspunsuri:

a.(i). Demonstrăm prin reducere la absurd, ținând cont de paritatea funcției cosinus (pară).
Se ajunge la

$$\begin{cases}\sin(-\omega+\varphi)=-1/2\Rightarrow \sin(\omega-\phi)=\sin(\omega)\cos(-\varphi)+\cos(\omega)\sin(-\varphi)=\sin(\omega)\cos(-\varphi)=1/2\\ \sin(\varphi)=0\Rightarrow\varphi=k\pi,\ k\in\mathbb{Z}\\ \sin(\omega+\varphi)=\sin(\omega)\cos(\varphi)+\cos(\omega)\sin(\varphi)=\sin(\omega)\cos(\varphi)=3/4\end{cases}$$

$\Rightarrow 3/4=1/2\ \ (F)\Rightarrow$ nu există $\omega,\varphi\in\mathbb{R}$ astfel încât funcția $g(x)=\sin(\omega x+\varphi)$ să treacă prin cele trei puncte.

(ii). Relațiile corespunzătoare unei iterații sunt: $\omega_{\text{new}} = \omega_{\text{old}} + \eta \dfrac{\partial J}{\partial \omega}$, $\varphi_{\text{new}} = \varphi_{\text{old}} + \eta \dfrac{\partial J}{\partial \varphi}$,

unde $J = \sum_{i=1}^{3} J^i$, cu $J^i = \dfrac{1}{2}(e^i)^2$, $e^i = y^i - z^i = g(x^i) - z^i$, $i = 1,2,3$.

Se obține:

$$\frac{\partial J}{\partial \omega}\left(\begin{bmatrix}\omega_{\text{old}}\\ \varphi_{\text{old}}\end{bmatrix}\right) = \left[\sin\left([1\ \ -1]\begin{bmatrix}\omega_{\text{old}}\\ \varphi_{\text{old}}\end{bmatrix}\right) + \frac{1}{2}\right]\cos\left([1\ \ -1]\begin{bmatrix}\omega_{\text{old}}\\ \varphi_{\text{old}}\end{bmatrix}\right) +$$

$$= \left[\sin\left([1\ \ 1]\begin{bmatrix}\omega_{\text{old}}\\ \varphi_{\text{old}}\end{bmatrix}\right) - \frac{3}{4}\right]\cos\left([1\ \ 1]\begin{bmatrix}\omega_{\text{old}}\\ \varphi_{\text{old}}\end{bmatrix}\right)$$

$$\frac{\partial J}{\partial \phi}\left(\begin{bmatrix}\omega_{\text{old}}\\ \varphi_{\text{old}}\end{bmatrix}\right) = \left[\sin\left([-1\ \ 1]\begin{bmatrix}\omega_{\text{old}}\\ \varphi_{\text{old}}\end{bmatrix}\right) + 1/2\right]\cos\left([-1\ \ 1]\begin{bmatrix}\omega_{\text{old}}\\ \varphi_{\text{old}}\end{bmatrix}\right) + \frac{1}{2}\sin\left([0\ \ 2]\begin{bmatrix}\omega_{\text{old}}\\ \varphi_{\text{old}}\end{bmatrix}\right)$$

$$+\left[\sin\left([1\ \ 1]\begin{bmatrix}\omega_{\text{old}}\\ \varphi_{\text{old}}\end{bmatrix}\right) - 3/4\right]\cos\left([1\ \ 1]\begin{bmatrix}\omega_{\text{old}}\\ \varphi_{\text{old}}\end{bmatrix}\right)$$

Similar se rezolvă punctele b) şi c) ale problemei.

Problema 7.16.

Se consideră sistemul din figura de mai jos:

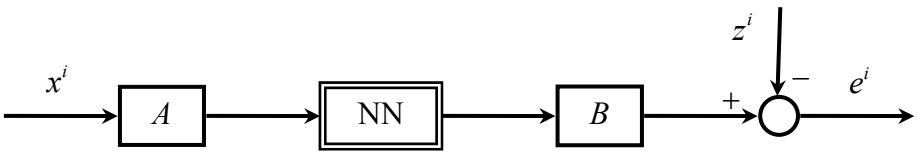


Figura 7.16.1. *Schema bloc a generării semnalelor de eroare din problema 7.16.*

în care $\left(x^i, z^i\right) \in \mathbb{R}^2$, $i = \overline{1,N}$, sunt perechi cunoscute de numere reale, A şi B sunt două valori reale cunoscute, iar NN este un neuron cu funcția de activare sigmoidală, ponderea w şi deplasarea b.

Fie funcția de cost $J = \dfrac{1}{2}\sum_{i=1}^{N}\left(e^i\right)^2$ care trebuie minimizată în raport cu w şi b. Să se scrie o iterație a algoritmului de gradient de forma $\begin{bmatrix}w_{\text{new}}\\ b_{\text{new}}\end{bmatrix} = F\left(\begin{bmatrix}w_{\text{old}}\\ b_{\text{old}}\end{bmatrix}\right)$ pentru cazurile:

a. funcția de activare a neuronului este logsigmoidală;
b. funcția de activare a neuronului este tansigmoidală.

Indicații şi răspunsuri:

Se aplică relațiile din paragraful 5.2.2.

Problema 7.17.

Să se reprezinte următoarele sisteme discrete sub forma unei rețele neuronale dinamice constituită dintr-o ADALINE şi întârzierile adecvate:

a) $G_1(z)=10\dfrac{(z+1)(z-1)}{\left(z-\frac{1}{2}\right)\left(z-\frac{1}{3}\right)}$; b) $G_2(z)=\dfrac{z^{-5}-0.3z^{-6}}{1-0.5z^{-1}+0.06z^{-2}}$;

c) $G_3(z)=\dfrac{b_2z^2+b_1z+b_0}{a_3z^3+a_2z^2+a_1z+a_0}$, $a_i,b_j\neq 0,\ i=\overline{0,3},\ j=\overline{0,2}$;

d) $G_4(z)=z^{-3}G_3(z)$.

Indicații şi răspunsuri:

a) $G_1(z)=10\dfrac{z^2-1}{z^2-\frac{5}{6}z+\frac{1}{6}}\Rightarrow G_1(z^{-1})=10\dfrac{1-z^{-2}}{1-\frac{5}{6}z^{-1}+\frac{1}{6}z^{-2}}$

$y[k]$ depinde de: $y[k-1]$, $y[k-2]$, $u[k]$, $u[k-2]$.

b) $G_2(z)=\dfrac{z^{-5}-0.3z^{-6}}{1-0.5z^{-1}+0.06z^{-2}}$

$y[k]$ depinde de: $y[k-1]$, $y[k-2]$, $u[k-5]$, $u[k-6]$

c) $G_3(z^{-1})=\dfrac{z^3(b_2z^{-1}+b_1z^{-2}+b_0z^{-3})}{z^3(a_3+a_2z^{-1}+a_1z^{-2}+a_0z^{-3})}$, $a_i,b_j\neq 0,\ i=\overline{0,3},\ j=\overline{0,2}$

$y[k]$ depinde de: $y[k-1]$, $y[k-2]$, $y[k-3]$, $u[k-1]$, $u[k-2]$, $u[k-3]$

d) $G_4(z)=z^{-3}G_3(z)$

$y[k]$ depinde de: $y[k-1]$, $y[k-2]$, $y[k-3]$, $u[k-4]$, $u[k-5]$, $u[k-6]$

Problema 7.18.

Se consideră rețeaua neuronală cu dinamică externă din figura 7.18.1, unde rețeaua ADALINE are parametrii:

$$\boldsymbol{w}=[2\quad 0\quad -18\quad \eta\quad -1],\ b=0,$$

iar q^{-1} notează operatorul de întârziere în domeniul timp.

a. Să se determine funcția de transfer a rețelei.
b. Să se discute stabilitatea IMEM în funcție de valoarea parametrului $\eta\in\mathbb{R}$.
c. Pentru valorile lui η pentru care sistemul este stabil IMEM să se arate că există rețele dinamice cu o configurație mai simplă care realizează acelaşi transfer intrare-ieşire şi să se reprezinte grafic aceste configurații.

Indicații şi răspunsuri:

a) Pentru rețeaua ADALINE cu parametrii $\boldsymbol{w}=[2\quad 0\quad -18\quad \eta\quad -1]$ şi $b=0$ se obține

$$y[k]=w_1u[k]+w_2u[k-1]+w_3u[k-2]+w_4y[k-1]+w_5y[k-2]+b\ \Rightarrow$$

$$\Rightarrow y[k]=2u[k]-18u[k-2]+\eta y[k-1]-y[k-2] \Leftrightarrow$$
$$\Leftrightarrow \left(1-\eta z^{-1}+z^{-2}\right)y[k]=\left(2-18z^{-2}\right)u[k].$$

Rezultă $G(z^{-1})=\dfrac{2-18z^{-2}}{1-\eta z^{-1}+z^{-2}} \Rightarrow G(z)=\dfrac{2z^2-18}{z^2-\eta z+1}$

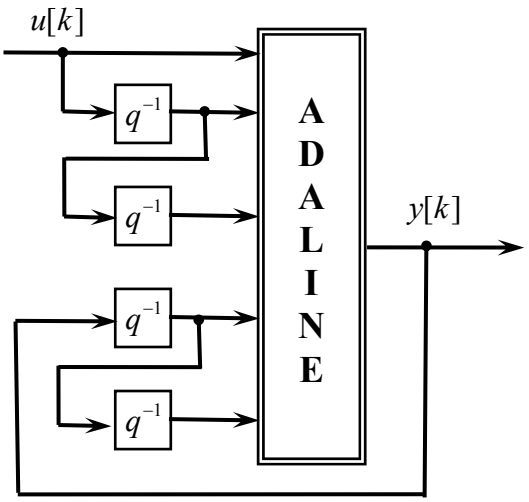


Figura 7.18.1. *Rețeaua neuronală dinamică considerată în problema 7.18.*

b) Un sistem dinamic în timp discret este stabil IMEM dacă şi numai dacă toți polii sunt plasați în interiorul cercului unitate. Polii rețelei neurale cu dinamică externă sunt soluțiile ecuației: $z^2-\eta z+1=0$. Cu notația $\Delta=\eta^2-4$, polii lui $G(z)$ sunt $z_{1/2}=\dfrac{\eta\pm\sqrt{\Delta}}{2}$.

1. Dacă $\eta\in(-\infty,-2]\cup[2,\infty)\Rightarrow\Delta\geq 0$ ecuația va avea soluții reale şi una dintre rădăcini va fi supraunitară în modul.

$$\left.\begin{aligned} z_1&=\frac{\eta+\sqrt{\Delta}}{2}=\frac{\eta}{2}+\frac{\sqrt{\Delta}}{2}\\ \eta\in[2,\infty)&\Rightarrow\frac{\eta}{2}\in[1,\infty)\end{aligned}\right\}\Rightarrow z_1\geq 1,\qquad \left.\begin{aligned} z_1&=\frac{\eta-\sqrt{\Delta}}{2}=\frac{\eta}{2}-\frac{\sqrt{\Delta}}{2}\\ \eta\in(-\infty,-2]&\Rightarrow\frac{\eta}{2}\in(-\infty,-1]\end{aligned}\right\}\Rightarrow z_1\leq -1$$

În aceste situații sistemul nu este stabil IMEM; cu excepția cazului în care polul care nu este poziționat în interiorul cercului unitate este egal cu unul dintre zerourile sistemului.

Zerourile sistemului sunt soluțiile ecuației: $2(z^2-9)=0\Rightarrow 2(z-3)(z+3)=0$

Se observă că:

(i)
$$\left.\begin{aligned} z_1&=\frac{\eta+\sqrt{\Delta}}{2}=\frac{\eta}{2}+\frac{\sqrt{\Delta}}{2}\\ \eta\in[2,\infty)&\Rightarrow\frac{\eta}{2}\in[1,\infty)\\ z_1&=3\end{aligned}\right\}\Rightarrow\frac{\eta}{2}+\frac{\sqrt{\Delta}}{2}=3\Rightarrow\eta+\sqrt{\eta^2-4}=6$$

$$\eta \in [2,6) \Rightarrow \eta^2 - 4 = \eta^2 - 12\eta + 36 \Rightarrow 12\eta = 40 \Rightarrow \eta = \frac{10}{3}$$

Polul sistemului este: $p = \frac{\eta - \sqrt{\eta^2 - 4}}{2} = \frac{\eta + \eta - 6}{2} = \eta - 3 = \frac{1}{3} < 1$

/ii)
$$\left.\begin{array}{l} z_1 = \frac{\eta - \sqrt{\Delta}}{2} = \frac{\eta}{2} - \frac{\sqrt{\Delta}}{2} \\ \eta \in (-\infty, -2] \Rightarrow \frac{\eta}{2} \in (-\infty, -1] \\ z_1 = -3 \end{array}\right\} \Rightarrow \frac{\eta}{2} - \frac{\sqrt{\Delta}}{2} = -3 \Rightarrow \eta - \sqrt{\eta^2 - 4} = -6$$

$$\eta \in (-6, -2] \Rightarrow \eta^2 - 4 = \eta^2 + 12\eta + 36 \Rightarrow 12\eta = -40 \Rightarrow \eta = -\frac{10}{3}$$

Polul sistemului este: $p = \frac{\eta + \sqrt{\eta^2 - 4}}{2} = \frac{\eta + \eta + 6}{2} = \eta + 3 = -\frac{1}{3} > -1$

2. Dacă $\eta \in (-2,2) \Rightarrow \Delta < 0$ sistemul va avea poli complex conjugați cu parte reală subunitară în modul. Pentru ca sistemul sa fie stabil IMEM polii $z_{1,2} = \frac{\eta \pm j\sqrt{-\Delta}}{2}$ trebuie sa fie satisfacă inecuația $(\mathrm{Re}(z_i))^2 + (\mathrm{Im}(z_i))^2 < 1$, adică

$$\left(\frac{\eta}{2}\right)^2 + \left(\frac{\sqrt{4 - \eta^2}}{2}\right)^2 < 1 \Leftrightarrow \frac{\eta^2 + 4 - \eta^2}{4} < 1 \Leftrightarrow 1 < 1 \, (F)$$

În acest caz sistemul nu este stabil IMEM ci se află la limita de stabilitate.

Dacă $\eta \in (-\infty, -2] \cup [2, \infty) - \{-10/3, 10/3\} \Rightarrow$ sistemul nu este stabil IMEM.

Daca $\eta \in \{-10/3, 10/3\} \Rightarrow$ sistemul este stabil IMEM.

Dacă $\eta \in (-2,2) \Rightarrow$ sistemul se afla la limita de stabilitate (nu este stabil IMEM).

c) Daca $\eta = 10/3$ rezultă

$$G(z^{-1}) = \frac{2 - 18z^{-2}}{1 - \eta z^{-1} + z^{-2}} = \frac{2(z-3)(z+3)}{(z-3)(z-1/3)} \Rightarrow G(z) = \frac{2(z+3)}{z - 1/3} = \frac{2 + 6z^{-1}}{1 - 1/3z^{-1}}.$$

Deci $y(k)$ depinde de: $y(k-1), u(k), u(k-1)$.

Dacă $\eta = -10/3$

$$G(z^{-1}) = \frac{2 - 18z^{-2}}{1 - \eta z^{-1} + z^{-2}} = \frac{2(z-3)(z+3)}{(z+3)(z+1/3)} \Rightarrow G(z) = \frac{2(z-3)}{z + 1/3} = \frac{2 - 6z^{-1}}{1 + 1/3z^{-1}}.$$

Deci $y(k)$ depinde de: $y(k-1), u(k), u(k-1)$

Problema 7.19.

Se consideră o instalaţie având funcţia de transfer

$$G(s)=\frac{5\,\mathrm{e}^{-20s}}{10s+1},$$

pentru care se determină un model discret folosind o reţea neuronală dinamică constituită dintr-o ADALINE şi întârzierile necesare.

Să se reprezinte grafic structura reţelei neuronale dinamice şi să se precizeze parametrii ADALINE considerând că discretizarea lui $G(s)$ se realizează exact, adică

$$G_d(z)=\left(1-z^{-1}\right)\cdot Z\left\{\frac{G(s)}{s}\right\},$$

pentru:

a. perioada de eşantionare de 1 sec;

b. perioada de eşantionare de 2 sec.

Indicaţii şi răspunsuri:

$$\frac{G(s)}{s}=\mathrm{e}^{-20s}F(s),\ \text{cu}\ F(s)=\frac{5}{s(10s+1)}\ \Rightarrow\ Z\left\{\frac{G(s)}{s}\right\}=Z\left\{e^{-20s}\right\}Z\{F(s)\}=z^{-\frac{20}{T}}Z\{F(s)\}.$$

$$F(z)=Z\{F(s)\}=\sum \mathrm{Rez}\,F(s)\frac{z}{z-e^{sT}}\Big|_{polesF(s)}$$

$$\left.\begin{aligned}R_1&=\lim_{s\to 0}\frac{z}{z-e^{sT}}\frac{5}{s(10s+1)}s=\frac{5z}{z-1}\\ R_2&=\lim_{s\to -\frac{1}{10}}\frac{z}{z-e^{sT}}\frac{5}{s(10s+1)}(s+1/10)=\frac{z}{z-e^{-\frac{T}{10}}}\frac{5}{-\frac{1}{10}10}=\frac{-5z}{z-e^{-\frac{T}{10}}}\end{aligned}\right\}\Rightarrow F(z)=\frac{5z}{z-1}-\frac{5z}{z-e^{-\frac{T}{10}}}$$

$$G_d(z)=(1-z^{-1})z^{-\frac{20}{T}}\left(\frac{5z}{z-1}-\frac{5z}{z-e^{-\frac{T}{10}}}\right)$$

a) Pentru $T=1\,\mathrm{sec}$:

$$G_d(z)=(1-z^{-1})z^{-20}\left(\frac{5z}{z-1}-\frac{5z}{z-\frac{1}{\sqrt[10]{e}}}\right)=5(1-z^{-1})z^{-19}\left(\frac{1}{z-1}-\frac{1}{z-0.9}\right)=\frac{0.5z^{-20}}{(z-0.9)}\ \Leftrightarrow$$

$$\Leftrightarrow\ G_d(z^{-1})=\frac{0.5z^{-21}}{(1-0.9z^{-1})}\ \Rightarrow\ y[k]=0.9y[k-1]+0.5u[k-21]$$

Modelul neuronal liniar constă dintr-o reţea ADALINE cu parametrii $\boldsymbol{w}=[0.5\ \ 0.9]$, $b=0$, la intrările căreia se aplică $y[k-1]$ şi $u[k-21]$.

b) Pentru $T=2\,\mathrm{sec}$

$$G_d(z)=(1-z^{-1})z^{-10}\left(\frac{5z}{z-1}-\frac{5z}{z-\frac{1}{\sqrt[5]{e}}}\right)=\frac{(z-1)}{z}z^{-10}5z\left(\frac{0.2}{(z-1)(z-0.8)}\right)=\frac{z^{-10}}{(z-0.8)}\Leftrightarrow$$

$$\Leftrightarrow G_d(z^{-1})=\frac{z^{-11}}{(1-0.8z^{-1})}\Rightarrow y[k]=0.9y[k-1]+u[k-11].$$

Modelul neuronal liniar constă dintr-o rețea ADALINE cu parametrii $\boldsymbol{w}=[1\ 0.8]$, $b=0$, la intrările căreia se aplică $y[k-1]$ şi $u[k-11]$.
Parametrii rețelei ADALINE sunt: $w=[1\ \ 0.8];\ b=0$.

Problema 7.20.

Se consideră un proces având funcția de transfer

$$G(s)=\frac{30s+5}{35s^2+12s+1},$$

pentru care se determină un model discret folosind o rețea neuronală dinamică alcătuită dintr-o ADALINE şi întârzierile adecvate.

Să se reprezinte grafic structura rețelei neuronale dinamice şi parametrii ADALINE presupunând că discretizarea lui $G(s)$ se realizează pentru o perioadă de eşantionare $T=1$ sec, folosind:

a. metoda dreptunghiului în avans, care foloseşte $s\simeq\frac{z-1}{T}$;

b. metoda dreptunghiului în întârziere, care foloseşte $s\simeq\frac{z-1}{Tz}$;

c. metoda trapezului, care foloseşte $s\simeq\frac{2}{T}\frac{z-1}{z+1}$.

Indicaţii şi răspunsuri:

a) $G(z)=\frac{30(z-1)+5}{35(z-1)^2+12(z-1)+1}=\frac{30z-25}{35z^2-58z-10}\Leftrightarrow G(z^{-1})=\frac{30/35\,z^{-1}-25/35\,z^{-2}}{1-58/35\,z^{-1}-10/35\,z^{-2}}$.

Modelul neuronal liniar constă dintr-o rețea ADALINE cu parametrii $b=0$ şi $\boldsymbol{w}=1/35[30\ \ -25\ \ 58\ \ 10]$, la intrările căreia se aplică $y[k-1]$, $y[k-2]$, $u[k-1]$, $u[k-2]$.

b) $G(z)=\frac{30\frac{z-1}{z}+5}{35\left(\frac{z-1}{z}\right)^2+12\left(\frac{z-1}{z}\right)+1}=\frac{30z-30+5z}{35(z-1)^2+12z(z-1)+z^2}\frac{z^2}{z}=\frac{35z^2-30z}{48z^2-82z+35}\Leftrightarrow$

$$\Leftrightarrow G(z^{-1})=\frac{35/48-30/48\,z^{-1}}{1-82/48\,z^{-1}+35/48\,z^{-2}}$$

Modelul neuronal liniar constă dintr-o rețea ADALINE cu parametrii $b=0$ şi $\boldsymbol{w}=1/48[35\ \ -30\ \ 82\ \ -35]$, la intrările căreia se aplică $y[k-1]$, $y[k-2]$, $u[k]$, $u[k-1]$.

c) $G(z)=\dfrac{60\dfrac{z-1}{z+1}+5}{140\left(\dfrac{z-1}{z+1}\right)^2+24\left(\dfrac{z-1}{z+1}\right)+1}=\dfrac{60z-60+5z+5}{140(z-1)^2+24(z+1)(z-1)+(z+1)^2}\dfrac{(z+1)^2}{z+1}\Leftrightarrow$

$\Leftrightarrow\ G(z)=\dfrac{65z^2+10z-55}{165z^2-278z+117}\Leftrightarrow\ G(z^{-1})=\dfrac{65/165+10/165\,z^{-1}-55/165\,z^{-2}}{1-278/165\,z^{-1}+117/165\,z^{-2}}$

Modelul neuronal liniar constă dintr-o rețea ADALINE cu parametrii $b=0$ și $\boldsymbol{w}=1/165[65\ 10\ -55\ 278\ 117]$, la intrările căreia se aplică $y[k-1]$, $y[k-2]$, $u[k]$, $u[k-1]$, $u[k-2]$.

Capitolul 8

PROBLEME CU SOLUȚII NUMERICE

Problema 8.1.

Se consideră un neuron cu o intrare (vezi figura 2.1.1). Să se elaboreze un program MATLAB pentru a studia modul de compunere a funcțiilor, considerând la intrarea neuronului valori $x \in [-v, v]$, cu $v = 10$.

Se vor prezenta grafic următoarele trei funcții:

1. $u(x)$ (relația (4)) pentru $x \in [-v, v]$. Pasul de discretizare al intervalului $[-v, v]$ se va alege astfel încât să asigure calitatea plotării;
2. $\sigma(x)$ pentru $u \in [u_{\min}, u_{\max}]$, acest interval reprezentând mulțimea valorilor lui $u(x)$, în condițiile de la punctul 1. Pasul de discretizare al intervalului $[u_{\min}, u_{\max}]$, se va alege astfel încât să asigure calitatea plotării;
3. $f(x) = \sigma(u(x))$ (relația (3)) pentru $x \in [-v, v]$.

Se vor considera cazurile funcției de activare de tip:

a) sigmoid bipolar;
b) treaptă unipolară.

Pentru fiecare din cazurile a) şi b), se vor studia următoarele situații:

(i) $w = 1$, $b = 0$; (ii) $w = 1$, $b = b^*$;

(iii) $w = w^*$, $b = 0$; (iv) $w = w^*$, $b = b^*$;

cu $w^* = \pm 0.5$ şi $b^* = \pm 1$.

Se vor comenta legăturile existente între graficele rezultate la (i) - (iv), iar atunci când este posibil, aceste legături vor fi utilizate pentru *trasarea directă* a graficelor, fără a mai face apel la programul elaborat în MATLAB.

Indicaţii şi răspunsuri:

De exemplu, pentru $w = 0.5$ şi $b = 1$ se obţin graficele din figura 8.1.1.

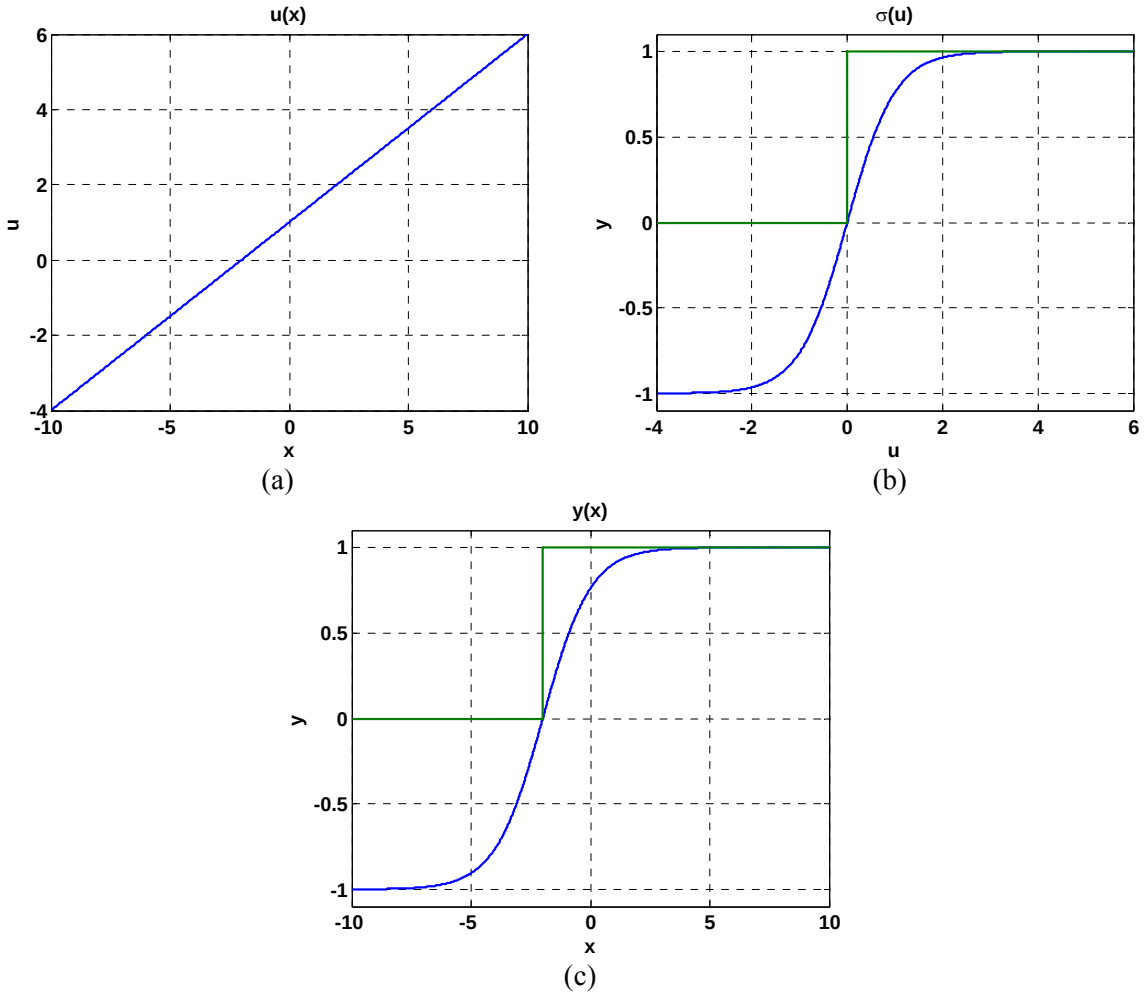


Figura 8.1.1. *Reprezentări grafice ale dependenţei:* (a) $u(x)$, (b) $\sigma(u)$ *şi* (c) $y(x)$, *pentru neuroni cu parametrii* $w = 0.5$ *şi* $b = 1$.

Aceste figuri au fost obţinute utilizând următorul program MATLAB:

```
% Problema 8.1
clear all; close all; clc

v= 5; vs= 0.01;
x= -v:vs:v;
% introducerea parametrilor neuronilor
w= input('pondere w = ');
b= input('deplasare b= ');
% calculul iesirilor
u= w*x+b;
y1= tansig(u);  % sigmoid bipolar
y2= hardlim(u); % treapta unipolara
```

```
% reprezentari grafice
figure(1); plot(x, u);
title('u(x)'); xlabel('x'); ylabel('u'); grid

uv= min(u):vs:max(u);
y1v= tansig(uv);
y2v= hardlim(uv);

figure(2); plot(uv, y1v, uv, y2v)
title('\sigma(u)'); xlabel('u'); ylabel('y')
axis([min(u) max(u) -1.1 1.1]); grid

figure(3); plot(x, y1, x, y2)
title('y(x)'); xlabel('x'); ylabel('y');
axis([-v v -1.1 1.1]); grid
```

Problema 8.2.

Se consideră un neuron cu două intrări (vezi figura 2.1.2 pentru $n = 2$). Să se elaboreze un program MATLAB pentru a studia modul de compunere a funcţiilor, considerând la intrarea neuronului, vectorii: $\boldsymbol{x} = [x_1 \;\; x_2]^T$, pentru $(x_1, x_2) \in [-v1, v1] \times [-v2, v2]$, cu $v1 = v2 = 10$.

Se vor prezenta grafic următoarele trei funcţii:

1. $u(\boldsymbol{x})$ pentru $(x_1, x_2) \in [-v1, v1] \times [-v2, v2]$;
2. $\sigma(u)$ pentru $u \in [u_{\min}, u_{\max}]$, acest interval reprezentând mulţimea în care $u(\boldsymbol{x})$ ia valori, în condiţiile de la punctul 1. Pasul de discretizare *us* al intervalului $[u_{\min}, u_{\max}]$, se va alege astfel încât să asigure calitatea plotării.
3. $f(\boldsymbol{x}) = \sigma(u(\boldsymbol{x}))$ pentru $(x_1, x_2) \in [-v1, v1] \times [-v2, v2]$.

Se vor considera cazurile funcţiei de activare:

a) sigmoid bipolar;

b) treaptă unipolară.

Pentru fiecare din cazurile a) şi b), se vor studia următoarele situaţii:

(i) $\boldsymbol{w} = [1 \;\; 0]$, $b = 0$; (ii) $\boldsymbol{w} = [1 \;\; 0]$, $b = b^*$;

(iii) $\boldsymbol{w} = [0 \;\; 1]$, $b = 0$; (iv) $\boldsymbol{w} = [0 \;\; 1]$, $b = b^*$;

(v) $\boldsymbol{w} = [1 \;\; 1]$, $b = 0$; (vi) $\boldsymbol{w} = [1 \;\; 1]$, $b = b^*$;

(vii) $\boldsymbol{w} = [w_1^* \;\; w_2^*]$, $b = 0$; ($viii$) $\boldsymbol{w} = [w_1^* \;\; w_2^*]$, $b = b^*$;

cu $w_1^*, w_2^* \in \{\pm 0.5, \pm 1, \pm 2\}$ şi $b^* \in \{0, \pm 2\}$.

Se vor comenta legăturile existente între graficele existente la (i) - ($viii$), iar atunci când este posibil, aceste legături vor fi utilizate pentru *trasarea directă* a graficelor, fără a mai face apel la programul elaborat în MATLAB.

Indicații și răspunsuri:

Pentru $w_1 = 0.5$, $w_2 = -0.5$ și $b = 0$ se obțin graficele din figurile 8.2.1 și 8.2.2.

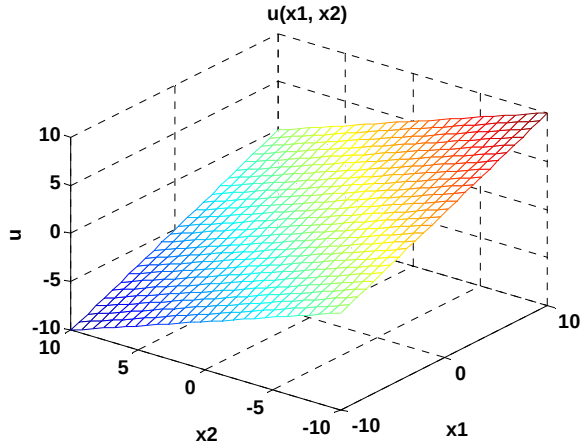
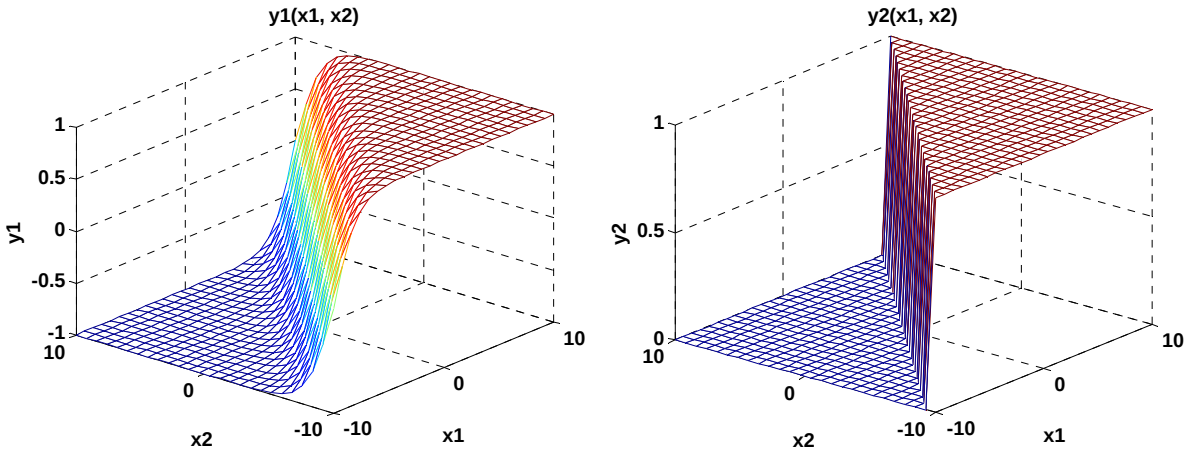


Figura 8.2.1. *Dependența* $u(x_1, x_2)$ *pentru un neuron cu 2 intrări.*



(a) (b)

Figura 8.2.2. *Dependența* $y(x_1, x_2)$ *pentru un neuron cu funcție de activare de tip* (a) *sigmoid bipolar și* (b) *treaptă unipolară.*

Aceste figuri au fost obținute utilizând următorul program MATLAB:

```
% Problema 8.2
clear all; close all; clc

v1= 10; vs1= 0.1; x1= -v1:vs1:v1;
v2= 10; vs2= 0.1; x2= -v2:vs2:v2;
% introducerea parametrilor neuronilor
w1= input('pondere w1 = ');
w2= input('pondere w2 = ');
b= input('deplasare b= ');
% calculul iesirilor
```

```
[X1, X2]= meshgrid(x1, x2);
U= w1*X1 + w2*X2+b;
Y1= tansig(U);
Y2= double(hardlim(U)); % hardlim returneaza valoare logica!!
% reprezentari grafice
figure(1); mesh(X1, X2, U)
title('u(x1, x2)'); xlabel('x1'); ylabel('x2'); zlabel('u')

uv= min(min(U)):vs1:max(max(U));
y1v= tansig(uv); y2v= hardlim(uv);

figure(2); plot(uv, y1v, uv, y2v)
title('\sigma(u)'); xlabel('u'); ylabel('y')
axis([min(min(U)) max(max(U)) -1.1 1.1]); grid

figure(3); mesh(X1, X2, Y1)
title('y1(x1, x2)'); xlabel('x1'); ylabel('x2'); zlabel('y1')

figure(4); mesh(X1, X2, Y2)
title('y2(x1, x2)'); xlabel('x1'); ylabel('x2'); zlabel('y2')
```

Problema 8.3.

Se consideră o rețea neuronală cu un singur strat cu doi neuroni (vezi figura 2.1.3 pentru $n=1$, $p=2$). Să se elaboreze un program MATLAB pentru a studia modul de compunere a funcțiilor, considerând la intrarea rețelei, valorile $x\in[-v,v]$, cu $v=10$.

Se vor prezenta grafic următoarele şase funcţii:

1. $u_1(x)$ şi $u_2(x)$ pentru $x\in[-v,v]$;

2. $\sigma_1(u_1)$ pentru $u_1\in[u_{1\min},u_{1\max}]$, acest interval reprezentând mulţimea valorilor lui $u(x)$, în condiţiile de la punctul 1. Pasul de discretizare al intervalului $[u_{1\min},u_{1\max}]$, se va alege astfel încât să asigure calitatea plotării.

3. $\sigma_2(u_2)$ pentru $u_2\in[u_{2\min},u_{2\max}]$, acest interval reprezentând mulţimea valorilor lui $u_2(x)$, în condiţiile de la punctul 1. Pasul de discretizare al intervalului $[u_{2\min},u_{2\max}]$, se va alege astfel încât să asigure calitatea plotării.

4. $f_1(x)=\sigma_1(u_1(x))$ şi $f_2(x)=\sigma_2(u_2(x))$, pentru $x\in[-v,v]$.

Se vor considera cazurile funcţiei de activare:

a) sigmoid bipolar;

b) treaptă unipolară.

Pentru fiecare din cazurile a) şi b), se vor studia următoarele situaţii:

(i) $\boldsymbol{W}=\begin{bmatrix}1\\0\end{bmatrix}$, $\boldsymbol{b}=\begin{bmatrix}0\\0\end{bmatrix}$; (ii) $\boldsymbol{W}=\begin{bmatrix}1\\0\end{bmatrix}$, $\boldsymbol{b}=\begin{bmatrix}b_1^*\\b_2^*\end{bmatrix}$; (iii) $\boldsymbol{W}=\begin{bmatrix}0\\1\end{bmatrix}$, $\boldsymbol{b}=\begin{bmatrix}0\\0\end{bmatrix}$;

(iv) $\boldsymbol{W}=\begin{bmatrix}0\\1\end{bmatrix}$, $\boldsymbol{b}=\begin{bmatrix}b_1^*\\b_2^*\end{bmatrix}$; (v) $\boldsymbol{W}=\begin{bmatrix}1\\1\end{bmatrix}$, $\boldsymbol{b}=\begin{bmatrix}0\\0\end{bmatrix}$; (vi) $\boldsymbol{W}=\begin{bmatrix}1\\1\end{bmatrix}$, $\boldsymbol{b}=\begin{bmatrix}b_1^*\\b_2^*\end{bmatrix}$;

(*vii*) $\boldsymbol{W}=\begin{bmatrix} w_1^* \\ w_2^* \end{bmatrix}$, $\boldsymbol{b}=\begin{bmatrix} 0 \\ 0 \end{bmatrix}$; (*viii*) $\boldsymbol{W}=\begin{bmatrix} w_1^* \\ w_2^* \end{bmatrix}$, $\boldsymbol{b}=\begin{bmatrix} b_1^* \\ b_2^* \end{bmatrix}$;

cu $w_1^*, w_2^* \in \{0, \pm 0.5, \pm 1, \pm 2\}$ şi $b^* \in \{0, \pm 2\}$.

Se vor comenta legăturile existente între graficele existente la (*i*) – (*viii*), iar atunci când este posibil, aceste legături vor fi utilizate pentru *trasarea directă* a graficelor, fără a mai face apel la programul elaborat în MATLAB. De asemenea, se vor comenta legăturile cu rezultatele obținute la Problema 8.1.

Indicaţii şi răspunsuri:

Programul MATLAB este asemănător celui prezentat la problema 8.1.1.

Problema 8.4.

Se consideră o rețea neurală cu două straturi, cu câte un neuron pe fiecare strat (vezi figura 2.2.2 pentru $n_1 = 1$, $p_1 = n_2 = 1$, $p_2 = 1$). Să se elaboreze un program MATLAB pentru a studia modul de compunere a funcțiilor, considerând la intrarea rețelei, valorile $x \in [-v, v]$, cu $v = 10$. Se vor prezenta grafic următoarele trei funcții:

1. $y^1(x^1)$ pentru $x \in [-v, v]$
2. $y^2(x^2)$ pentru $x^2 \in [y_{\min}, y_{\max}]$, acest interval reprezentând mulțimea valorilor lui $y^1(x^1)$, în condițiile de la punctul 1. Pasul de discretizare ys al intervalului $[y_{\min}, y_{\max}]$ se va alege astfel încât să asigure calitatea plotării.
3. $y^2(x^1)$ pentru $x \in [-v, v]$.

Se vor considera cazurile funcției de activare:

a) sigmoid bipolar pe ambele straturi;

b) treaptă unipolară pe ambele straturi.

Pentru fiecare din cazurile a) şi b), se vor considera valorile $w^1, w^2 \in \{0, \pm 0.5, \pm 1, \pm 2\}$ şi $b^1, b^2 \in \{0, \pm 2\}$. Se vor comenta legăturile cu rezultatele obținute la Problema 8.1.

Indicaţii şi răspunsuri:

Programul MATLAB este asemănător celui prezentat la problema 8.1.1.

Problema 8.5.

Se consideră un perceptron cu două intrări care se utilizează pentru a clasifica următorii 5 vectori bidimensionali:

$$\boldsymbol{x}_1 = \begin{bmatrix} 0.3 \\ -0.4 \end{bmatrix}, \boldsymbol{x}_2 = \begin{bmatrix} -0.4 \\ -0.5 \end{bmatrix}, \boldsymbol{x}_3 = \begin{bmatrix} 0.5 \\ 1.5 \end{bmatrix}, \boldsymbol{x}_4 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \boldsymbol{x}_5 = \begin{bmatrix} -0.3 \\ 0.5 \end{bmatrix},$$

în două clase, notate $\mathscr{C}_1$ şi $\mathscr{C}_2$, astfel:

$$\boldsymbol{x}_1 \in \mathscr{C}_2,\ \boldsymbol{x}_2 \in \mathscr{C}_1,\ \boldsymbol{x}_3 \in \mathscr{C}_2,\ \boldsymbol{x}_4 \in \mathscr{C}_1,\ \boldsymbol{x}_5 \in \mathscr{C}_1.$$

Să se elaboreze un program MATLAB care să creeze şi să antreneze perceptronul pentru un număr de iteraţii stabilit de către utilizator. Se vor reprezenta grafic vectorii de intrare şi se va trasa dreapta definită prin parametrii perceptronului antrenat. În finalul antrenării se va verifica, prin simulare, dacă reţeaua antrenată permite realizarea clasificării.

Se vor raporta următoarele rezultate ale experimentelor:

(*i*) valorile parametrilor iniţiali ai reţelei;
(*ii*) valorile parametrilor reţelei la sfârşitul antrenării;
(*iii*) reprezentarea grafică a vectorilor de intrare şi dreapta de separare a claselor definită de reţea, în urma antrenării;
(*iv*) numărul de iteraţii cât a durat antrenarea;
(*v*) reprezentarea grafică a erorii pătratice globale ca funcţie de numărul de iteraţii parcurse.

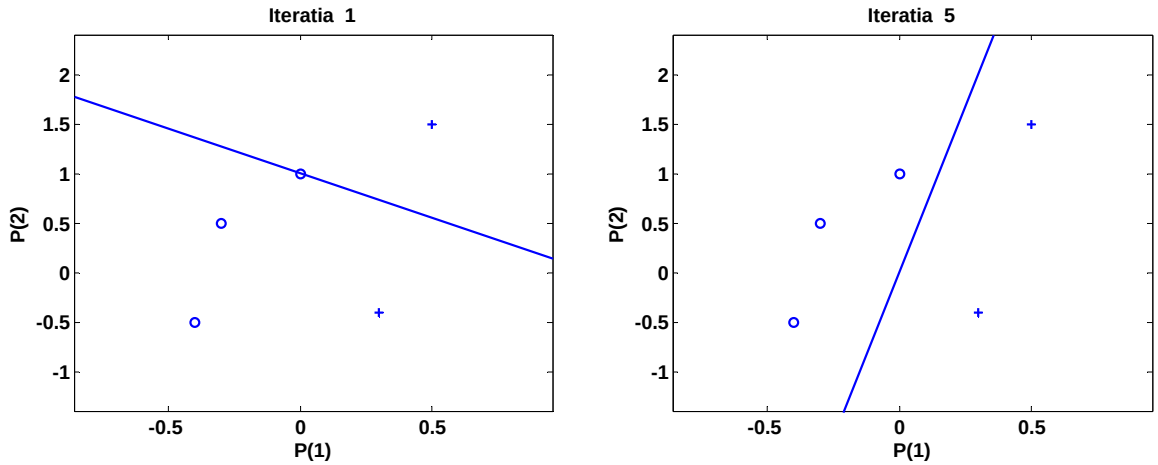
Indicaţii şi răspunsuri:

Se rulează următorul program MATLAB.

```
% Problema 8.5
clear all; close all; clc

% vectori prototip
x1= [0.3; -0.4];      x2= [-0.4; -0.5];
x3= [0.5; 1.5];       x4= [0; 1];
x5= [-0.3; 0.5];
P= [x1 x2 x3 x4 x5];

% vectori tinta
z1= 0;  z2= 1;
T= [z2 z1 z2 z1 z1];

% reprezentare grafica a vectorilor de clasificat
figure(1);  plotpv(P, T);  pause(1);

% creare retea neurala
PR= [min(P')' max(P')'];
net = newp(PR, 1);
w= net.IW{1,1};   % pondere
b= net.b{1};      % deplasare
fprintf('\n Valori initiale parametri retea neurala:');
fprintf('\n\t ponderi: %6.2f, %6.2f \n\t deplasare: %6.2f\n', w(1), w(2), b);

figure(1);  plotpc(w, b);  pause(1);

net.trainParam.epochs = 1;  % numar de epoci pentru antrenare

for i=1:5,
      fprintf('\n Iteratia %d :', i);
      [net, tr, Y, E] = train(net,P,T);  % antrenare retea 1 epoca
      w= net.IW{1,1};   % pondere
```

```
        b= net.b{1};        % deplasare
        fprintf('\n Valori  parametri retea neurala:');
        fprintf('\n\t ponderi: %6.2f, %6.2f \n\t deplasare: %6.2f', w(1), w(2), b);
        perf = mse(E);   % indicator de performanta
        fprintf('\n Valoare indicator performanta MSE: %6.2f\n', perf);
        figure(1); clf;
        plotpv(P, T);  plotpc(w, b);
        str= ['Iteratia  ', num2str(i)];  title(str);
        pause(1)
end
```

Parametrii rețelei neurale sunt inițializați cu valori nule. După o iterație (1 epocă de antrenare a rețelei neurale) se obțin parametrii $\boldsymbol{w} = [0.9 \;\; 1]$, $b = -1$, rețeaua realizând separarea vectorilor prototip prezentată în figura 8.5.1.(a). Dup 5 iterații (5 epoci de antrenare a rețelei neurale) se ajunge la valorile $\boldsymbol{w} = [2 \;\; -0.3]$, $b = 0$, separarea vectorilor prototip fiind prezentată în figura 8.5.1.(b).



(a) (b)

Figura 8.5.1. *Separarea vectorilor prototip realizată de rețeaua neurală antrenată:* (a) *1 epocă*, (b) *5 epoci.*

Problema 8.6.

Să se reia problema 8.5 pentru cazul când se urmăreşte clasificarea celor 5 vectori astfel:

$$\boldsymbol{x}_1 \in \mathscr{C}_2,\ \boldsymbol{x}_2 \in \mathscr{C}_1,\ \boldsymbol{x}_3 \in \mathscr{C}_2,\ \boldsymbol{x}_4 \in \mathscr{C}_1,\ \boldsymbol{x}_5 \in \mathscr{C}_2\ .$$

Indicații şi răspunsuri:

În programul prezentat la problema 8.5 se modifică matricea vectorilor țintă în conformitate cu noua separare dorită a vectorilor prototip:

```
% vectori tinta
z1= 0;  z2= 1;
T= [z2 z1 z2 z1 z2];
```

Se observă că impunând această clasificare, vectorii prototip nu sunt liniar separabili, astfel că separare dorită nu poate fi realizată cu un perceptron.

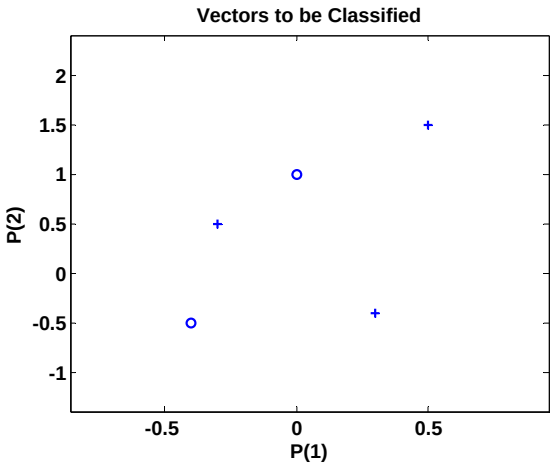


Figura 8.6.1. *Clasificare vectorilor prototip impusă în problema 8.6.*

Problema 8.7.

Se consideră o rețea perceptron cu un singur strat cu doi neuroni, având două intrări, care se utilizează pentru a clasifica următorii 7 vectori bidimensionali:

$$\boldsymbol{x}_1 = \begin{bmatrix} 0.8 \\ 1.6 \end{bmatrix}, \boldsymbol{x}_2 = \begin{bmatrix} 1.25 \\ 1 \end{bmatrix}, \boldsymbol{x}_3 = \begin{bmatrix} 0.3 \\ 0.5 \end{bmatrix}, \boldsymbol{x}_4 = \begin{bmatrix} -1 \\ -1.25 \end{bmatrix}, \boldsymbol{x}_5 = \begin{bmatrix} 0. \\ 0.2 \end{bmatrix}, \boldsymbol{x}_6 = \begin{bmatrix} -0.3 \\ 0.8 \end{bmatrix}, \boldsymbol{x}_7 = \begin{bmatrix} -0.5 \\ -1.5 \end{bmatrix},$$

în patru clase, notate $\mathscr{C}_1, \mathscr{C}_2, \mathscr{C}_3, \mathscr{C}_4$, astfel:

$$\boldsymbol{x}_1 \in \mathscr{C}_1,\ \boldsymbol{x}_2 \in \mathscr{C}_2,\ \boldsymbol{x}_3 \in \mathscr{C}_3,\ \boldsymbol{x}_4 \in \mathscr{C}_4,\ \boldsymbol{x}_5 \in \mathscr{C}_3,\ \boldsymbol{x}_6 \in \mathscr{C}_3,\ \boldsymbol{x}_7 \in \mathscr{C}_4\,.$$

Să se elaboreze un program MATLAB care să creeze şi să antreneze rețeaua.

Se vor reprezenta grafic vectorii de intrare şi se vor trasa cele două drepte definite prin parametrii rețelei antrenate. În finalul antrenării se va verifica, prin simulare, dacă rețeaua antrenată permite realizarea clasificării.

Se vor raporta următoarele rezultate ale experimentelor:

(*i*) valorile parametrilor inițiali ai rețelei;

(*ii*) valorile parametrilor rețelei la sfârşitul antrenării;

(*iii*) reprezentarea grafică a vectorilor de intrare şi dreptele de separare a claselor definite de rețea, în urma antrenării;

(*iv*) numărul de iterații cât a durat antrenarea;

(*v*) reprezentarea grafică a erorii pătratice globale ca funcție de numărul de iterații parcurse.

Pentru antrenarea rețelei neurale se vor considera următorii vectori țintă:

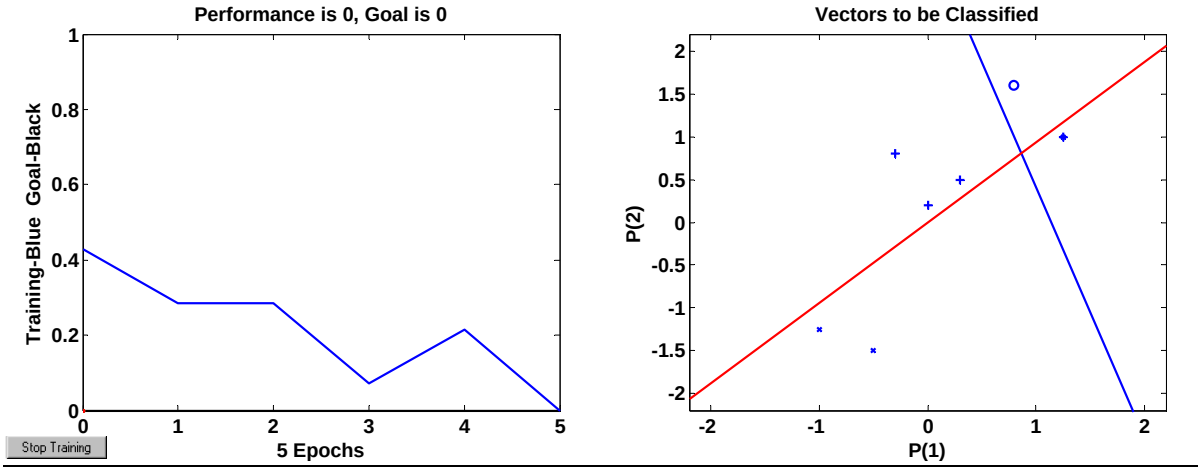
(A) $z_1=\begin{bmatrix}0\\0\end{bmatrix}, z_2=\begin{bmatrix}0\\1\end{bmatrix}, z_3=\begin{bmatrix}1\\0\end{bmatrix}, z_4=\begin{bmatrix}1\\1\end{bmatrix};$ (B) $z_1=\begin{bmatrix}0\\0\end{bmatrix}, z_2=\begin{bmatrix}1\\1\end{bmatrix}, z_3=\begin{bmatrix}0\\1\end{bmatrix}, z_4=\begin{bmatrix}1\\0\end{bmatrix}.$

Ce observați?

Indicații și răspunsuri:

Pentru vectorii țintă considerați în cazul (A) se utilizează programul MATLAB prezentat mai jos. Graficele din figura 8.7.1 prezintă evoluția indicatorului de performanță MAE pe parcursul antrenării rețelei și clasificarea realizată de rețeaua antrenată.

Utilizând vectorii țintă precizați la punctul (B) nu se poate realiza separarea dorită.



(a) (b)

Figura 8.7.1. (a) *Evoluția indicatorului de performanță MAE pe parcursul antrenării rețelei.* (b) *Clasificarea realizată de rețeaua antrenată.*

```
% Problema 8.7.a
clear all; close all; clc

% vectori prototip
x1= [0.8; 1.6];      x2= [1.25; 1];
x3= [0.3; 0.5];      x4= [-1; -1.25];
x5= [0; 0.2];                x6= [-0.3; 0.8];
x7= [-0.5; -1.5];
P= [x1 x2 x3 x4 x5 x6 x7];

% vectori tinta
z1= [0; 0];          z2= [0; 1];
z3= [1; 0];          z4= [1; 1];
T= [z1 z2 z3 z4 z3 z3 z4];
```

```
% reprezentare grafica a vectorilor de clasificat
figure(1);   plotpv(P, T)
% legend('class C_1', 'class C_2', 'class C_3', 'class C_4', 4)
axis([-2.2 2.2 -2.2 2.2])

% creare retea neurala
PR= [min(P')' max(P')'];
net = newp(PR, 2);       % retea cu 2 neuroni

% parametri initiali retea
disp('   Valori initiale parametri retea:');
w= net.IW{1,1}    % ponderile celor 2 neuroni
b= net.b{1,1}       % deplasarile celor 2 neuroni

% antrenare retea
net.trainParam.epochs = 10; % numar de epoci de antrenare
net.trainFcn= 'trainb';
[net, tr, Y, E] = train(net,P,T);
% parametri finali retea
disp('   Valori finale parametri retea:');
w= net.IW{1,1}
b= net.b{1,1}

perf = mse(E);   % indicator de performanta
fprintf('\n Valoare indicator performanta MSE: %6.2f\n', perf);

% reprezentare grafica a clasificarii realizate de reteaua antrenata
figure(1); hold on
plotpc(w, b)
```

Problema 8.8.

Se va studia şi lansa în execuţie programul MATLAB ***arna_wh***, prezentat mai jos, care antrenează un neuron ADALINE cu o singură intrare, utilizând algoritmul Widrow-Hoff, pentru vectorii prototip:

$$x_1 = 0,\ x_2 = 1,$$

şi, respectiv, vectorii ţintă:

$$z_1 = 0,\ z_2 = 2.$$

1. Să se determine valorile parametrilor neuronului utilizând funcţia newlind.
2. Să se determine valorile parametrilor neuronului făcând modificările necesare în programul ***arna_wh*** în următoarele situaţii:
   a) - număr de iteraţii maxime 30,
      - pragul erorii 0.01,
      - rata de învăţare dată de funcţia maxlinlr;
   b) - număr de iteraţii maxime 30,
      - pragul erorii 0,

- rata de învăţare dată de funcţia maxlinlr;

c) - număr de iteraţii maxime 40,
- pragul erorii 0.0001,
- rata de învăţare 0.2;

d) - număr de iteraţii maxime 40,
- pragul erorii 0.01,
- rata de învăţare 0.6;

e) - număr de iteraţii maxime 20,
- pragul erorii 0. 01,
- rata de învăţare 0.8.

Pentru fiecare situaţie se vor preciza valorile selectate iniţial pentru parametrii reţelei, valorile rezultate în final pentru parametrii reţelei, numărul de iteraţii cât a durat optimizarea şi valoarea finală a erorii pătratice globale. Se va preciza condiţia care determină încheierea antrenării. De asemenea se va schiţa suprafaţa erorii pătratice globale şi traseul parcurs în planul parametrilor.

În cazul când se consideră relevant, se pot testa mai multe condiţii iniţiale pentru fiecare din situaţiile a) - e).

Indicaţii şi răspunsuri:

```
% Problema 8.8, program ARNA_WH

clear all; close all; clc

% vectori prototip
x1= 0;   x2= 1;
P= [x1 x2];

% vectori tinta
z1= 0;          z2= 2;
T= [z1 z2];

%%%%%
% a. proiectare retea utilizand functia newlind
net= newlind(P, T);
% parametri retea
fprintf('\n\nValorile ale parametrilor retelei proiectate cu NEWLIND:');
fprintf('\n\t pondere: %g\n\t deplasare: %g', net.IW{1,1}, net.b{1});
% simulare
Y= sim(net, P);
% reprezentare grafica
figure(1);
 plot(P, T, '*r'), hold on;  plot(P, Y, 'd-b')
xlabel('vectori prototip P');
ylabel('vectori tinta T');
fprintf('\n\n<ENTER...>\n');  pause
```

```
%%%%%
% b. crearea unei retele ADALINE ce va fi antrenata cu rata de invatare
% lr= maxlinlr(P)          % valoarea maxima ce asigura convergenta
% algoritmului de antrenare sau cu
lr= input('Rata de invatare: ');% valoarea introdusa de utilizator

lin= newlin(minmax(P), 1, [0], lr);
        % neuronul liniar creat este pregatit pt a fi antrenat cu subrutina "learnwh"
        % (se verifica valorile cerute prin apelarea "help learnwh")
        % eroarea este data de functia "mse" (eroarea medie patratica)

% parametrii specifici antrenarii
iteratii=input('Nr de iteratii= ');
lin.trainParam.epochs=1; % la fiecare iteratie reteaua va fi antrenata 1 epoca
        % pentru a putea reprezenta grafic evolutia parametrilor
lin.trainParam.goal=input('Eroarea medie patratica dorita= ');
lin.trainParam.show=inf;    % eroarea nu este afisata in timpul antrenarii
        % eroarea va fi memorata la fiecare pas si afisarea va fi facuta la sfarsit

% suprafata erorii
w_range = -2:0.4:6;      b_range = -4:0.4:4;
figure;
ES = errsurf(P,T,w_range,b_range,'purelin')/length(T); % valorile erorii medii patratice
plotes(w_range,b_range,ES);

% alegerea cu mouse-ul a parametrilor initiali ai retelei
subplot(1,2,2);
h = text(mean(get(gca,'xlim')),mean(get(gca,'ylim')),'* Alege cu mouse-ul *');
set(h,'horizontal','center', 'fontweight','bold');
[lin.IW{1,1},lin.b{1}] = ginput(1);
        % valorile alese cu mouse-ul devin parametrii initiali ai retelei
fprintf('\n\nValorile initiale ale parametrilor retelei:');
fprintf('\n\t pondere: %g\n\t deplasare: %g', lin.IW{1,1}, lin.b{1});
delete(h);

% antrenarea retelei si reprezentarea la fiecare pas (epoca)
% a parametrilor pe suprafata de eroare
Err(1)= mse(T-sim(lin,P));   % eroarea medie patratica inainte de inceperea antrenarii;
        % vectorul Err va contine evolutia erorii
Wi(1)= lin.IW{1,1};        % vectorul care va contine evolutia ponderilor
bi(1)= lin.b{1};      % vectorul care va contine evolutia deplasarilor
h= plotep(lin.IW{1,1}, lin.b{1}, Err(1));
        % parametrii initiali reprezentati pe suprafata de eroare
        % (cu eroarea corespunzatoare)
fprintf('\nEroarea medie patratica initiala = %g\n', Err(1))    ;

for i= 1:iteratii,
        [lin, tr, Y, E] = train(lin,P,T);
        h= plotep(lin.IW{1,1}, lin.b{1}, tr.perf(1),h);
                % evolutia parametrilor pe suprafata de eroare
        Err(i)= tr.perf(1); % eroarea pt noile valori ale parametrilor
```

```
        Wi(i+1)= lin.IW{1,1};
        bi(i+1)= lin.b{1};
        if Err(i) <= lin.trainParam.goal
            break;  % iesire pe conditia de atingere a pragului de eroare impus
        end
    end

    iteratii=i;     % numarul de epoci cat a durat antenarea
    fprintf('\n\nAntrenarea a durat %g epoci\n', iteratii);
    fprintf('\n\nValorile finale ale parametrilor retelei:');
    fprintf('\n\t pondere: %g\n\t deplasare: %g', lin.IW{1,1}, lin.b{1});
    fprintf('\nEroarea medie patratica finala = %g\n', Err(end))  ;

    % evolutia erorii
    disp('Apasati o tasta pentru a vedea evolutia erorii functie de nr de epoci...');  pause;
    figure;
    semilogy((1:iteratii), Err);
    xlabel('iteratii');
    ylabel('MSE');
    title('Eroarea medie patratica functie de numarul de iteratii');

    % dinamica aproximarii
    disp('Apasati o tasta pentru a vedea aproximarea initiala...');  pause;
    figure;
    plot(P,T,'r+'); % punctele prin care se doreste aproximarea
    xlabel('vectori prototip P');
    ylabel('vectori tinta T');
    hold on;
    h= plot(P, Wi(1)*P+bi(1), 'd-b');

    disp('Apasati o tasta pentru a vedea evolutia dinamica a aproximarii...');  pause;
    for i=1:iteratii
        pause(0.5);
        delete(h);
        h= plot(P, Wi(i+1)*P+bi(i+1), 'd-b');
        title(['Iteratia ' num2str(i)])
    end

    hold off;
```

Prin proiectare directă se obține rețeaua ADALINE cu parametrii $w = 2$ şi $b = 0$. Pentru cei doi vectori prototip (denumire generică, având de fapt valori scalare) utilizați se observă că ieşirile rețelei coincid cu valorile țintă impuse (figura 8.8.1), funcția aproximată de rețeaua neurală fiind efectiv liniară.

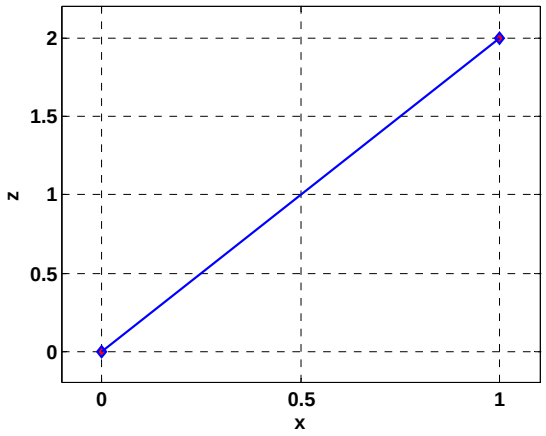


Figura 8.8.1. *Comparație între valorile țintă impuse şi ieşirile rețelei ADALINE proiectate corespunzătoare valorilor prototip considerate în problema 8.8.*

Figura 8.8.2 prezintă evoluția parametrilor rețelei ADALINE pe suprafața de eroare pe parcursul a 20 de epoci de antrenare cu rata de învățare $\eta = 0.6$. Evoluția erorii pătratice medii funcție de iterație este ilustrată în figura 8.8.3. Valorile parametrilor rețelei ADALINE la care se ajunge în urma antrenării sunt $w = 2.01191$ şi $b = -0.00732$, care asigură o eroare medie pătratică de $2.0995\,\mathrm{e}-005$.

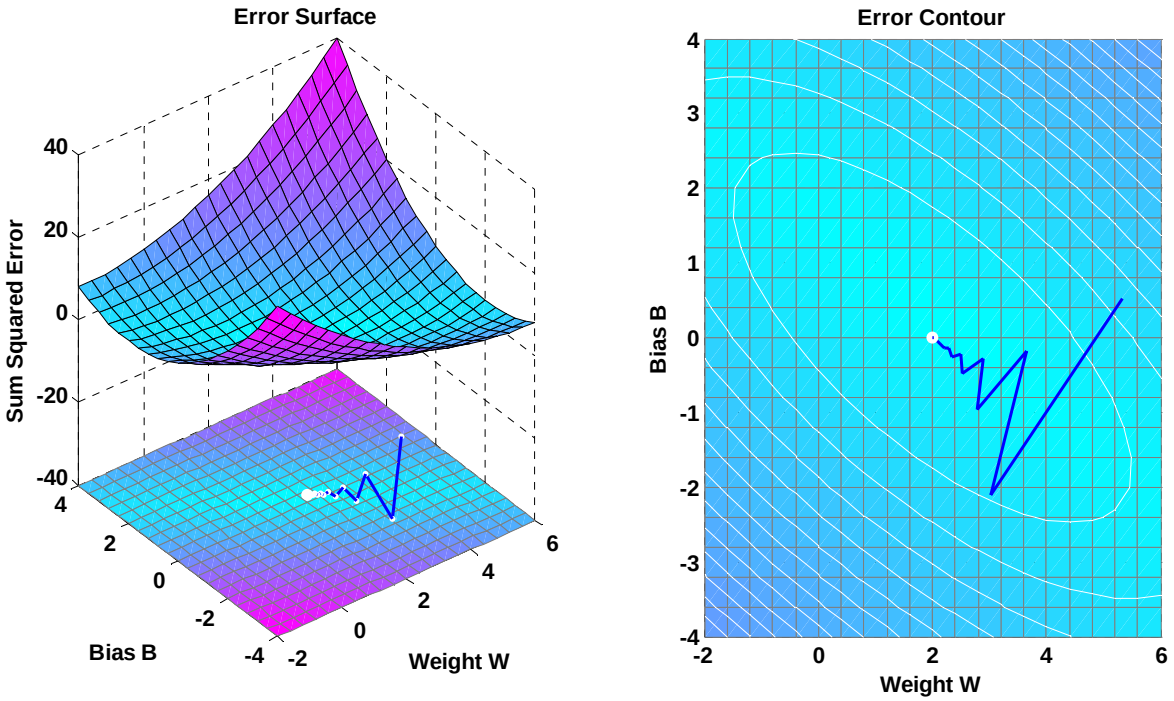


Figura 8.8.2. *Evoluția parametrilor rețelei ADALINE pe suprafața de eroare pe parcursul a 20 de epoci de antrenare cu rata de învățare* $\eta = 0.6$.

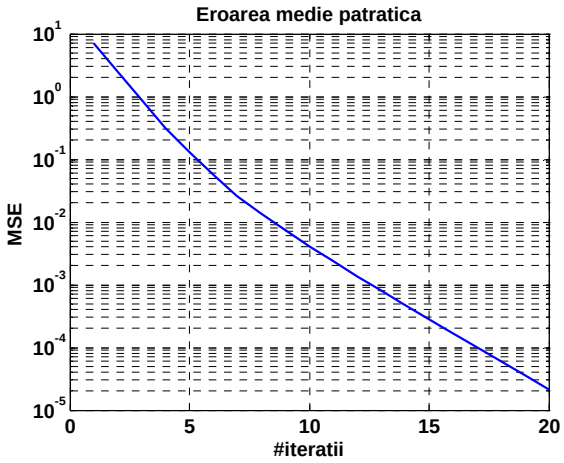


Figura 8.8.3. *Evoluția erorii pătratice medii pe parcursul a 20 de epoci de antrenare cu rata de învățare* $\eta = 0.6$ *pentru rețeaua ADALINE din problema 8.8.*

Problema 8.9.

Se va relua Problema 8.8 pentru un singur vector prototip $x_1 = 1$ şi, respectiv, un singur vector țintă $z_1 = 2$, pentru situațiile 2.a) şi 2.e).

Indicaţii şi răspunsuri:

În programul prezentat la problema 8.8 se modifică în mod corespunzător matricea vectorilor prototip şi cea a vectorilor țintă. Se observă că există o infinitate de rețele ADALINE care satisfac cerințele impuse în problemă.

Problema 8.10.

Se va relua Problema 8.8 pentru vectorii prototip:

$$x_1 = 0,\ x_2 = 1,\ x_3 = -1,$$

şi, respectiv, vectorii țintă:

$$,\ z_1 = 0,\ z_2 = 2,\ z_3 = -1,$$

pentru situațiile 2.a), 2.c) şi 2.d).

Indicaţii şi răspunsuri:

În programul prezentat la problema 8.8 se modifică în mod corespunzător matricea vectorilor prototip şi cea a vectorilor țintă.

```
% vectori prototip
x1= 0;          x2= 1;  x3= -1;
P= [x1  x2  x3];
% vectori tinta
z1= 0;          z2= 2;  z3= -1;
T= [z1  z2  z3];
```

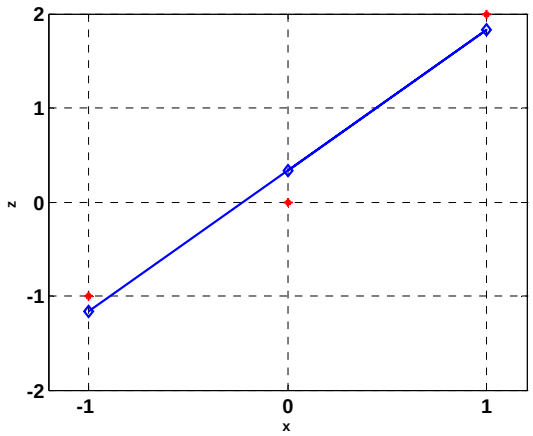


Figura 8.10.1. *Comparație între valorile țintă impuse şi ieşirile rețelei ADALINE proiectate corespunzătoare valorilor prototip considerate în problema 8.10.*

Prin proiectare directă se obține rețeaua ADALINE cu parametrii $w = 1.5$ şi $b = 0.33$. Această rețea aproximează cel mai bine, în sensul celor mai mici pătrate, funcția neliniară precizată prin cele trei perechi prototip-țintă precizată în enunțul problemei.

Problema 8.11.

Se consideră amplificatorul operațional din fig. 1 care posedă în reacție rezistența $R = 100\,\mathrm{k}\Omega$ şi la intrare rezistențele R_1, R_2, R_3.

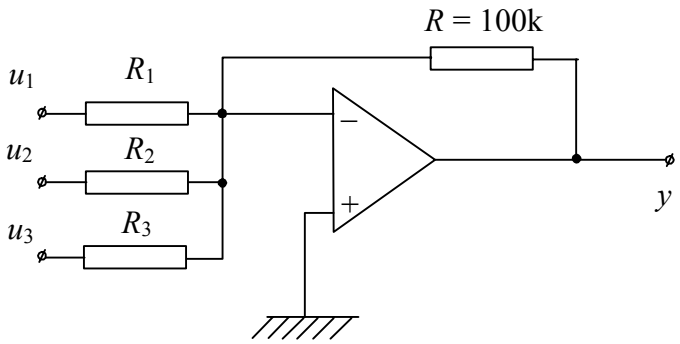


Figura 8.11.1. *Schema amplificatorului operațional utilizat în Problema 8.11.*

La intrarea u_1 se aplică o tensiune constantă de 2V. Pe intervalul de timp cuprins între 0 s şi 5 s, pe intrările u_2 şi u_3 se aplică două semnale de frecvență joasă, care, eşantionate cu perioada $T = 0.01$ s, sunt memorate în fişierul de date ***opamp_wh.mat*** în vectorii ***uuu2*** şi respectiv ***uuu3***. Semnalul obținut la ieşirea y, eşantionat cu aceeaşi perioadă, este memorat în vectorul ***yyy*** din fişierul de date ***opamp_wh.mat***.

Utilizând datele din fişierul ***opamp_wh.mat*** să se determine valorile rezistențelor necunoscute R_1, R_2, R_3 prin următoarele metode:

a) obţinând reţeaua ADALINE prin proiectare directă, apelând funcţia ***newlind***.
b) antrenând o reţea ADALINE cu arhitectură adecvată, utilizând funcţiile ***newlin*** şi ***train***;

Indicaţii şi răspunsuri:

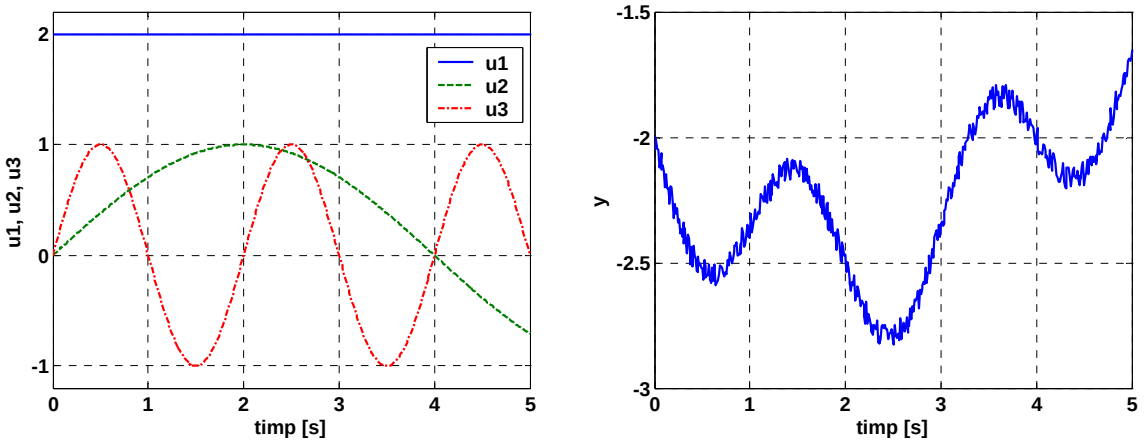
Pentru estimarea valorilor solicitate în enunţul problemei se ţine cont de dependenţa ieşirii amplificatorului operaţional de cele trei intrări ale sale:

$$y(u_1,u_2,u_3) = -\frac{R}{R_1}u_1 - \frac{R}{R_2}u_2 - \frac{R}{R_3}u_3 .$$

Astfel, reţeaua ADALINE ce modelează funcţionarea acestui amplificator operaţional va avea 1 neuron cu 3 intrări, ale cărui ponderi reprezintă $w_1 = -R/R_1$, $w_2 = -R/R_2$, respectiv $w_3 = -R/R_3$. Deplasarea neuronului ar trebui să fie nulă.

```
% Problema 8.11
clear all;  close all;  clc
load opamp_wh

t= 0:0.01:5;
% vectori prototip
uuu1= 2*ones(size(uuu2));
P= [ uuu1; uuu2; uuu3];
% vectori tinta
T= yyy;
% reprezentare grafica a intrarilor si iesirii functie de timp
figure(1); plot(t, P); legend ('u1', 'u2', 'u3')
figure(2); plot(t, T);

% a. proiectare retea ADALINE utilizand functia newlind
net= newlind(P, T);
% parametri retea proiectata
wd= net.IW{1};    bd= net.b{1};
% valori estimate ale rezistentelor R1, R2 si R3
R= 100;
R1d= -R/wd(1)
R2d= -R/wd(2)
R3d= -R/wd(3)

% b. creare retea ADALINE si antrenare
lr= maxlinlr(P)    % rata de invatare

lin= newlin(minmax(P),1,[0],lr);

% parametrii specifici antrenarii
lin.trainParam.epochs= input('Nr de epoci de antrenare = ');
lin.trainParam.goal= input('Eroarea medie patratica dorita= ');
lin.trainParam.show= 5;

lin = train(lin,P,T);
```

```
% parametri retea antrenata
wt= lin.IW{1};
bt= lin.b{1};
% valori estimate ale rezistentelor R1, R2 si R3
R1t= -R/wt(1)
R2t= -R/wt(2)
R3t= -R/wt(3)
```



(a) (b)

Figura 8.11.2. *Reprezentarea grafică a semnalelor* (a) *de intrare şi* (b) *de ieşire corespunzătoare amplificatorului operaţional studiat în problema 8.11.*

Reprezentarea grafică a semnalelor de intrare şi de ieşire pentru amplificatorul operaţional este prezentată în figura 8.11.2.

Prin proiectarea directă a reţelei ADALINE se obţin valorile $R_1^d = 99.85$, $R_2^d = 201.47$ şi $R_3^d = 301.21$. Pentru antrenarea reţelei neurale, rata maximă de învăţare are valoarea 4.7356e-04. După 100 de epoci de antrenare se ajunge la valoarea $MSE = 0.000816$ şi se obţin valorile $R_1^t = 124.73$, $R_2^t = 202.56$ şi $R_3^t = 302.125$.

Mărirea numărului de epoci de antrenare nu aduce îmbunătăţiri substanţiale în calitate aproximării valorii rezistenţei R_1. Aceasta se datorează faptului că pe una dintre liniile matricei prototipurilor sunt elemente constante.

Problema 8.12.

Se va studia şi lansa în execuţie programul MATLAB ***arna_bp1***, prezentat mai jos, care antrenează (prin aplicarea regulii delta) un neuron cu funcţie sigmoidală bipolară, pentru a aproxima o funcţie neliniară φ definită prin vectorii prototip:

$$x_i = \frac{(i-1)\pi}{12}, \quad i = 1,...,7,$$

şi vectorii ţintă:

$$z_i = \sin(x_i), \quad i = 1,...,7 .$$

Să se opereze modificările necesare în program asupra ratei de învăţare şi a numărului de iteraţii pentru a realiza antrenarea, pornind de la condiţiile iniţiale situate în următoarele domenii:

1. $(w_0, b_0) \in [-4.5, -1.5] \times [-4.5, -1.5]$
2. $(w_0, b_0) \in [-1.5, 1.5] \times [-4.5, -1.5]$
3. $(w_0, b_0) \in [1.5, 4.5] \times [-4.5, -1.5]$
4. $(w_0, b_0) \in [-4.5, -1.5] \times [-1.5, 1.5]$
5. $(w_0, b_0) \in [-1.5, 1.5] \times [-1.5, 1.5]$
6. $(w_0, b_0) \in [1.5, 4.5] \times [-1.5, 1.5]$
7. $(w_0, b_0) \in [-4.5, -1.5] \times [1.5, 4.5]$
8. $(w_0, b_0) \in [-1.5, 1.5] \times [1.5, 4.5]$
9. $(w_0, b_0) \in [1.5, 4.5] \times [1.5, 4.5]$

În fiecare din situaţiile (1) - (9) se vor preciza:

a) valorile iniţiale ale parametrilor, reprezentarea grafică a aproximării realizate de reţea pe baza acestor valori şi eroarea corespunzătoare;
b) valorile finale ale parametrilor (rezultate din antrenare), reprezentarea grafică a aproximării realizate de reţea pe baza acestor valori şi eroarea corespunzătoare;
c) numărul total de iteraţii cât a durat antrenarea şi reprezentarea grafică a evoluţiei erorii ca funcţie de numărul de iteraţii;
d) traiectoria în planul parametrilor (w, b) – reprezentare grafică;
e) valorile utilizate pentru parametrii de antrenare comentate prin prisma convergenţei algoritmului şi a formei suprafeţei erorii. Se vor raporta şi explica eventualele eşecuri în antrenare, datorate alegerii neadecvate a parametrilor de antrenare.

Indicaţii şi răspunsuri:

```
% Problema 8.12, program ARNA_BP1
clear all; close all; clc

% definirea prototipurilor
P = 0:pi/12:pi/2;
% definirea tintelor
T = sin(P);

% reprezentarea grafica a suprafatei erorii
w_range = -4.5:0.5:4.5;
b_range = -4.5:0.5:4.5;
figure
ES = errsurf(P, T, w_range, b_range, 'tansig')/length(T); %valorile erorii medii patratice
plotes(w_range, b_range, ES, [50 25]);

%  crearea neuronului tansigmoidal, ce urmeaza a fi antrenat cu
%  subrutina 'learngd' (metoda gradientului)
sigm=newff(minmax(P), [1], {'tansig'}, 'traingd');
```

```
% parametrii specifici antrenarii
sigm.trainParam.epochs= 1; % antrenarea este efectuata epoca cu epoca
iteratii= input('Numar de epoci de antrenare:  ');
sigm.trainParam.goal= 1.5e-4;
sigm.trainParam.show= inf    % eroarea nu este afisata in timpul antrenarii
        % eroarea va fi memorata la fiecare pas si afisarea va fi facuta la sfirsit
sigm.trainParam.lr= input('Valoarea ratei de invatare: ');

% selectarea valorilor initiale ale parametrilor neuronului cu mouse-ul
subplot(1,2,2);
h = text(mean(get(gca,'xlim')), mean(get(gca,'ylim')), '* Alege cu mouse-ul *');
set(h,'horizontal','center', 'fontweight','bold');
[sigm.IW{1,1}, sigm.b{1}] = ginput(1);
        % valorile alese cu mouse-ul devin parametrii initiali ai retelei

fprintf('\n Valori initiale parametri retea neurala:');
fprintf('\n\t pondere: %6.2f\n\t deplasare: %6.2f\n', sigm.IW{1,1}, sigm.b{1});
delete(h);

pause % Apasa o tasta pentru a antrena neuronul...

% Antrenarea retelei si reprezentarea la fiecare pas (epoca)
% a parametrilor pe suprafata de eroare
Err(1)= mse(T-sim(sigm, P));
        % eroarea medie patratica inainte de inceperea antrenarii;
        % vectorul Err va contine evolutia erorii
Wi(1)= sigm.IW{1,1};    % vectorul care va contine evolutia ponderilor
bi(1)= sigm.b{1};          % vectorul care va contine evolutia deplasarilor
h= plotep(sigm.IW{1,1}, sigm.b{1}, Err(1));
        % parametrii initiali reprezentati pe suprafata de eroare

for i=1:iteratii,
        [sigm,tr]= train(sigm,P,T);
        h= plotep(sigm.IW{1,1}, sigm.b{1}, tr.perf(1), h);
                %evolutia parametrilor pe suprafata de eroare
        title(['Iteratia: ' num2str(i)]);
        Err(i+1)= tr.perf(1);   %eroarea pt noile valori ale parametrilor
        Wi(i+1)= sigm.IW{1,1};
        bi(i+1)= sigm.b{1};
        if (Err(i)<=sigm.trainParam.goal),
        break; %iesire pe conditia de atingere a pragului de eroare impus
        end
end

iteratii= i; %numarul de epoci cit a durat antenarea
fprintf('\n\nAntrenarea a durat %g iteratii\n', iteratii);
fprintf('\n\nValorile finale ale parametrilor retelei:');
fprintf('\n\t pondere: %g\n\t deplasare: %g', sigm.IW{1,1}, sigm.b{1});
fprintf('\nEroarea medie patratica finala = %g\n', Err(end));
%evolutia erorii
```

```
disp('Apasati o tasta pentru a vedea evolutia erorii functie de nr de epoci...');
pause;
figure;
semilogy((0:iteratii), Err);
xlabel('#iteratii');
ylabel('MSE');
title('Eroarea medie patratica functie de numarul de iteratii');

%dinamica aproximarii
disp('Apasati o tasta pentru  a vedea aproximarea initiala...');
pause;

figure;
plot(P,T,'r+'); %punctele prin care se doreste aproximarea
xlabel('vectori prototip P');
ylabel('vectori tinta T');
hold on;
h=plot(P,tansig(Wi(1)*P+bi(1)));
title('Aproximarea initiala');

disp('Apasati o tasta pentru a vedea evolutia dinamica a aproximarii...');
pause;

for i=1:iteratii,
        pause(0.2);
        delete(h);
        h=plot(P, tansig(Wi(i+1)*P+bi(i+1)),'b*-');
        title(['Iteratia:',num2str(i)]);
end

hold off;
```

Figura 8.12.1 prezintă evoluţia parametrilor neuronului tansigmoidal pe suprafaţa de eroare pe parcursul a 100 de epoci de antrenare cu rata de învăţare $\eta = 0.9$. Figurile 8.12.2.(a) şi 8.12.3.(a) prezintă evoluţia erorii medii pătratice MSE pe parcursul a 100 de epoci de antrenare cu rata de învăţare $\eta = 0.9$ şi, respectiv, $\eta = 0.2$, pornind de la aceleaşi valori iniţiale ale parametrilor neuronului. Aceste grafice permit vizualizarea efectului pe care îl are rata de învăţare asupra progresului antrenării.

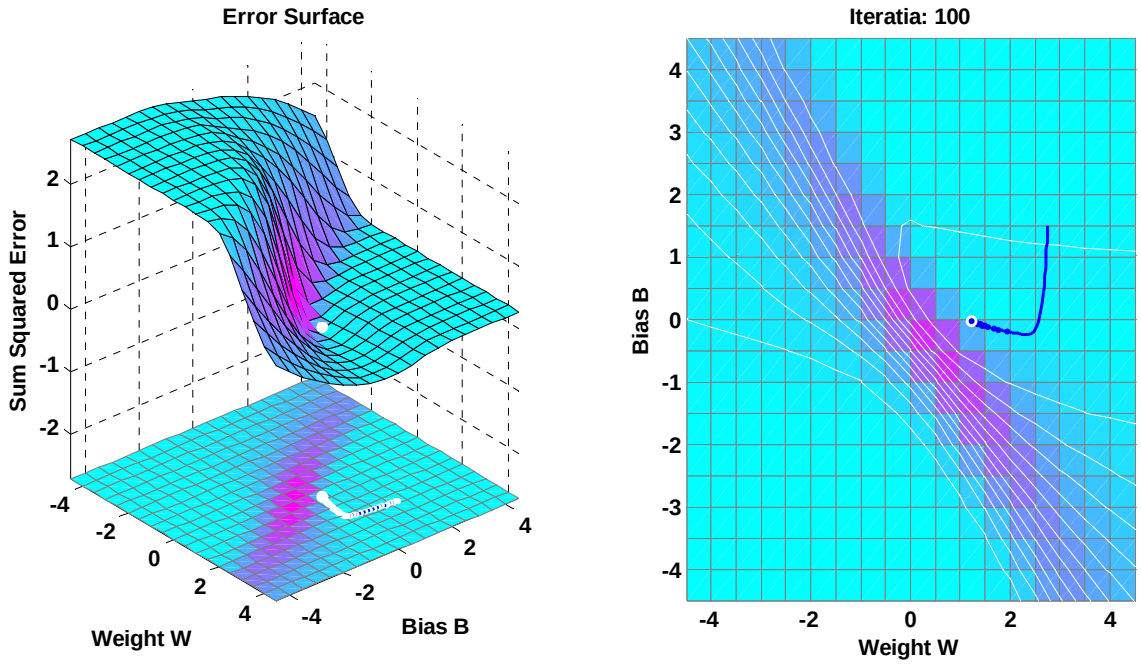
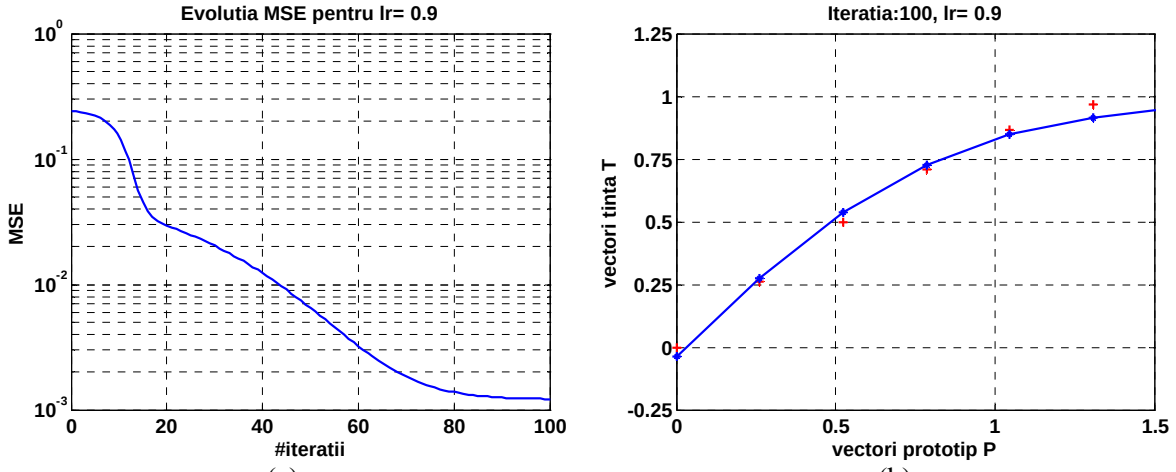


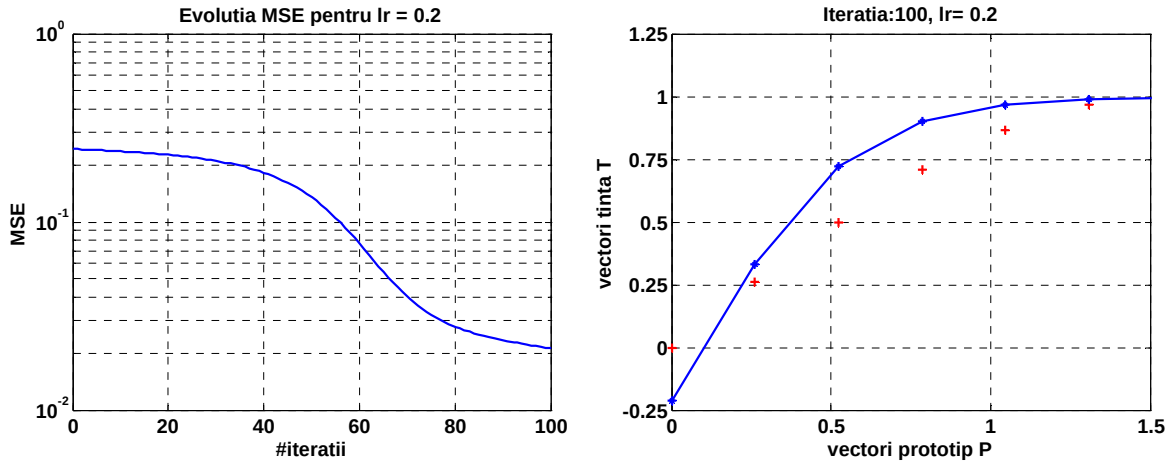
Figura 8.12.1. *Evoluția parametrilor neuronului tansigmoidal pe suprafața de eroare pe parcursul a 100 de epoci de antrenare cu rata de învățare* $\eta = 0.9$.

Figurile 8.12.2.(b) şi 8.12.3.(b) prezintă aproximarea realizată de neuronul rezultat după 100 de epoci de antrenare cu rata de învățare $\eta = 0.9$ şi, respectiv, $\eta = 0.2$. Aceste grafice permit vizualizarea semnificației MSE în aprecierea calității aproximării realizate.



(a) (b)

Figura 8.12.2. (a) *Evoluția erorii medii pătratice MSE pe parcursul a 100 de epoci de antrenare cu rata de învățare* $\eta = 0.9$. (b) *Aproximarea realizată de neuronul antrenat.*



(a) (b)

Figura 8.12.3. (a) *Evoluția erorii medii pătratice MSE pe parcursul a 100 de epoci de antrenare cu rata de învățare* $\eta = 0.2$. (b) *Aproximarea realizată de neuronul antrenat.*

Problema 8.13.

Să se utilizeze programul ***arna_bp1*** prezentat în problema 8.12 (cu modificările necesare considerate) pentru a selecta o regiune din planul parametrilor (w,b) şi o gamă de valori a ratei de învățare care să asigure antrenarea eficientă pentru perechile de vectori (prototip, țintă):

$$x_i = \frac{\pi}{2} + \frac{(i-1)\pi}{12};\ \ z_i = \sin(x_i);\ \ i = 1,...,7.$$

Să se comenteze criteriile care au condus la selectarea respectivă, prin referire la convergența algoritmului şi forma suprafeței erorii.

Indicații şi răspunsuri:

În programul ***arna_bp1*** se modifică în mod corespunzător matricea vectorilor prototip şi cea a vectorilor țintă.

```
% definirea prototipurilor P:
P = pi/2 : pi/12 : pi;
% definirea tintelor T:
T = sin(P);
```

Problema 8.14.

Se va studia şi lansa în execuție programul MATLAB ***arna_bp2*** care antrenează o rețea cu două straturi (cu noduri tansigmoidale pe primul strat şi liniare pe al doilea) pentru a aproxima o funcție neliniară $\varphi : \mathbb{R} \to \mathbb{R}$ cunoscută prin intermediul perechilor de vectori prototip-țintă:

$$(x_i, z_i), \quad z_i = \varphi(x_i), \quad i = 1,...,N.$$

Se consideră vectorii ţintă:

$$x_i = (i-1)/50, \quad i = 1,...,51,$$

şi vectorii prototip:

$$z_i = \varphi(x_i) = 1 + \exp(-x_i)\sin(2\pi x_i - \frac{\pi}{2}), \quad i = 1,...,51.$$

Antrenarea reţelei se va face folosind mai întâi metoda gradientului (rutina '**traingd**'), iar apoi algoritmul Levenberg-Marquardt (rutina '**trainlm**'). În fiecare din cele două cazuri se vor studia următoarele aspecte:

a) Operând modificările necesare în programul ***arna_bp2*** să se studieze rolul următorilor factori asupra eficienţei antrenării reţelei:
   - numărul de neuroni în stratul de intrare;
   - rata de învăţare (doar în cazul folosirii rutinei '**traingd**').

   Aprecierea eficienţei se va face în termenii erorii pătratice globale obţinute în urma antrenării şi a numărului de iteraţii necesare pentru atingerea unui prag impus pentru eroare.

   Se va comenta modul de alegere a valorilor iniţiale pentru parametrii reţelei.

b) Pentru testele de antrenare eficiente realizate, se vor preciza valorile parametrilor reţelei rezultaţi în urma antrenării. De asemenea se va testa capacitatea de generalizare a reţelei antrenate considerând câteva perechi prototip-ţintă (x, z) corespunzătoare funcţiei φ „nevăzute" de reţea pe parcursul antrenării (corespunzătoare unor valori x situate în afara intervalului $[0, 1]$ folosit în antrenare).

Indicaţii şi răspunsuri:

```
% Problema 8.14, program ARNA_BP2
clear all; close all; clc;

% definirea prototipurilor P
P = 0:0.02:1;

% definirea tintelor T
T = 1 + exp(-P).*sin(2*pi*P - pi/2);

% reprezentarea grafica a perechilor  (prototip, tinta)
figure;  plot(P, T, 'r+');
xlabel('Vectori prototip P');
ylabel('Vectori tinta T');
title('Functia neliniara de aproximat')

pause % Tasteaza pentru a dimensiona reteaua
neur= input('\n\nNumarul de neuroni de intrare: ');

disp('Selectare metoda de antrenare:  ');
met = 0;
```

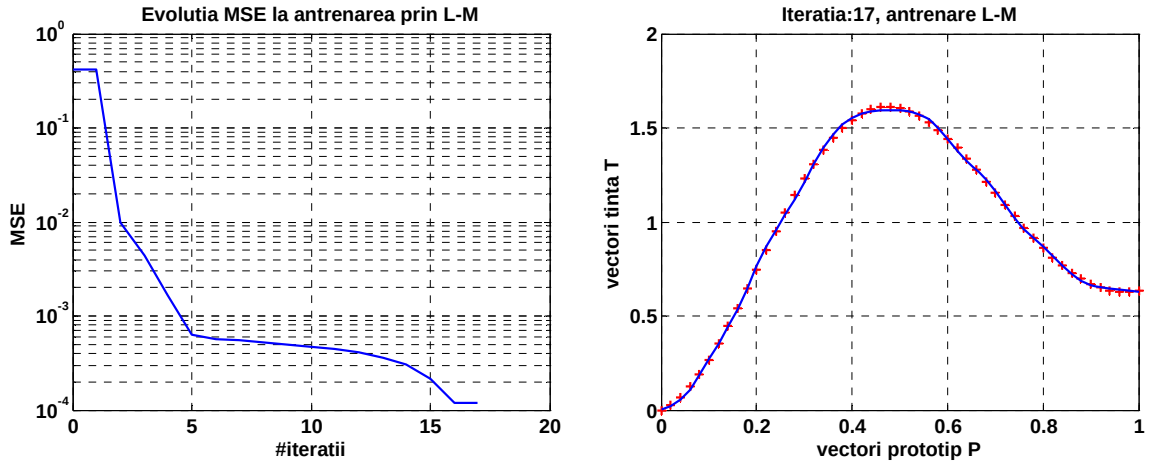
```
while ~((met == 1) || (met == 2)),
        met= input('  1 - gradient, 2 - Levenberg-Marquardt...');
end
if met == 1, % pentru antrenare prin metoda gradientului
        sigm= newff(minmax(P),[neur 1],{'tansig' 'purelin'},'traingd');
        sigm.trainParam.lr= input('\nValoarea ratei de invatare: ');
else            % pentru antrenare prin metoda Levenberg-Marquardt
        sigm= newff(minmax(P),[neur 1],{'tansig' 'purelin'},'trainlm');
end

disp('Valorile initiale ale parametrilor:')
disp('     Stratul de intrare:')
sigm.IW{1,1}
sigm.b{1}
disp('     Stratul de iesire:')
sigm.LW{2,1}
sigm.b{2}

iteratii= input('\nNumar de iteratii: ');
sigm.trainParam.epochs= 1;
sigm.trainParam.goal= 2e-4;
sigm.trainParam.show= inf;

figure(1);
hold on;
h= plot(P,sim(sigm,P));
title('Aproximarea initiala');
Err(1)= mse(T-sim(sigm,P));
        %eroarea medie patratica inainte de inceperea antrenarii;
        % vectorul "Err" va contine evolutia erorii

%  antrenarea retelei
echo off
for i= 1:iteratii,
        [sigm, tr]= train(sigm,P,T);
        Err(i+1)= tr.perf(1);
        pause(0.1);
        delete(h);
        title(strcat('Iteratia:', num2str(i)));
        h= plot(P, sim(sigm,P));
        if Err(i)<= sigm.trainParam.goal
                break;
        end
end

%evolutia erorii
disp('Apasati o tasta pentru a vedea evolutia erorii functie de numarul de epoci...');
pause;

iteratii= i;    %numarul de epoci cat a durat antenarea
fprintf('Eroarea medie patratica obtinuta este %g\n\n', Err(i));
```

```
fprintf('\n\nAntrenarea a durat %g iteratii\n', iteratii);

disp('Valorile finale ale parametrilor:')
disp('    Stratul de intrare:')
disp('    Ponderi:')
sigm.IW{1,1}
disp('    Deplasari:')
sigm.b{1}

disp('    Stratul de iesire:')
disp('    Ponderi:')
sigm.LW{2,1}
disp('    Deplasari:')
sigm.b{2}

figure;
semilogy((0:iteratii), Err);
xlabel('iteratii');
ylabel('eroarea');
title('Eroarea medie patratica functie de numarul de iteratii');

disp('Apasati o tasta pentru a vizualiza capacitatea de generalizare a retelei
antrenate...');
pause;

% verificare capacitate de generalizare a retelei antrenate
Pg1 = -0.1:.02:0;
Tg1 =1+exp(-Pg1).*sin(2*pi*Pg1-pi/2);
Pg2 = 1:.02:1.1;
Tg2 =1+exp(-Pg2).*sin(2*pi*Pg2-pi/2);

figure(1); hold on
plot(Pg1, Tg1, 'r*'); plot(Pg2, Tg2, 'r*');
Yg1= sim(sigm,Pg1); Yg2= sim(sigm,Pg2);
plot(Pg1, Yg1, 'g-');  plot(Pg2, Yg2, 'g-');
hold off;
```

Programul prezentat mai sus poate fi folosit pentru a efectua analiza cerută.

De exemplu, pentru o rețea cu 7 neuroni pe stratul de intrare, impunând 2e-4 ca valoare de prag pentru eroarea pătratică medie, antrenarea prin metoda Levenberg-Marquardt durează 17 epoci, ajungându-se la eroarea pătratică medie 0.000117845.

Figura 8.14.1.(a) ilustrează evoluția erorii medii pătratice MSE pe parcursul a 17 de epoci de antrenare utilizând algoritmul Levenberg-Marquardt pentru rețeaua cu 7 neuroni pe stratul de intrare. Aproximarea realizată de rețeaua antrenată poate fi vizualizată în figura 8.14.1.(b).



(a) (b)

Figura 8.14.1. (a) *Evoluţia erorii medii pătratice MSE pe parcursul a 17 de epoci de antrenare utilizând algoritmul Levenberg-Marquardt pentru o reţea cu 7 neuroni de intrare.* (b) *Aproximarea realizată de reţeaua antrenată.*

Problema 8.15.

Să se reia Problema 3 pentru aceiaşi vectori ţintă şi vectorii prototip:

$$z_i = \varphi(x_i) = 1 + \exp(-x_i)\sin(8\pi x_i - \frac{\pi}{2}), \quad i = 1,...,51 .$$

Indicaţii şi răspunsuri:

În programul ***arna_bp2*** se modifică în mod corespunzător matricea vectorilor ţintă.

```
% definirea tintelor T
T = 1 + exp(-P).*sin(8*pi*P - pi/2);
```

Problema 8.16.

Se consideră imaginea RGB din figura 8.16.1 reprezentând un preparat microscopic dintr-un aliaj de cupru care conţine trei elemente diferite, şi anume: cupru (cromaticitate maron închis), crom (cromaticitate maron deschis) şi plumb (cromaticitate albastră).

Să se determine procentele din aria totală a imaginii pe care le ocupă fiecare din cele trei elemente.

Indicaţii şi răspunsuri:

Datorită complexităţii problemei, soluţia acesteia este prezentată în continuare foarte sumar. Pentru elaborarea programelor MATLAB, în afară de funcţiile din pachetul Neural Networks Toolbox sunt necesare funcţii specifice prelucrării de imagini, din Image Processing Toolbox.

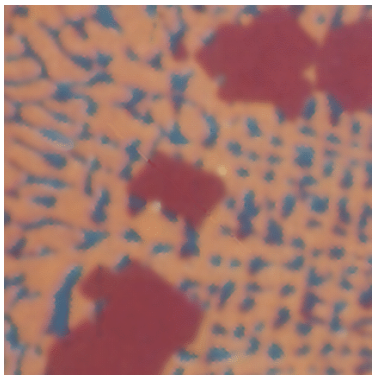


Figura 8.16.1. *Imaginea color a preparatului microscopic din aliaj de cupru.*

Pentru clasificarea pixelilor imaginii considerate se ţine cont de faptul că fiecare pixel p al imaginii color este caracterizat prin:

(i) un vector cu două componente, notat $g(p)$, care exprimă coordonatele geometrice ale pixelului;

(ii) un vector cu două componente, $c(p)$, care exprimă coordonatele de culoare din cubul RGB.

Folosind coordonatele geometrice, să definim mulţimea pixelilor G_k care corespund regiunilor ocupate de elementul e_k, $k = 1,2,3$, unde e_1 semnifică cuprul, e_2 semnifică cromul şi e_3 semnifică plumbul. Evident, mulţimile G_k sunt disjuncte.

În cubul RGB (scalat $[0,1]\times[0,1]\times[0,1]$) definim mulţimea de culori asociată fiecărui element e_k, $k = 1,2,3$.

$$C_k = \left\{ \bigcup_p c(p) \mid g(p) \in G_k \right\}. \tag{8.16.1}$$

Ipoteză de lucru: Cel puţin din punct de vedere teoretic, se poate considera că mulţimile C_1, C_2, C_3 sunt distincte. Satisfacerea practică a acestei ipoteze depinde de calitatea imaginii color.

Algoritm

Pasul 1. Se selectează eşantioane de culoare reprezentative pentru cele trei tipuri regiuni şi se definesc mulţimile de eşantioane $\tilde{C}_1, \tilde{C}_2, \tilde{C}_3$. (Aceasta constituie cea mai bună posibilitate de a obţine informaţii parţiale despre mulţimile de culori C_1, C_2, C_3 care sunt necunoscute ca valori concrete ocupate în cubul RGB. Se elimină vectorii identici din fiecare mulţime de culoare. În baza ipotezei de lucru considerate, $\tilde{C}_k \subseteq C_k$, $k = 1,2,3$ şi, drept urmare mulţimile de eşantioane $\tilde{C}_1, \tilde{C}_2, \tilde{C}_3$ sunt disjuncte.)

Figura 8.16.2 ilustrează o modalitate de alegere a mulțimilor de eşantioane $\tilde{C}_1, \tilde{C}_2, \tilde{C}_3$. Figura 8.16.3 prezintă plasarea culorilor din $\tilde{C}_1, \tilde{C}_2, \tilde{C}_3$ în cubul RGB. Pentru reprezentarea grafică se utilizează convenția: $\tilde{C}_1$ - galben, $\tilde{C}_2$ - roşu, $\tilde{C}_3$ - bleu.

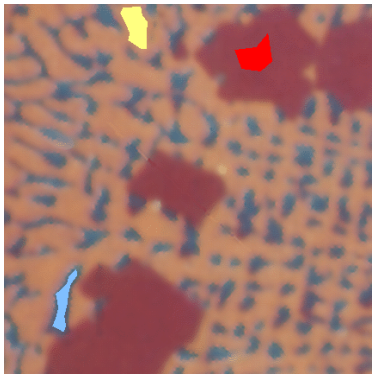


Figura 8.16.2. *Alegerea eşantioanelor de culoare din regiuni reprezentative ale imaginii.*

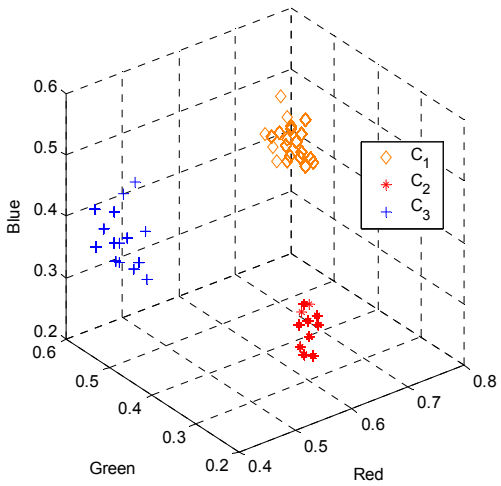


Figura 8.16.3. *Plasarea culorilor din mulțimile de eşantioane* $\tilde{C}_1, \tilde{C}_2, \tilde{C}_3$ *în cubul RGB scalat* $[0,1]\times[0,1]\times[0,1]$.

Pasul 2. Se antrenează o rețea neurală cu arhitectură adecvată astfel încât să clasifice în 3 clase culorile din mulțimile de eşantione $\tilde{C}_1, \tilde{C}_2, \tilde{C}_3$.

Pasul 3. Se utilizează rețeaua antrenată pentru a clasifica toți pixelii din imagine în cele 4 clase, corespunzătoare lui $\tilde{C}_1, \tilde{C}_2, \tilde{C}_3$.

Pasul 4. Se construieşte o nouă imagine digitală în care se definesc următoarele 3 regiuni

$$\hat{G}_k = \{\bigcup_p g(p) \,|\, c(p) \in \text{class } k\}, \quad k = 1,2,3, \tag{2}$$

care estimează regiunile ideale de interes G_k. Se utilizează 3 culori distincte pentru a reprezenta regiunile estimate $\tilde{G}_1, \tilde{G}_2, \tilde{G}_3$ (strategie care asigură o bună vizualizare). În limbajul specific procesării imaginilor, operaţia prin care se obţin regiunile $\tilde{G}_1, \tilde{G}_2, \tilde{G}_3$ ca estimări ale regiunile ideale G_1, G_2, G_3 .poartă denumirea de "segmentare".

Figura 8.16.4 prezintă clasificarea realizată de o reţea cu 8 neuroni în primul strat, antrenată în Pasul 3 prin algoritmul Levenberg-Marquardt până la o eroare pătratică medie 4.87e-010 (obţinută după 16 iteraţii).

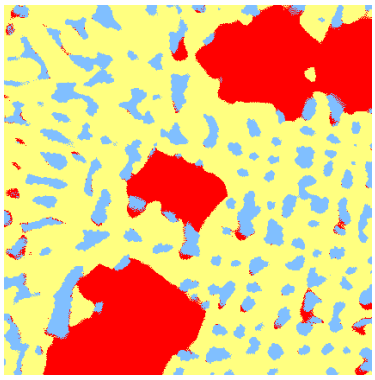


Figura 8.16.4. *Separarea în clase (regiuni de interes realizată de o reţea cu 8 neuroni în primul strat şi antrenată prin algoritmul Levenberg-Marquardt.*

Pasul 5. Se măsoară ariile regiunilor estimate $\tilde{G}_1$, $\tilde{G}_2$, $\tilde{G}_3$ şi se calculează procentele ocupate de fiecare.

Pentru segmentarea prezentată în figura 8.16.4 se obţin următoarele procente: 59.27 % pentru $\tilde{G}_1$ (galben), 23.55 % pentru $\tilde{G}_2$ (roşu) şi 17.18 % pentru $\tilde{G}_3$ (bleu). Ajungem astfel la concluzia că aliajul considerat conţine 59.27 % crom, 23.55 % cupru şi 17.18 % plumb.

Problema 8.17.

Pe baza exemplului prezentat în secţiunea 6.4.1.A să se construiască modelul Simulink pentru identificarea sistemului liniar descris de funcţia de transfer

$$G(s) = \frac{125e^{-3s}}{100s^2 + 40s + 3},$$

considerând perioada de eşantionare a semnalelor de 1 secundă.

Să se ruleze experimentele de identificare off-line şi on-line în următoarele condiţii:

a. ieşirea procesului nu este afectată de perturbaţii;

b. ieşirea procesului este afectată aditiv de perturbaţii de tip Gaussian care pot atinge 1% din amplitudinea maximă a semnalului util (corespunzând, de exemplu, zgomotului de conversie al unui convertor A/D cu rezoluţie scăzută).

Corespunzător ecuaţiei (6.1.1), se vor lua în discuţie următoarele cazuri:

(*i*) $d = 4$, $n = 2$, $m = 1$;
(*ii*) $d = 4$, $n = 2$, $m = 2$;
(*iii*) $d = 4$, $n = 3$, $m = 2$;
(*iv*) $d = 4$, $n = 3$, $m = 3$;

Pentru fiecare din cazurile (*i*)-(*iv*) se va fixa valoarea de prag a erorii pătratice medii utilizată în antrenarea reţelei (*mean squared error goal*) la valorile 10^{-2} şi apoi 0; în fiecare caz în parte se va comenta legătura dintre parametrii reţelei neurale şi coeficienţii funcţiei de transfer discrete corespunzătoare sistemului, obţinută prin apelarea funcţiei Matlab c2d.

După rularea unui experiment de identificare, se va rula, pentru validarea modelului, un experimentul de simulare, considerând ca semnal de intrare în sistem şi în modelul neural obţinut atât un semnal de tip zgomot alb, cât şi un semnal determinist, de tip treaptă unitară sau sinusoidal.

Indicaţii şi răspunsuri:

Se utilizează modelele Simulink prezentate în figurile 8.17.1 şi 8.17.2.

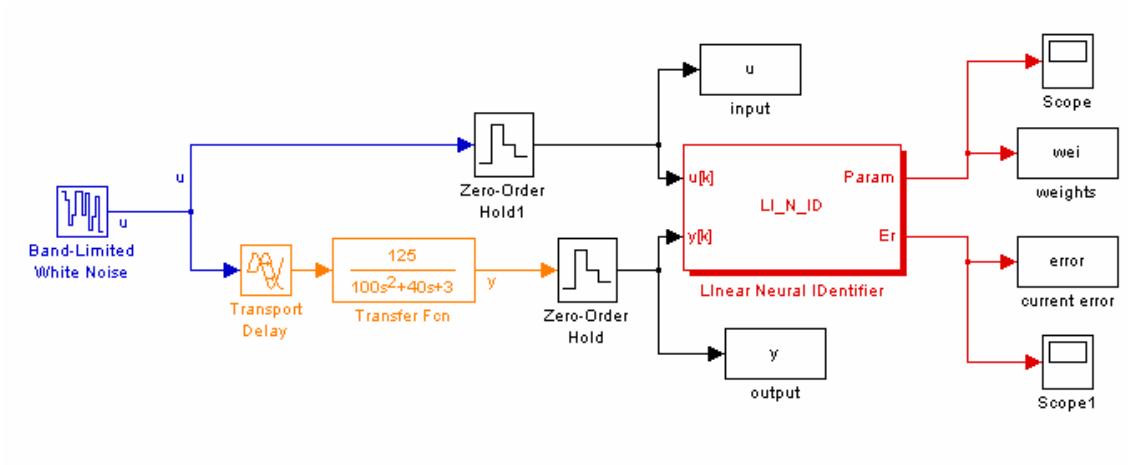


Figura 8.17.1. *Exemplu de construire a modelului Simulink pentru identificarea sistemului dinamic din problema 8.17.*

Pentru discretizarea funcţiei de transfer date se utilizează secvenţa de comenzi MATLAB de mai jos.

```
% functia de transfer in timp continuu
s= tf('s');
Gc= 125 / (100*s*s + 40*s + 3);
Gc.ioDelay= 3

% discretizare cu perioada de esantionare T=1
Gd= c2d(Gc, 1, 'zoh')
```

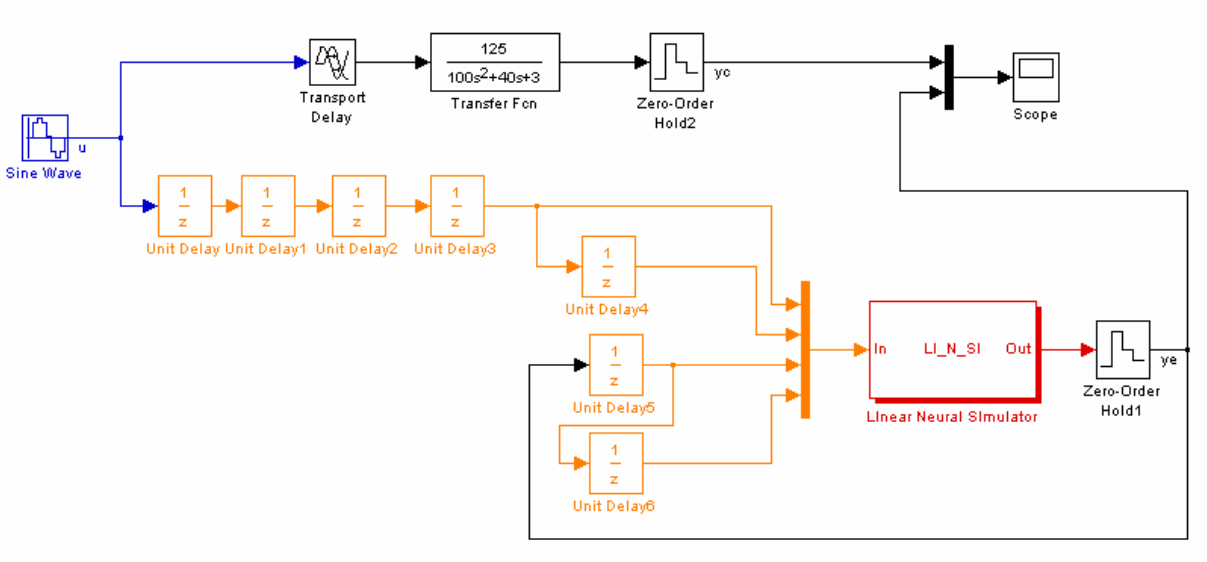


Figura 8.17.2. *Exemplu de construire a modelului Simulink pentru simularea sistem dinamic utilizând modelul neural identificat.*

Se obține astfel

$$G_d(z) = z^{-3}\frac{0.54z+0.47}{z^2-1,64z+0.67} = z^{-4}\frac{0.54+0.47z^{-1}}{1-1,64z^{-1}+0.67z^{-2}},$$

ceea ce arată că în timp discret sistemul liniar considerat este descris de ecuația cu diferențe:

$$\left(1-1,64z^{-1}+0.67z^{-2}\right)y[k] = z^{-4}\left(0.54+0.47z^{-1}\right)u[k]$$

adică:

$$y[k] = 1,64y[k-1]-0.67y[k-2]+0.54u[k-4]+0.47u[k-5].$$

Parametrii modelului neural obținut prin identificare pot fi comparați cu parametrii modelului exact obținut prin discretizare.

Problema 8.18.

Se consideră sistemul neliniar în timp discret descris de ecuația cu diferențe (pentru o perioadă de eşantionare T=1 secundă):

$$y(k) = \frac{y[k-1]+y[k-2]}{1+2y[k-1]y[k-2]}-1.5u(k-1)+u[k-2].$$

1. Folosind blocurile Simulink prezentate în secțiunea 6.3, să se obțină un model neuronal de tip MLP pentru procesul dat, efectuând experimente de identificare off-line şi on-line în următoarele condiții:

(*a*) ieşirea procesului nu este afectată de perturbații;

(*b*) ieşirea procesului este afectată aditiv de perturbații de tip Gaussian care pot atinge 1% din amplitudinea maximă a semnalului util (corespunzând, de exemplu, zgomotului de conversie al unui convertor A/D cu rezoluție scăzută).

Corespunzător modelului NNARX, se va lua în discuție atât cazul când întârzierile rețelei neuronale sunt egale cu întârzierile reale ale procesului, cât şi cazul introducerii unor regresori suplimentari în modelul neuronal.

2. Calitatea modelelor neuronale obținute va fi apreciată din punct de vedere al analizei comparative a răspunsurilor obținute de la proces şi respectiv model, pentru următoarele cazuri:

(*a*) semnal de intrare aleator cu distribuție uniformă în intervalul [-1, 1];

(*b*) semnal de intrare aleator binar de nivele 1 şi -1.

3. Modelele obținute vor fi folosite pentru a realiza controlul predictiv al procesului considerat în următoarele cazuri:

(*a*) referință sinusoidală, cu amplitudine 1, fază inițială 0 şi pulsație $2\pi/250$ rad/sec.;

(*b*) referință în formă de dinți de ferăstrău de amplitudine 1, offset 0.5 şi perioadă 100 secunde.

Pentru algoritmul de control predictiv, se vor considera următorii parametrii: orizont de control N_u=2, orizont minim de predicție N_1=1, orizont de predicție N_2=3 şi ponderea asupra comenzii λ=0,2.

Indicații şi răspunsuri:

Se utilizează blocurile descrise în secțiunea 6.3 pentru a construi modele Simulink similare celor prezentate în secțiunea 6.4.

Anexe

Anexa A

METODE DE OPTIMIZARE UTILIZATE CA ALGORITMI DE ANTRENARE SUPERVIZATĂ A REȚELELOR NEURALE

A.1. Introducere

Se consideră o funcție $f:\mathbb{R}^n \to \mathbb{R}$ a cărei valoare minimă pe o mulțime $\Omega \subseteq \mathbb{R}$ dorim să o determinăm. Se obține astfel problema de optimizare:

$$\text{minimizarea } f(\boldsymbol{x}) \qquad \text{(A.1.1)}$$
$$\text{cu restricția } \boldsymbol{x} \in \Omega .$$

Funcția f este denumită *funcție de cost* sau *funcție obiectiv*, iar Ω reprezintă *mulțimea valorilor fezabile* ale vectorului de decizie $\boldsymbol{x} = [x_1, x_2, ..., x_n]^T \in \mathbb{R}^n$.

În cazul în care $\Omega \subset \mathbb{R}^n$, (A.1.1) reprezintă forma generală a unei probleme de optimizare cu restricții. Dacă $\Omega = \mathbb{R}^n$ se obține o problemă de optimizare fără restricții.

În multe situații concrete, mulțimea restricțiilor este precizată sub forma:

$$\Omega = \left\{ \boldsymbol{x} \in \mathbb{R}^n \,\middle|\, \boldsymbol{h}(\boldsymbol{x}) = 0, \boldsymbol{g}(\boldsymbol{x}) \le 0 \right\},$$

unde funcțiile $\boldsymbol{h}:\mathbb{R}^n \to \mathbb{R}^m$ şi $\boldsymbol{g}:\mathbb{R}^n \to \mathbb{R}^p$ sunt cunoscute.

Definiția A.1.1. Fie $f:\mathbb{R}^n \to \mathbb{R}$ şi $\Omega \subset \mathbb{R}^n$.

Un punct $\boldsymbol{x}^* \in \Omega$ este *punct de minim local* pentru f peste Ω dacă $\exists \varepsilon > 0$ astfel încât $f(\boldsymbol{x}) \ge f(\boldsymbol{x}^*)$, pentru orice $\boldsymbol{x} \in \Omega \setminus \{\boldsymbol{x}^*\}$ satisfăcând $\| \boldsymbol{x} - \boldsymbol{x}^* \| < \varepsilon$.

Un punct $\boldsymbol{x}^* \in \Omega$ este *punct de minim global* pentru f peste Ω dacă $f(\boldsymbol{x}) \geq f(\boldsymbol{x}^*)$ pentru orice $\boldsymbol{x} \in \Omega \setminus \{\boldsymbol{x}^*\}$. ■

Punctul de minim $\boldsymbol{x}^*$ reprezintă argumentul care minimizează funcția f (pe Ω sau pe întreg spațiu $\mathbb{R}^n$) și se notează $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}\in\Omega} f(\boldsymbol{x})$ sau $\boldsymbol{x}^* = \arg\min f(\boldsymbol{x})$.

Dacă în definiția A.1 se înlocuiește "$\geq$" cu ">" atunci punctul $\boldsymbol{x}^*$ reprezintă un *punct de minim (local sau global) strict.*

Figura A.1.1 prezintă graficul unei funcții $f:\mathbb{R}\to\mathbb{R}$. Punctele x_1^* și x_2^* sunt puncte de minim global, respectiv local, strict; iar x_3^* este punct minim local nestrict.

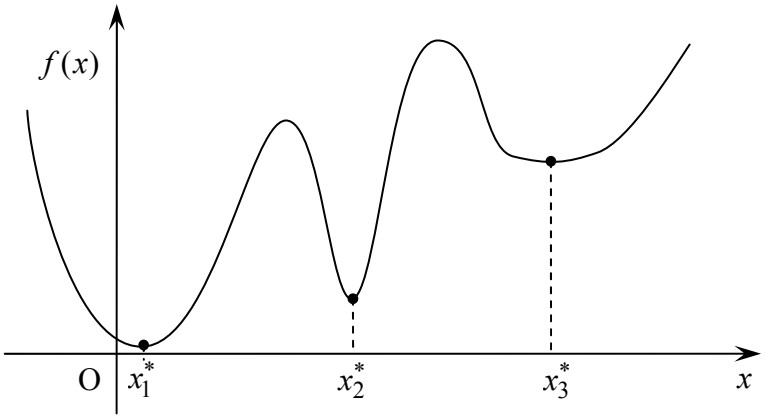


Figura A.1.1. *Puncte de minim ale unei funcții reale.*

A.1.1. Condiții de minim local

Pentru prezentarea unor condiții necesare (și suficiente) de minim, este nevoie de câteva cunoștințe preliminare.

Definiția A.1.2. Se consideră funcția $f:\mathbb{R}^n\to\mathbb{R}$, de două ori derivabilă pe $\mathbb{R}^n$.
Gradientul funcției scalare f într-un punct $\boldsymbol{x}$ este vectorul coloană format cu derivatele parțiale de ordinul 1 ale lui f,

$$\nabla f(\boldsymbol{x}) = \left[\frac{\partial f}{\partial x_1}(\boldsymbol{x}) \quad \frac{\partial f}{\partial x_2}(\boldsymbol{x}) \quad \cdots \quad \frac{\partial f}{\partial x_n}(\boldsymbol{x})\right]^T \tag{A.1.2}$$

iar matricea Hessian a lui f este formată cu derivatele parțiale de ordinul 2 ale lui f:

$$\boldsymbol{H}_f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^{\,2}}(\boldsymbol{x}) & \frac{\partial^2 f}{\partial x_1 \partial x_2}(\boldsymbol{x}) & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n}(\boldsymbol{x}) \\ \frac{\partial^2 f}{\partial x_2 \partial x_1}(\boldsymbol{x}) & \frac{\partial^2 f}{\partial x_2^{\,2}}(\boldsymbol{x}) & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n}(\boldsymbol{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1}(\boldsymbol{x}) & \frac{\partial^2 f}{\partial x_n \partial x_2}(\boldsymbol{x}) & \cdots & \frac{\partial^2 f}{\partial x_n^{\,2}}(\boldsymbol{x}) \end{bmatrix} \left(\triangleq \nabla^2 f(\boldsymbol{x})\right) \quad \blacksquare \text{ (A.1.3)}$$

Dacă funcția f este de clasă $C^2_{\mathbb{R}^n}$ (adică f este de două ori derivabilă şi derivata a doua a sa este continuă pe $\mathbb{R}^n$) atunci, conform *teoremei Clairaut-Schwarz* are loc egalitatea derivatelor parțiale de ordinul 2

$$\frac{\partial^2 f}{\partial x_i \partial x_j}(\boldsymbol{x}) = \frac{\partial^2 f}{\partial x_j \partial x_i}(\boldsymbol{x}),\ \forall \boldsymbol{x} \in \mathbb{R}^n,\ i, j = \overline{1,n},$$

astfel că matricea Hessian este simetrică.

Exemplul A.1.1.

a) *O funcție liniară* pe $\mathbb{R}^n$ este definită printr-o relație de forma $f(\boldsymbol{x}) = \boldsymbol{d}^T \boldsymbol{x}$, unde $\boldsymbol{d} = [d_1\ d_2\ \ldots\ d_n]^T \in \mathbb{R}^n$. Gradientul acestei funcții este $\nabla f(\boldsymbol{x}) = \boldsymbol{d}$.

b) Considerăm funcția definită prin $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T \boldsymbol{A}\,\boldsymbol{x} + \boldsymbol{b}^T \boldsymbol{x} + c$, unde $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, $\boldsymbol{b} \in \mathbb{R}^n$ şi $c \in \mathbb{R}$. Dacă matricea $\boldsymbol{A}$ este simetrică, funcția f se numeşte *formă pătratică* şi $\nabla f(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$ iar $\boldsymbol{H}_f(\boldsymbol{x}) = \boldsymbol{A}$. Dacă matricea $\boldsymbol{A}$ nu este simetrică, atunci $\nabla f(\boldsymbol{x}) = \frac{1}{2}(\boldsymbol{A} + \boldsymbol{A}^T)\boldsymbol{x} + \boldsymbol{b}$ şi $\boldsymbol{H}_f(\boldsymbol{x}) = \frac{1}{2}(\boldsymbol{A} + \boldsymbol{A}^T)$. ■

Definiția A.1.3. Un vector $\boldsymbol{d} \in \mathbb{R}^n$, $\boldsymbol{d} \neq 0$ reprezintă o *direcție fezabilă* în punctul $\boldsymbol{x} \in \Omega$ dacă există $\alpha_0 > 0$ astfel încât $\boldsymbol{x} + \alpha\,\boldsymbol{d} \in \Omega$, pentru orice $\alpha \in [0, \alpha_0]$. ■

În figura A.1.2 direcția $\boldsymbol{d}_1$ este fezabilă în punctul $\boldsymbol{x}$, în timp ce $\boldsymbol{d}_2$ nu este fezabilă. Se observă că dacă $\boldsymbol{x}$ este punct interior mulțimii Ω atunci orice direcție este fezabilă în $\boldsymbol{x}$.

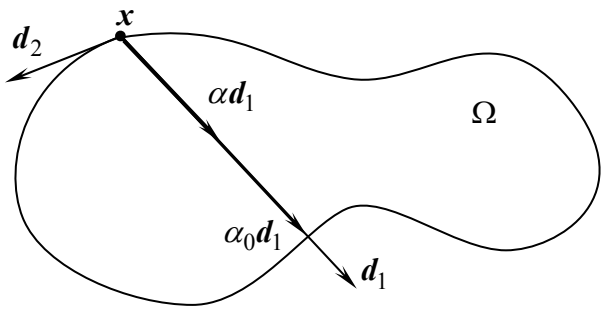


Figura A.1.2. *Ilustrarea grafică a conceptului de direcție fezabilă într-un punct* $\boldsymbol{x}$.

Definiţia A.1.4. Fie $f:\mathbb{R}^n \to \mathbb{R}$ şi $\boldsymbol{d} \in \mathbb{R}^n$ o direcţie fezabilă în punctul $\boldsymbol{x} \in \Omega$. *Derivata direcţională* a lui f în direcţia $\boldsymbol{d}$, notată $\frac{\partial f}{\partial \boldsymbol{d}}$, este definită prin:

$$\frac{\partial f}{\partial \boldsymbol{d}}(\boldsymbol{x}) = \lim_{\alpha \to 0} \frac{f(\boldsymbol{x}+\alpha \boldsymbol{d}) - f(\boldsymbol{x})}{\alpha} \qquad \blacksquare \text{ (A.1.4)}$$

Dacă vectorul $\boldsymbol{d}$ este unitar, adică $\| \boldsymbol{d} \| = 1$, atunci $\partial f / \partial \boldsymbol{d}$ reprezintă rata de creştere a lui f în direcţia $\boldsymbol{d}$ şi poate fi calculată considerând $\boldsymbol{f}(\boldsymbol{x}+\alpha \boldsymbol{d})$ ca funcţie de α:

$$\frac{\partial f}{\partial \boldsymbol{d}}(\boldsymbol{x}) = \frac{d}{d\alpha} f(\boldsymbol{x}+\alpha \boldsymbol{d})|_{\alpha=0} = [\nabla f(\boldsymbol{x})]^T \cdot \boldsymbol{d} = \boldsymbol{d}^T \cdot \nabla \boldsymbol{f}(\boldsymbol{x}). \qquad \text{(A.1.4')}$$

Următoarele propoziţii precizează condiţii necesare (şi suficiente) de minim.

Teorema A.1.1. Fie $\Omega \subset \mathbb{R}^n$ şi $f:\mathbb{R}^n \to \mathbb{R}$, $f \in C^1_{\mathbb{R}^n}$. Dacă $\boldsymbol{x}^*$ este punct de minim local al lui f peste Ω, atunci pentru orice direcţie $\boldsymbol{d}$ fezabilă în $\boldsymbol{x}^*$ este satisfăcută condiţia

$$\boldsymbol{d}^T \cdot \nabla f(\boldsymbol{x}^*) \geq 0. \qquad \blacksquare \text{ (A.1.5)}$$

Corolar A.1.1. Fie $\Omega \subset \mathbb{R}^n$ şi $f:\mathbb{R}^n \to \mathbb{R}$, $f \in C^1_{\mathbb{R}^n}$. Dacă $\boldsymbol{x}^*$ este punct de minim local al lui f şi x^* este punct interior mulţimii Ω atunci:

$$\nabla f(\boldsymbol{x}^*) = 0. \qquad \blacksquare \text{ (A.1.6)}$$

Teorema A.1.2. Fie $\Omega \subset \mathbb{R}^n$ şi $f:\mathbb{R}^n \to \mathbb{R}$, $f \in C^2_{\mathbb{R}^n}$, $\boldsymbol{x}^*$ un punct de minim local al lui f peste Ω şi $\boldsymbol{d}$ o direcţie fezabilă în $\boldsymbol{x}^*$. Dacă $\boldsymbol{d}^T \cdot \nabla f(\boldsymbol{x}^*) = 0$, atunci

$$\boldsymbol{d}^T \boldsymbol{H}_f(\boldsymbol{x}^*) \boldsymbol{d} \geq 0, \qquad \text{(A.1.7)}$$

unde $\boldsymbol{H}_f(\boldsymbol{x}^*)$ reprezintă matricea Hessian a lui f calculată în $\boldsymbol{x}^*$. $\blacksquare$

Corolar A.1.2. Fie $\boldsymbol{x}^*$ punct interior mulţimii $\Omega \subset \mathbb{R}^n$ şi $f:\mathbb{R}^n \to \mathbb{R}$, $f \in C^2_{\mathbb{R}^n}$. Dacă $\boldsymbol{x}^*$ este punct de minim local al lui f atunci:

$$\nabla f(\boldsymbol{x}^*) = 0 \qquad \text{(A.1.8)}$$

şi matricea Hessian $\boldsymbol{H}_f(\boldsymbol{x}^*)$ este pozitiv semidefinită ($\boldsymbol{H}_f(\boldsymbol{x}^*) \geq 0$), adică

$$\boldsymbol{d}^T \boldsymbol{H}_f(\boldsymbol{x}^*) \boldsymbol{d} \geq 0, \ \forall \boldsymbol{d} \in \mathbb{R}^n \setminus \{0\}. \qquad \blacksquare \text{ (A.1.9)}$$

Condiţiile necesare prezentate în Corolarul A.1.2 nu sunt însă şi suficiente, aşa cum se poate observa în cazul funcţiei $f_1:\mathbb{R} \to \mathbb{R}$, $f_1(x) = x^3$, pentru punctul $x^* = 0$ (figura A.1.3.a) care satisface $f_1'(0) = f_1''(0) = 0$. O situaţie similară este cea a funcţiei $f_2:\mathbb{R}^2 \to \mathbb{R}$, $f_2(\boldsymbol{x}) = x_1^2 - x_2^2$, în punctul $\boldsymbol{x}^* = [0 \ \ 0]^T$ (figura A.1.3.b) care reprezintă un *punct în şa*.

Următoarea teoremă prezintă condiţii suficiente de minim local.

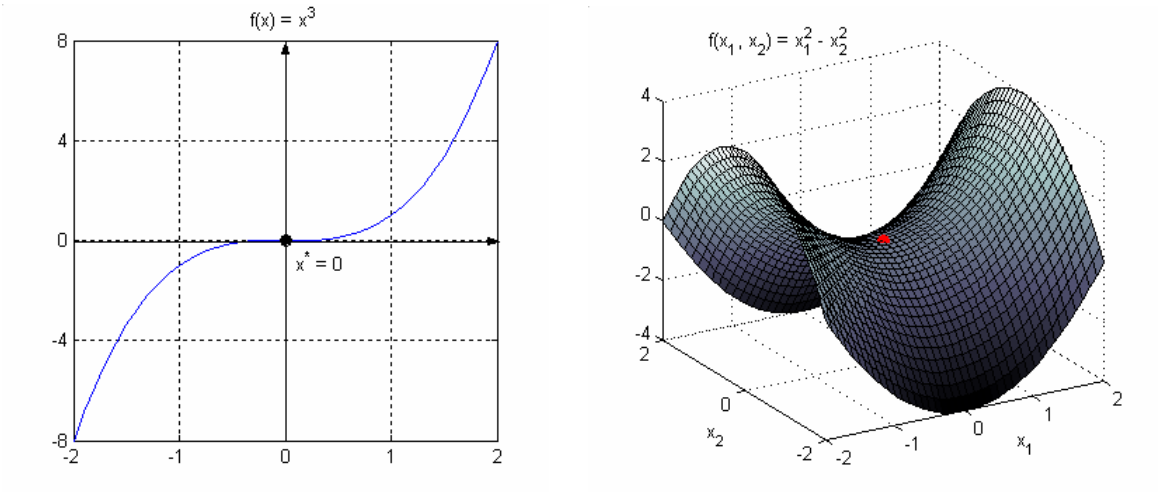
Teorema A.1.3. Fie $f:\mathbb{R}^n \to \mathbb{R}$, $f \in C^2_{\mathbb{R}^n}$, şi $\boldsymbol{x}^*$ un punct interior mulțimii $\Omega \subset \mathbb{R}^n$. Dacă

$$\nabla f(\boldsymbol{x}^*) = 0, \tag{A.1.10}$$

şi matricea Hessian în punctul $\boldsymbol{x}^*$ este pozitiv definită ($\boldsymbol{H}_f(\boldsymbol{x}^*) > 0$), adică

$$\boldsymbol{d}^T \boldsymbol{H}_f(\boldsymbol{x}^*)\boldsymbol{d} > 0, \ \forall \boldsymbol{d} \in \mathbb{R}^n \setminus \{0\}, \tag{A.1.11}$$

atunci $\boldsymbol{x}^*$ este punct de minim local strict al funcției f. ■



(a) (b)

Figura A.1.3. *Ilustrarea grafică a conceptului de punct în şa: (a) pentru funcția* $f(x) = x^3$ *în punctul* $x^* = 0$, *(b) pentru funcția* $f(x_1, x_2) = x_1^2 - x_2^2$ *în punctul* $\boldsymbol{x}^* = [0 \ \ 0]^T$.

Exemplul A.1.2. Considerăm funcția $f:\mathbb{R}^2 \to \mathbb{R}$, $f(\boldsymbol{x}) = x_1^2 + x_2^2$, a cărei reprezentare grafică este prezentată în figura A.1.4.

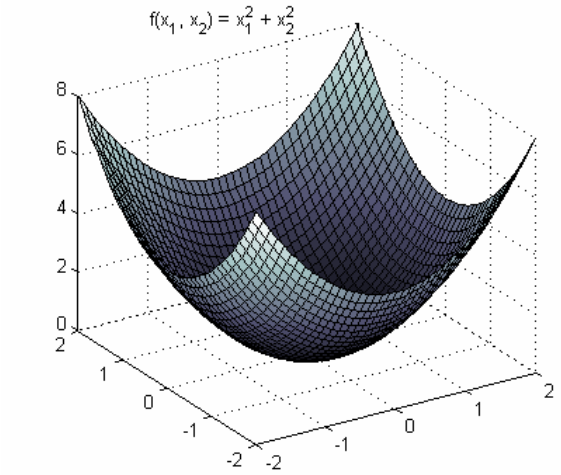


Figura A.1.4. *Reprezentarea grafică a funcției* $f(x_1, x_2) = x_1^2 + x_2^2$.

Gradientul lui *f* într-un punct oarecare este $\nabla f(\boldsymbol{x}) = \begin{bmatrix} 2x_1 & 2x_2 \end{bmatrix}^T$. Condiția necesară de minim (A.1.10) conduce la $\nabla f(\boldsymbol{x}^*) = 0$ dacă şi numai dacă $\boldsymbol{x}^* = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$. Matricea Hessian este constantă pe $\mathbb{R}^2$, $\boldsymbol{H}_f = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$, simetrică şi cu valori proprii strict pozitive, $\lambda_1 = \lambda_2 = 2 > 0$, deci este pozitiv definită. Aplicând Teorema A.3 rezultă că $x^* = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$ este punct de minim local al funcției *f* pe orice mulțime Ω ce conține $\boldsymbol{x}^*$ în interior. În plus $\boldsymbol{x}^*$ este punct de minim global al lui *f* pe $\mathbb{R}^2$. ■

Exemplul A.1.3. O generalizare a cazului din exemplul precedent este reprezentată de forma pătratică

$$f : \mathbb{R}^n \to \mathbb{R},\ f(\boldsymbol{x}) = \tfrac{1}{2}\boldsymbol{x}^T \boldsymbol{A}\,\boldsymbol{x} + \boldsymbol{b}^T \boldsymbol{x} + c, \qquad \text{(A.1.12)}$$

unde $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ este matrice simetrică, $\boldsymbol{A}^T = \boldsymbol{A}$, $\boldsymbol{b} \in \mathbb{R}^n$ şi $c \in \mathbb{R}$. Pentru această funcție, $\nabla f(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$ iar $\boldsymbol{H}_f(\boldsymbol{x}) = \boldsymbol{A}$.

Dacă matricea $\boldsymbol{A}$ este pozitiv definită, atunci funcția *f* admite un unic punct de minim global pe $\mathbb{R}^n$, care se determină prin rezolvarea ecuației $\nabla f(\boldsymbol{x}) = 0$, ceea ce conduce la $\boldsymbol{x}^* = -\boldsymbol{A}^{-1}\boldsymbol{b}$. Dacă matricea $\boldsymbol{A}$ este negativ definită (adică $-\boldsymbol{A}$ este pozitiv definită), atunci funcția *f* admite un unic punct de maxim global pe $\mathbb{R}^n$, care satisface $\nabla f(\boldsymbol{x}) = 0$, adică $\boldsymbol{x}^* = -\boldsymbol{A}^{-1}\boldsymbol{b}$. În cazul în care nici $\boldsymbol{A}$ nici $-\boldsymbol{A}$ nu sunt pozitiv definite nu se poate spune nimic despre punctele de extrem ale funcției *f*. ■

A.1.2. Rezolvarea ecuației $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$

În numeroase cazuri problema determinării parametrilor necunoscuți ai unui model revine la rezolvarea unei ecuații liniare în vectorul $\boldsymbol{x} \in \mathbb{R}^n$, de forma

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \qquad \text{(A.1.13)}$$

unde $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ şi $\boldsymbol{b} \in \mathbb{R}^m$. În funcție de dimensiunile şi rangul matricei sistemului sunt posibile următoarele situații:

a) Dacă $m = n$ şi matricea $\boldsymbol{A}$ este nesingulară, adică $\det \boldsymbol{A} \neq 0$, atunci ecuația (A.1.13) admite soluția unică $\boldsymbol{x}^* = \boldsymbol{A}^{-1}\boldsymbol{b}$.

b) Consideram cazul $m \geq n$ şi $\operatorname{rang} \boldsymbol{A} = n$. Dacă $\operatorname{rang} \boldsymbol{A} < \operatorname{rang}\begin{bmatrix} \boldsymbol{A} & \boldsymbol{b} \end{bmatrix}$ (se spune că matricea $\boldsymbol{A}$ este de rang complet pe coloane), atunci ecuația (A.1.13) nu admite soluții exacte.

În această situație, se caută o soluție $\boldsymbol{x}^*$ care să minimizeze norma euclidiană a erorii $\boldsymbol{e} = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$. Vectorul $\boldsymbol{x}^*$ care satisface condiția $\| \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \|_2^2 \geq \| \boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{b} \|_2^2$, pentru orice $\boldsymbol{x} \in \mathbb{R}^n$, notat $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}\in\mathbb{R}^n} \| \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \|_2^2$, poartă denumirea de *soluție în sensul celor mai mici pătrate* a ecuației (A.1.13).

Pentru a determina această soluție considerăm funcția

$$f:\mathbb{R}^n\to\mathbb{R},\ f(\boldsymbol{x})=\|\boldsymbol{Ax}-\boldsymbol{b}\|_2^2=(\boldsymbol{Ax}-\boldsymbol{b})^T\cdot(\boldsymbol{Ax}-\boldsymbol{b})=\boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{Ax}-2\boldsymbol{b}^T\boldsymbol{Ax}+\boldsymbol{b}^T\boldsymbol{b}, \quad \text{(A.1.14)}$$

care este pătratică, de forma (A.1.12), şi pentru care calculăm

$$\nabla f(\boldsymbol{x})=2\boldsymbol{A}^T\boldsymbol{Ax}-2\boldsymbol{A}^T\boldsymbol{b}$$

$$\boldsymbol{H}_f(\boldsymbol{x})=2\boldsymbol{A}^T\boldsymbol{A}$$

Din condiția $\operatorname{rang}\boldsymbol{A}=n$ rezultă că $\operatorname{rang}(\boldsymbol{A}^T\boldsymbol{A})=n$, astfel că ecuația $\nabla f(\boldsymbol{x})=0$ are soluție unică

$$\boldsymbol{x}^*=\left(\boldsymbol{A}^T\boldsymbol{A}\right)^{-1}\boldsymbol{A}^T\boldsymbol{b}. \quad \text{(A.1.15)}$$

Se observă că matricea simetrică $\boldsymbol{A}^T\boldsymbol{A}$ este pozitiv definită (deoarece $\boldsymbol{d}^T\boldsymbol{A}^T\boldsymbol{Ad}=\|\boldsymbol{Ad}\|_2^2>0$ pentru orice $\boldsymbol{d}\neq 0$), deci, conform teoremei A.3, punctul $\boldsymbol{x}^*$ (A.1.15) este unicul punct de minim al funcției *f*.

În particular, dacă $m=n$ soluția (A.1.15) devine $\boldsymbol{x}^*=\boldsymbol{A}^{-1}\boldsymbol{b}$.

Valoarea minimă a funcției (A.1.14) este

$$f(\boldsymbol{x}^*)=\left(\boldsymbol{Ax}^*-\boldsymbol{b}\right)^T\cdot\left(\boldsymbol{Ax}^*-\boldsymbol{b}\right)=\left(\boldsymbol{Ax}^*\right)^T\boldsymbol{e}^*-\boldsymbol{b}^T\boldsymbol{e}^*=-\boldsymbol{b}^T\boldsymbol{e}^*,$$

unde $\boldsymbol{e}^*=\boldsymbol{Ax}^*-\boldsymbol{b}$, deoarece $\left(\boldsymbol{Ax}^*\right)^T\boldsymbol{e}^*=\frac{1}{2}\nabla f(\boldsymbol{x}^*)=0$.

c) Considerăm cazul $m\le n$ şi $\operatorname{rang}\boldsymbol{A}=m$ (se spune că matricea $\boldsymbol{A}$ este de rang complet pe linii), ecuația $\boldsymbol{Ax}=\boldsymbol{b}$ admite o infinitate de soluții. În aceasta situație ne interesează determinarea acestei soluții care este mai apropiată de origine, adică minimizează $\|\boldsymbol{x}\|_2$, sau, altfel spus, soluția problemei de optimizare

$$\begin{aligned}&\text{minimizare } \|\boldsymbol{x}\|_2 \quad \text{(A.1.16)}\\ &\text{cu restricția } \boldsymbol{Ax}=\boldsymbol{b}\end{aligned}$$

Soluția unică a acestei probleme este dată de

$$\boldsymbol{x}^*=\boldsymbol{A}^T(\boldsymbol{AA}^T)^{-1}\boldsymbol{b}. \quad \text{(A.1.17)}$$

Pentru a demonstra acest fapt, considerăm o soluție $\boldsymbol{x}\neq\boldsymbol{x}^*$ a ecuației $\boldsymbol{Ax}=\boldsymbol{b}$. Se observă că

$$\|\boldsymbol{x}\|_2^2=\|(\boldsymbol{x}-\boldsymbol{x}^*)+\boldsymbol{x}^*\|_2^2=\left[(\boldsymbol{x}-\boldsymbol{x}^*)+\boldsymbol{x}^*\right]^T\cdot\left[(\boldsymbol{x}-\boldsymbol{x}^*)+\boldsymbol{x}^*\right]=$$

$$=\|\boldsymbol{x}-\boldsymbol{x}^*\|_2^2+\|\boldsymbol{x}^*\|_2^2+2\boldsymbol{x}^{*T}\cdot(\boldsymbol{x}-\boldsymbol{x}^*).$$

Arătăm în continuare că $\boldsymbol{x}^{*T}\cdot(\boldsymbol{x}-\boldsymbol{x}^*)=0$ deoarece:

$$\boldsymbol{x}^{*T}\cdot(\boldsymbol{x}-\boldsymbol{x}^*)=\left[\boldsymbol{A}^T(\boldsymbol{AA}^T)^{-1}\boldsymbol{b}\right]^T\left[\boldsymbol{x}-\boldsymbol{A}^T(\boldsymbol{AA}^T)^{-1}\boldsymbol{b}\right]=\boldsymbol{b}^T(\boldsymbol{AA}^T)^{-1}\boldsymbol{A}\left[\boldsymbol{x}-\boldsymbol{A}^T(\boldsymbol{AA}^T)^{-1}\boldsymbol{b}\right]=$$

$$=\boldsymbol{b}^T(\boldsymbol{AA}^T)^{-1}\left[\boldsymbol{Ax}-(\boldsymbol{AA}^T)(\boldsymbol{AA}^T)^{-1}\boldsymbol{b}\right]=\boldsymbol{b}^T(\boldsymbol{AA}^T)^{-1}[\boldsymbol{b}-\boldsymbol{b}]=0$$

În consecință, pentru orice soluție $\boldsymbol{x}\neq\boldsymbol{x}^*$ a ecuației $\boldsymbol{Ax}=\boldsymbol{b}$, cum $\|\boldsymbol{x}-\boldsymbol{x}^*\|_2\neq 0$, rezultă că $\|\boldsymbol{x}\|_2^2=\|\boldsymbol{x}-\boldsymbol{x}^*\|_2^2+\|\boldsymbol{x}^*\|_2^2>\|\boldsymbol{x}^*\|_2^2$, ceea ce trebuia demonstrat.

În particular, dacă $m = n$ soluția (A.1.17) devine $\boldsymbol{x}^* = \boldsymbol{A}^{-1}\boldsymbol{b}$.

Observație. Fie $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ o matrice dată. Matricea $\boldsymbol{A}^\dagger \in \mathbb{R}^{m\times n}$ se numeşte *pseudo-inversa Morre-Penrose* (inversa generalizată) a matricei $\boldsymbol{A}$ dacă

$$\boldsymbol{A}\,\boldsymbol{A}^\dagger\boldsymbol{A} = \boldsymbol{A}$$

şi există matricele $\boldsymbol{U} \in \mathbb{R}^{n\times n}$, $\boldsymbol{V} \in \mathbb{R}^{m\times m}$, astfel încât

$$\boldsymbol{A}^\dagger = \boldsymbol{U}\boldsymbol{A}^T,\ \boldsymbol{A}^\dagger = \boldsymbol{A}^T\boldsymbol{V}.$$

Relația $\boldsymbol{A}^\dagger = \boldsymbol{U}\boldsymbol{A}^T$ arată că fiecare linie din matricea $\boldsymbol{A}^\dagger$ este o combinație liniară a liniilor din $\boldsymbol{A}^T$ iar $\boldsymbol{A}^\dagger = \boldsymbol{A}^T\boldsymbol{V}$ arată că fiecare coloană din $\boldsymbol{A}^\dagger$ este o combinație liniară a coloanelor din $\boldsymbol{A}^T$.

Se poate demonstra că, dacă există o pseudo-inversă Morre-Penrose $\boldsymbol{A}^\dagger$ a matricei $\boldsymbol{A}$, atunci este unică. Pseudo-inversa unei matrice are următoarele proprietăți:

$$\left(\boldsymbol{A}^T\right)^\dagger = \left(\boldsymbol{A}^\dagger\right)^T,\ \left(\boldsymbol{A}^\dagger\right)^\dagger = \boldsymbol{A},$$

$$\boldsymbol{A}\,\boldsymbol{A}^\dagger\boldsymbol{A} = \boldsymbol{A},\ \boldsymbol{A}^\dagger\boldsymbol{A}\,\boldsymbol{A}^\dagger = \boldsymbol{A}^\dagger,$$

$$\left(\boldsymbol{A}\,\boldsymbol{A}^\dagger\right)^T = \boldsymbol{A}\,\boldsymbol{A}^\dagger,\ \left(\boldsymbol{A}^\dagger\boldsymbol{A}\right)^T = \boldsymbol{A}^\dagger\boldsymbol{A}.$$

În cazul în care $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, cu $m \geq n$ şi $\operatorname{rang}\boldsymbol{A} = n$, atunci matricea $\boldsymbol{A}^\dagger = \left(\boldsymbol{A}^T\boldsymbol{A}\right)^{-1}\boldsymbol{A}^T$ este pseudo-inversa Morre-Penrose a matricei $\boldsymbol{A}$.

Dacă $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, cu $m \leq n$ şi $\operatorname{rang}\boldsymbol{A} = m$, matricea $\boldsymbol{A}^\dagger = \boldsymbol{A}^T\left(\boldsymbol{A}\,\boldsymbol{A}^T\right)^{-1}$ este pseudo-inversa Morre-Penrose a matricei $\boldsymbol{A}$.

Cu aceste observații, soluția (A.1.15) în sensul celor mai mici pătrate obținută în cazul $m \geq n$ cu $\operatorname{rang}\boldsymbol{A} = n$, şi soluția (A.1.17) care minimizează $\|\boldsymbol{x}\|_2$ obținută în cazul $m \leq n$ cu $\operatorname{rang}\boldsymbol{A} = m$, pot fi puse sub forma

$$\boldsymbol{x}^* = \boldsymbol{A}^\dagger\boldsymbol{b}. \qquad \blacksquare \text{ (A.1.18)}$$

A.1.3. Estimatorul celor mai mici pătrate

În numeroase situații, ieşirea unui model, $\boldsymbol{y} \in \mathbb{R}^p$, depinde de intrarea $\boldsymbol{u} \in \mathbb{R}^m$ printr-o relație liniară în parametrii $\boldsymbol{x} = [x_1, x_2, ..., x_n]^T \in \mathbb{R}^n$:

$$\boldsymbol{y} = \boldsymbol{f}_1(\boldsymbol{u})x_1 + \boldsymbol{f}_2(\boldsymbol{u})x_2 + ... + \boldsymbol{f}_n(\boldsymbol{u})x_n, \qquad \text{(A.1.19)}$$

în care funcțiile $\boldsymbol{f}_1, \boldsymbol{f}_2, ..., \boldsymbol{f}_n : \mathbb{R}^m \to \mathbb{R}^p$ sunt cunoscute. Ecuația (A.1.19) poate fi scrisă sub forma matriceală:

$$\boldsymbol{y} = \boldsymbol{M}(\boldsymbol{u})\boldsymbol{x} \qquad \text{(A.1.20)}$$

în care matricea $\boldsymbol{M}(\boldsymbol{u}) \in \mathbb{R}^{p\times n}$ este dată de $\boldsymbol{M}(\boldsymbol{u}) = \left[\boldsymbol{f}_1(\boldsymbol{u})\ \boldsymbol{f}_2(\boldsymbol{u})\ \ldots\ \boldsymbol{f}_n(\boldsymbol{u})\right]$.

Pentru determinarea parametrilor modelului, de regulă se efectuează experimente care să asigure obținerea unui set de perechi de valori intrare-ieşire $(\boldsymbol{u}_i, \boldsymbol{y}_i)$, $i = \overline{1, N}$. Înlocuind

fiecare asemenea pereche în ecuația (A.1.20) se obține:

$$\begin{cases} \boldsymbol{y}_1 = \boldsymbol{M}_1 \boldsymbol{x} \\ \boldsymbol{y}_2 = \boldsymbol{M}_2 \boldsymbol{x} \\ \quad \ldots \\ \boldsymbol{y}_N = \boldsymbol{M}_N \boldsymbol{x} \end{cases}$$

unde $\boldsymbol{M}_i = \boldsymbol{M}(\boldsymbol{u}_i)$, $i = \overline{1, N}$. Cu notațiile

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{M}_1 \\ \boldsymbol{M}_2 \\ \vdots \\ \boldsymbol{M}_N \end{bmatrix} \in \mathbb{R}^{Np \times n}, \ \boldsymbol{b} = \begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \\ \vdots \\ \boldsymbol{y}_N \end{bmatrix} \in \mathbb{R}^{Np},$$

sistemul de ecuații precedent poate fi scris în forma echivalentă

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}. \tag{A.1.21}$$

Astfel, problema determinării parametrilor modelului (A.1.19) revine la rezolvarea unei ecuații matriceale liniare de forma (A.1.13). În general, numărul N de perechi de date pe baza cărora se dorește determinarea parametrilor $\boldsymbol{x}$ este mare, astfel încât matricea $\boldsymbol{A}$ satisface condiția $\operatorname{rang} \boldsymbol{A} = n$ şi se ajunge la cazul prezentat în secțiunea A.1.2.b.

Vectorul

$$\boldsymbol{x}^* = \left(\boldsymbol{A}^T \boldsymbol{A}\right)^{-1} \boldsymbol{A}^T \boldsymbol{b},$$

de forma (A.1.15), care asigură minimizarea erorii pătratice

$$f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 = \frac{1}{2}\sum_{i=1}^{N}\|\boldsymbol{M}_i\boldsymbol{x} - \boldsymbol{y}_i\|_2^2 \tag{A.1.22}$$

poarta denumirea de *estimator al celor mai mici pătrate* (eng. *Least-Squares Estimator – LSE*) corespunzător modelului (A.1.19).

A.1.4. Forma generală a metodelor iterative de optimizare

Considerăm o funcție $f : \mathbb{R}^n \to \mathbb{R}$ de clasă $C^2_{\mathbb{R}^n}$ al cărei punct de minim notat $\boldsymbol{x}^*$ presupunem că există şi ne propunem să-l determinăm. În general, funcția f poate fi neliniară în argument, astfel că pentru determinarea minimului acesteia se utilizează o procedură iterativă care să permită explorarea domeniului de definiție al funcției într-un mod cât mai eficient.

Ideea de bază a metodelor iterative este următoarea: fiind dat un punct inițial $\boldsymbol{x}_0 \in \mathbb{R}^n$ se generează o secvență $\{\boldsymbol{x}_k\}_{k \in \mathbb{N}}$ pe baza unei reguli de forma $\boldsymbol{x}_{k+1} = \boldsymbol{x}_{k+1}(\boldsymbol{x}_k)$, $k = 1, 2, \ldots$. Un algoritm iterativ este acceptabil atunci când este garantată convergența secvenței generate către soluția optimală a problemei. Dacă condiția de convergență este satisfăcută, procedura iterativă se opreşte atunci când o anumită condiție de stop este îndeplinită.

Fie $\boldsymbol{x}_k$ punctul determinat la sfârşitul iterației $k-1$. La iterația *k*, pornind din $\boldsymbol{x}_k$ se determină direcția $\boldsymbol{d}_k$ de căutare a minimului şi factorul $\eta_k > 0$ care precizează lungimea pasului ce urmează a fi făcut pornind din $\boldsymbol{x}_k$ în direcția $\boldsymbol{d}_k$. Noul punct $\boldsymbol{x}_{k+1}$ se determină utilizând relația

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \eta_k \boldsymbol{d}_k . \tag{A.1.23}$$

Dacă algoritmul vizează găsirea punctului de minim al funcției *f*, acest punct ar trebui să satisfacă inegalitatea

$$f(\boldsymbol{x}_{k+1}) = f(\boldsymbol{x}_k + \eta_k \boldsymbol{d}_k) < f(\boldsymbol{x}_k) . \tag{A.1.24}$$

În literatura neuro-fuzzy, factorul $\eta > 0$ este denumit *rată de învățare* (eng. *learning rate*).

Criteriul de stop este testat la începutul fiecărei iterații. Unul dintre criteriile de stop utilizate în metodele iterative de căutare a minimului unei funcții este

$$\left\| \nabla f(\boldsymbol{x}_k) \right\| < \varepsilon \tag{A.1.25}$$

unde $\| \ \|$ notează o normă vectorială convenabil aleasă (de regulă norma euclidiană) şi $\varepsilon > 0$ reprezintă toleranța impusă la startarea algoritmului. Dacă condiția (A.1.25) este satisfăcută, vectorul gradient în punctul curent $\nabla f(\boldsymbol{x}_k)$ tinde la zero şi secvența iterativă $\{\boldsymbol{x}_k\}_{k \in \mathbb{N}}$ tinde către un punct staționar al funcției *f*.

Alte criterii de stop frecvent utilizate sunt:

$$\left\| \boldsymbol{x}_k - \boldsymbol{x}_{k-1} \right\| < \varepsilon , \tag{A.1.26}$$

$$\left| f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k-1}) \right| < \varepsilon . \tag{A.1.27}$$

Criteriile (A.1.26), (A.1.27) se utilizează atunci când este de aşteptat ca algoritmul să conveargă rapid, în timp ce criteriul (A.1.25) este utilizat atunci când algoritmul converge mai lent. Criteriile de stop de forma

$$\frac{\left\| \boldsymbol{x}_k - \boldsymbol{x}_{k-1} \right\|}{\left\| \boldsymbol{x}_{k-1} \right\|} < \varepsilon \tag{A.1.28}$$

$$\frac{\left| f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k-1}) \right|}{\left| f(\boldsymbol{x}_{k-1}) \right|} < \varepsilon \tag{A.1.29}$$

testează valori relative, independente de unitățile de măsură.

Structura de bază a unui algoritm de optimizare iterativ este următoarea:

Algoritmul O (*algoritm de bază*).

Pas 0. *(Inițializare) Pentru* $k = 0$ *se precizează punctul inițial* $\boldsymbol{x}_0 \in \mathbb{R}^n$*, valoarea de prag* $\varepsilon > 0$ *şi numărul maxim de iterații* $N \in \mathbb{N}$.

Pas 1. *(Testare criterii de stop)*
1.1 Dacă $\left\| \nabla f(\boldsymbol{x}_k) \right\| < \varepsilon$*, treci la Pasul 6.*
1.2. Dacă $k = N$*, treci la Pasul 6.*

Pas 2. *(Determinarea direcției de căutare) Utilizând o anumită schemă iterativă se determină direcția* $\boldsymbol{d}_k$ *de descreştere a funcției f.*

Pas 3. *(Determinarea lungimii pasului) Se determină factorul* η_k *care precizează lungimea pasului, astfel încât să fie satisfăcută condiția de descreştere*
$$f(\boldsymbol{x}_k + \eta_k \boldsymbol{d}_k) < f(\boldsymbol{x}_k).$$

Pas 4. *(Actualizare) Se calculează următorul punct*
$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \eta_k \boldsymbol{d}_k .$$

Pas 5. *(Ciclare) Se setează* $k := k+1$ *şi se trece la Pasul 1.*

Pas 6. *(Finalizare) Se returnează punctul final* $\boldsymbol{x}_f = \boldsymbol{x}_k$.

Evident că la pasul 1.1 în locul criteriului (A.1.25) poate fi utilizat oricare dintre criteriile de stop (A.1.26)–(A.1.29) prezentate anterior.

Diferențele dintre diferite metode de minimizare constau în modul de determinare a direcților succesive de căutare. Odată determinată această direcție, unii algoritmi consideră valori constante ale lui η, alți algoritmi „adaptează" lungimea pasului la fiecare iterație.

A.1.5. Metode de căutare liniară

Metodele de optimizare unidimensională, numite şi *metode de căutare liniară*, stau la baza metodelor de optimizare a funcțiilor multivariabile. Acest fapt poate fi observat la pasul 3 al algoritmului general de optimizare unde factorul $\eta > 0$ este frecvent determinat astfel încât să fie minimizată funcția $\varphi(\eta) = f(\boldsymbol{x}_k + \eta \boldsymbol{d}_k)$ cu $\eta \geq 0$.

Pentru fixarea cadrului de discuție, considerăm o funcție $\varphi : \mathbb{R} \to \mathbb{R}$, care admite un punct de minim $\eta^* \in [0, +\infty)$, adică

$$\varphi(\eta^*) = \min_{\eta \geq 0} \varphi(\eta) . \tag{A.1.30}$$

Dacă există un interval închis $[a,b] \subset [0,+\infty)$ astfel încât $\eta^* \in [a,b]$ (fără ca poziția efectivă a punctului de minim să fie cunoscută), intervalul $[a,b]$ este numit interval de căutare (sau interval de incertitudine) pentru problema de minimizare unidimensională $\min_{\eta \geq 0} \varphi(\eta)$.

Una dintre cele mai frecvent utilizate metode de căutare liniară este *metoda secțiunii de aur* (eng. *golden search method*). Această metodă poate fi aplicată în ipoteza că funcția φ este unimodală pe intervalul de căutare $[a,b]$, adică este strict descrescătoare pe intervalul $[a,\eta^*]$ şi strict crescătoare pe intervalul $[\eta^*,b]$. În această ipoteză, oricare ar fi două puncte $\lambda, \mu \in [a,b]$, cu $\lambda < \mu$, au loc următoarele implicații:

- dacă $\mu < \eta^*$, atunci $\varphi(\lambda) > \varphi(\mu)$;
- dacă $\lambda > \eta^*$, atunci $\varphi(\lambda) < \varphi(\mu)$;

Ipoteza de unimodalitate nu presupune continuitatea sau derivabilitatea funcției de minimizat. În plus, utilizând această ipoteză, intervalul de căutare poate fi restrâns eliminând acele porțiuni care, cu certitudine, nu pot conține punctul de minim. Aceasta se realizează observând că, dacă funcția φ este unimodală pe intervalul $[a,b]$ şi considerăm $\lambda, \mu \in [a,b]$, cu $\lambda < \mu$, au loc următoarele implicații:

- dacă $\varphi(\lambda) \le \varphi(\mu)$, atunci $[a, \mu]$ este un interval unimodal pentru φ;
- dacă $\varphi(\lambda) \ge \varphi(\mu)$, atunci $[\lambda, b]$ este un interval unimodal pentru φ.

Pe baza acestor observații, principiul de bază al metodei secțiunii de aur este următorul. Considerăm că la iterația k intervalul de căutare este $[a_k, b_k]$, interval pe care funcția φ este unimodală. Considerăm două puncte $\lambda_k, \mu_k \in [a_k, b_k]$, cu $\lambda_k < \mu_k$, şi aplicăm observația precedentă:

- dacă $\varphi(\lambda_k) \le \varphi(\mu_k)$, atunci $[a_k, \mu_k]$ este un interval unimodal pentru φ şi setăm $a_{k+1} = a_k$, $b_{k+1} = \mu_k$;
- dacă $\varphi(\lambda_k) \ge \varphi(\mu_k)$, atunci $[\lambda_k, b_k]$ este un interval unimodal pentru φ şi setăm $a_{k+1} = \lambda_k$, $b_{k+1} = b_k$.

Pentru a reduce numărul de evaluări ale funcției φ la fiecare iterație, punctele λ_k şi μ_k se aleg astfel încât să fie satisfăcute următoarele condiții:

- $\mu_k - a_k = b_k - \lambda_k$,
- rata de reducere a intervalului de incertitudine la fiecare iterație să fie aceeaşi, $b_{k+1} - a_{k+1} = \rho(b_k - a_k)$, $\rho \in (0, 1)$,
- la fiecare iterație să fie introdus numai un singur punct nou.

Ținând cont de primele două cerințe se ajunge la

$$\begin{aligned} \lambda_k &= a_k + (1-\rho)(b_k - a_k), \\ \mu_k &= a_k + \rho(b_k - a_k). \end{aligned} \tag{A.1.31}$$

Aplicând aceste relații pentru intervalul $[a_{k+1}, b_{k+1}]$ şi impunând condiția ca la o iterație sa fie introdus numai un singur punct nou, se ajunge la concluzia că factorul $\rho \in (0, 1)$ trebuie să fie ales astfel încât $\rho^2 = 1 - \rho$. Singura soluție pozitivă a acestei ecuații fiind

$$\rho = \frac{\sqrt{5}-1}{2} \approx 0.618, \tag{A.1.32}$$

relațiile (A.1.31) pot fi rescrise în forma

$$\begin{aligned} \lambda_k &= a_k + 0.382(b_k - a_k), \\ \mu_k &= a_k + 0.618(b_k - a_k). \end{aligned} \tag{A.1.33}$$

Din acest motiv metoda secțiunii de aur mai este numită şi *metoda 0.618*.

Structura metodei secțiunii de aur pentru minimizarea unei funcții φ unimodale pe un interval $[a, b] \subset [0, +\infty)$ este următoarea:

Algoritmul GSM (*Golden Section Method*).

Pas 0. *(Inițializare) Pentru $k = 0$ se precizează intervalul inițial de căutare $[a_0, b_0]$, valoarea de prag $\varepsilon > 0$ şi numărul maxim de iterații $N \in \mathbb{N}$. Se calculează*

$$\begin{aligned} \lambda_0 &= a_0 + 0.382(b_0 - a_0), \\ \mu_0 &= a_0 + 0.618(b_0 - a_0). \end{aligned}$$

Pas 1. *(Testare criterii de stop)*
1.1 Dacă $b_k - a_k < \varepsilon$, *treci la Pasul 6.*
1.2. Dacă $k = N$, *treci la Pasul 6.*

Pas 2. *(Compararea valorilor funcției) Dacă* $\varphi(\lambda_k) \le \varphi(\mu_k)$ *treci la pasul 3, altfel, dacă* $\varphi(\lambda_k) > \varphi(\mu_k)$ *treci la pasul 4.*

Pas 3. *(Cazul 1) Se setează*

$$a_{k+1} = \lambda_k, \quad b_{k+1} = b_k, \quad \lambda_{k+1} = \mu_k, \quad \varphi(\lambda_{k+1}) = \varphi(\mu_k).$$

Se calculează noul punct $\mu_{k+1} = a_{k+1} + 0.618(b_{k+1} - a_{k+1})$ *şi valoarea funcției în acest punct,* $\varphi(\mu_{k+1})$. *Se trece la pasul 5.*

Pas 4. *(Cazul 2) Se setează*

$$a_{k+1} = a_k, \quad b_{k+1} = \mu_k, \quad \mu_{k+1} = \lambda_k, \quad \varphi(\mu_{k+1}) = \varphi(\lambda_k).$$

Se calculează noul punct $\lambda_{k+1} = a_{k+1} + 0.382(b_{k+1} - a_{k+1})$ *şi valoarea funcției în acest punct,* $\varphi(\lambda_{k+1})$. *Se trece la pasul 5.*

Pas 5. *(Ciclare) Se setează* $k := k+1$ *şi se trece la Pasul 1.*

Pas 6. *(Finalizare) Se returnează punctul final* $\eta_f = \frac{1}{2}(\lambda_k + \mu_k)$.

În afara metodei secțiunii de aur, pentru căutare liniară mai există numeroase metode bazate pe interpolare pătratică sau cubică, sau metode de căutare inexactă. Aceste metode presupun însă că funcția criteriu satisface unele condiții suplimentare, de exemplu să fie de două ori derivabilă.

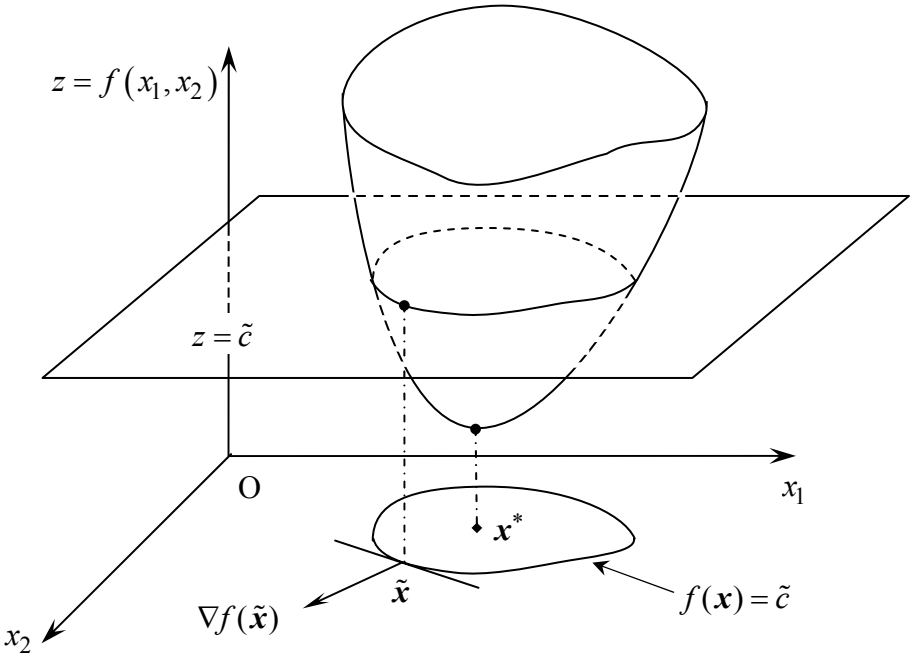


Figura A.2.1. Interpretarea geometrică a gradientului într-un punct pentru o funcție reală de două variabile.

A.2.Metode de optimizare bazate pe gradient

O categorie importantă de metode de minimizare se bazează pe calcularea gradientului funcţiei obiectiv în punctul curent pentru determinarea direcţiei de căutare, fiind denumite *metode de gradient*. Din această categorie fac parte: *metoda de descreştere maximă* (eng. *steepest – descent method*) şi *metodele de tip Gauss – Newton*. Aceste metode sunt frecvent utilizate în probleme de aproximare a modelelor neliniare, fiind astfel în strânsă legătură cu tehnicile neuro-fuzzy.

Metodele bazate pe gradient utilizează proprietăţile geometrice ale gradientului unei funcţii într-un punct $\tilde{\boldsymbol{x}}$ pentru care $f(\tilde{\boldsymbol{x}}) = \tilde{c}$. Dacă gradientul funcţiei f în punctul $\tilde{\boldsymbol{x}}$ este nenul, $\nabla f(\tilde{\boldsymbol{x}}) \neq 0$, atunci $\nabla f(\tilde{\boldsymbol{x}})$ este un vector ortogonal pe vectorul tangent la o curbă netedă arbitrară ce trece prin $\tilde{\boldsymbol{x}}$ pe mulţimea de nivel constant $\boldsymbol{f}(\boldsymbol{x}) = \tilde{c}$ (figura A.2.1). Reamintim faptul că, fiind dată o constantă $c \in \mathbb{R}$, *mulţimea de nivel c* a funcţiei f este mulţimea punctelor $\boldsymbol{x} \in \mathbb{R}^n$ ce satisfac $f(\boldsymbol{x}) = c$.

Ţinând cont de această interpretare, pentru ca o direcţie de căutare a minimului $\boldsymbol{d} \neq 0$ să fie fezabilă în punctul $\tilde{\boldsymbol{x}}$ trebuie ca $\boldsymbol{d}^T \cdot \nabla \boldsymbol{f}(\tilde{\boldsymbol{x}}) < 0$ (figura A.2.2). Se observă că direcţia $\boldsymbol{d} = -\nabla \boldsymbol{f}(\tilde{\boldsymbol{x}})$ este fezabilă în $\tilde{\boldsymbol{x}}$.

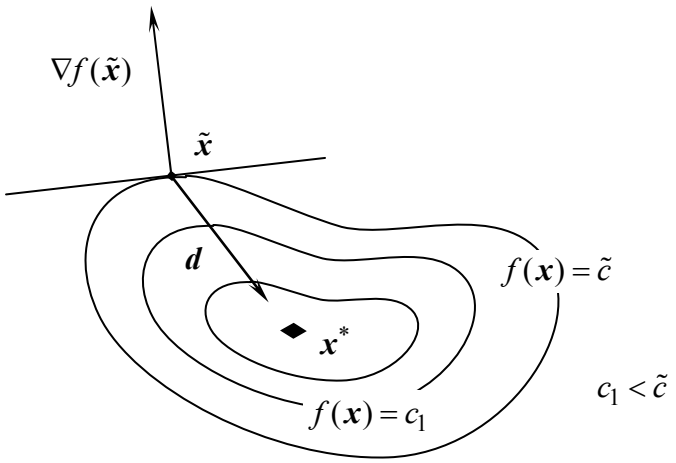


Figura A.2.1. *Alegerea unei direcţii fezabile de căutare a minimului.*

A.2.1. Metoda steepest-descent (de descreştere maximă)

În această metodă, la fiecare iteraţie este ales pasul $\eta \boldsymbol{d}$ astfel încât să fie asigurată descreşterea maximă a funcţiei obiectiv în direcţia $\boldsymbol{d}$ considerată. Concret, la iteraţia k, pornind din punctul $\boldsymbol{x}_k$ este efectuată o căutare a minimului în direcţia opusă gradientului, $\boldsymbol{d}_k = -\nabla f(\boldsymbol{x}_k)$. Mărimea pasului este determinată astfel încât să fie minimizată funcţia: $\varphi_k(\eta) = f\left(\boldsymbol{x}_k - \eta \nabla f(\boldsymbol{x}_k)\right)$, alegând

$$\eta_k = \arg\min_{\eta \geq 0} f\left(\boldsymbol{x}^{(k)} - \eta \nabla f(\boldsymbol{x}^k)\right). \tag{A.2.1}$$

Se obține astfel punctul

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \eta_k \nabla f(\boldsymbol{x}_k), \tag{A.2.2}$$

din care va fi efectuată o nouă căutare în acelaşi mod la următoarea iterație.

Se poate demonstra că dacă secvența $\boldsymbol{x}_0$, $\boldsymbol{x}_1$, … a fost determinată prin algoritmul de descreştere maximă pentru minimizarea unei funcții f pornind din $\boldsymbol{x}_0$, atunci pentru orice $k \in \mathbb{N}$ vectorul $\boldsymbol{x}_{k+1} - \boldsymbol{x}_k$ este ortogonal pe $\boldsymbol{x}_{k+2} - \boldsymbol{x}_{k+1}$ (vezi figura A.2.2).

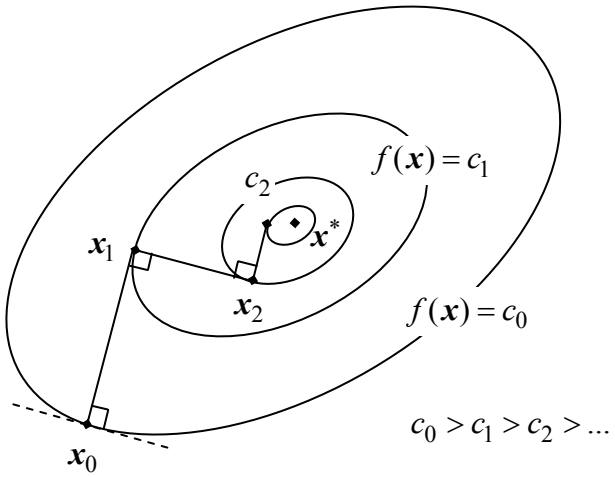


Figura A.2.2. *Ilustrarea primelor iterații ale algoritmului steepest-descent.*

Aşa cum s-a văzut mai sus, dacă $\nabla f(\boldsymbol{x}_k) \neq 0$, utilizând acest algoritm, din $\boldsymbol{x}_k$ se determină noul punct $\boldsymbol{x}_{k+1}$ care satisface condiția de descreştere $f(\boldsymbol{x}_{k+1}) < f(\boldsymbol{x}_k)$.

Dacă se ajunge la un punct $\boldsymbol{x}_k$ pentru care $\nabla f(\boldsymbol{x}_k) = 0$, atunci $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k$. Cum punctul $\boldsymbol{x}_k$ satisface condiția necesară de minim, nu mai are sens să aplicăm, algoritmul în continuare. În practică, în locul condiției $\nabla f(\boldsymbol{x}_k) = 0$, pentru oprirea algoritmului iterativ, se utilizează criterii de stop de forma (A.1.25)–(A.1.29).

Algoritmului steepest-descent de minimizare a unei funcții obiectiv $f : \mathbb{R}^n \to \mathbb{R}$, de clasa $C^1_{\mathbb{R}^n}$ este următoarea:

Algoritmul SD (*metoda steepest descent*).

Pas 0. *(Inițializare) Pentru $k = 0$ se precizează punctul inițial $\boldsymbol{x}_0 \in \mathbb{R}^n$, valoarea de prag $\varepsilon > 0$ şi numărul maxim de iterații $N \in \mathbb{N}$.*

Pas 1. *(Testare criterii de stop) Fie $\boldsymbol{g}_k = \nabla f(\boldsymbol{x}_k)$.*

1.1 Dacă $\|\boldsymbol{g}_k\| < \varepsilon$, treci la Pasul 5.

1.2. Dacă $k = N$, treci la Pasul 5.

Pas 2. *(Determinarea lungimii pasului) Se determină factorul* η_k *astfel încât*

$$f(\boldsymbol{x}_k - \eta_k \boldsymbol{g}_k) = \min_{\eta>0} f(\boldsymbol{x}_k - \eta\, \boldsymbol{g}_k).$$

Pas 3. *(Actualizare) Se calculează următorul punct*

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \eta_k \boldsymbol{g}_k .$$

Pas 4. *(Ciclare) Se setează* $k := k+1$ *şi se trece la Pasul 1.*

Pas 5. *(Finalizare) Se returnează punctul final* $\boldsymbol{x}_f = \boldsymbol{x}_k$.

Exemplul A.2.1. Pentru a ilustra aplicarea metodei steepest-descent considerăm forma pătratică

$$f : \mathbb{R}^n \to \mathbb{R},\ f(\boldsymbol{x}) = \tfrac{1}{2}\boldsymbol{x}^T \boldsymbol{A}\, \boldsymbol{x} + \boldsymbol{b}^T \boldsymbol{x} + c, \tag{A.2.3}$$

pentru care matricea $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ este simetrică şi pozitiv definită ($\boldsymbol{A}^T = \boldsymbol{A} > 0$), $\boldsymbol{b} \in \mathbb{R}^n$ şi $c \in \mathbb{R}$. Aşa cum a fost prezentat în exemplul A.1.3, funcția f admite unic punct de minim, $\boldsymbol{x}^* = -\boldsymbol{A}^{-1}\boldsymbol{b}$, care reprezintă soluția ecuației $\nabla f(\boldsymbol{x}) = 0$.

Aplicând algoritmul steepest-descent, la iterația k în punctul curent $\boldsymbol{x}_k$, se calculează gradientul $\boldsymbol{g}_k = \nabla f(\boldsymbol{x}_k) = \boldsymbol{A}\, \boldsymbol{x}_k + \boldsymbol{b}$ şi presupunem că satisface condiția $\boldsymbol{g}_k \neq 0$ (adică $\boldsymbol{x}_k$ nu este punct de minim pentru f).

Factorul $\eta_k > 0$ la iterația curentă se determină astfel încât $\eta_k = \arg\min_{\eta\geq 0} \varphi_k(\eta)$, unde

$$\begin{aligned}\varphi_k(\eta) &= f(\boldsymbol{x}_k - \eta \boldsymbol{g}_k) = \\ &= \tfrac{1}{2}\boldsymbol{g}_k^T \boldsymbol{A}\boldsymbol{g}_k \eta^2 - \left[\boldsymbol{g}_k^T \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{g}_k^T \boldsymbol{b}\right]\eta + \left[\tfrac{1}{2}\boldsymbol{x}_k^T \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{b}^T \boldsymbol{x}_k + c\right] = \\ &= \tfrac{1}{2}\boldsymbol{g}_k^T \boldsymbol{A}\boldsymbol{g}_k \eta^2 - \boldsymbol{g}_k^T \boldsymbol{g}_k \eta + f(\boldsymbol{x}_k).\end{aligned}$$

Funcția $\varphi_k(\eta)$ are gradul 2 în η şi coeficientul termenului dominant pozitiv ($\tfrac{1}{2}\boldsymbol{g}_k^T \boldsymbol{A}\boldsymbol{g}_k > 0$ deoarece matricea $\boldsymbol{A}$ este pozitiv definită), astfel că admite minim în

$$\eta_k = \frac{\boldsymbol{g}_k^T \boldsymbol{g}_k}{\boldsymbol{g}_k^T \boldsymbol{A}\, \boldsymbol{g}_k}.$$

Astfel, la iterația k a algoritmului steepest descent, dacă $\boldsymbol{g}_k \neq 0$, se calculează următorul punct utilizând

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \frac{\boldsymbol{g}_k^T \boldsymbol{g}_k}{\boldsymbol{g}_k^T \boldsymbol{A}\, \boldsymbol{g}_k}\boldsymbol{g}_k . \tag{A.2.4}$$

Dacă pentru minimizarea funcției pătratice (A.2.3) se utilizează un factor $\eta > 0$ constant, convergența algoritmului este garantată numai dacă

$$0 < \eta < \frac{2}{\lambda_{\max}(\boldsymbol{A})} \tag{A.2.5}$$

unde $\lambda_{\max}(\boldsymbol{A})$ notează valoarea proprie maximă a matricei $\boldsymbol{A}$ (care este strict pozitivă datorită faptului că $\boldsymbol{A}$ este pozitiv definită). ■

A.2.2. Metoda lui Newton

Metoda lui Newton de minimizarea a unei funcții obiectiv f se bazează pe aproximarea lui f printr-o formă pătratică şi minimizarea acestei forme. Astfel, la fiecare iterație direcția de căutare este determinată utilizând a doua derivată a funcției obiectiv f.

Presupunem că funcția $f:\mathbb{R}^n \to \mathbb{R}$ al cărei punct de minim dorim să-l determinăm printr-o procedură iterativă este de clasă $C^2_{\mathbb{R}^n}$. La iterația k, punctul curent $\boldsymbol{x}_k$ este astfel încât matricea Hessian în $\boldsymbol{x}_k$, notată $\boldsymbol{H}_k = \nabla^2 f(\boldsymbol{x}_k)$, este pozitiv definită. Dacă punctul curent $\boldsymbol{x}_k$ este suficient de apropiat de punctul minim local $\boldsymbol{x}^*$, funcția f poate fi aproximată în vecinătatea lui $\boldsymbol{x}_k$ prin

$$f(\boldsymbol{x}_k + \boldsymbol{s}) \approx q_k(\boldsymbol{s}) = f(\boldsymbol{x}_k) + \boldsymbol{g}_k^T \boldsymbol{s} + \frac{1}{2}\boldsymbol{s}^T \boldsymbol{H}_k \boldsymbol{s}$$

unde $\boldsymbol{g}_k = \nabla f(\boldsymbol{x}_k)$ şi $\boldsymbol{s} = \boldsymbol{x} - \boldsymbol{x}_k$. Această expresie reprezintă dezvoltarea funcției f în serie Taylor din care termenii de ordin mai mare ca 2 au fost neglijați.

Punctul de minim al acestei forme pătratice se determină egalând cu 0 gradientul său (vezi exemplul A.1.3). Astfel se obține $\boldsymbol{H}_k(\boldsymbol{x} - \boldsymbol{x}_k) + \boldsymbol{g}_k = 0$, care, în ipoteza că matricea $\boldsymbol{H}_k$ este nesingulară, admite soluția

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{H}_k^{-1}\boldsymbol{g}_k \tag{A.2.6}$$

Pasul acestei metode, având forma $\boldsymbol{d}_k = -\boldsymbol{H}_k^{-1}\boldsymbol{g}_k$, este denumit *pas Newton*.

Metoda Newton de minimizare a unei funcții obiectiv $f:\mathbb{R}^n \to \mathbb{R}$, de clasă $C^2_{\mathbb{R}^n}$, este următoarea:

Algoritmul Nw (*metoda lui Newton*).

Pas 0. *(Inițializare) Pentru $k = 0$ se precizează punctul inițial $\boldsymbol{x}_0 \in \mathbb{R}^n$, valoarea de prag $\varepsilon > 0$ şi numărul maxim de iterații $N \in \mathbb{N}$.*

Pas 1. *(Testare criterii de stop) Fie $\boldsymbol{g}_k = \nabla f(\boldsymbol{x}_k)$.*
1.1 Dacă $\|\boldsymbol{g}_k\| < \varepsilon$, treci la Pasul 5.
1.2. Dacă $k = N$, treci la Pasul 5.

Pas 2. *(Determinarea pasului) Se calculează $\boldsymbol{H}_k = \nabla^2 f(\boldsymbol{x}_k)$ şi se determină vectorul $\boldsymbol{s}_k$ soluție a ecuației*

$$\boldsymbol{H}_k \boldsymbol{s} = -\boldsymbol{g}_k . \tag{A.2.7}$$

Pas 3. *(Actualizare) Se calculează următorul punct*

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k .$$

Pas 4. *(Ciclare) Se setează $k := k+1$ şi se trece la Pasul 1.*

Pas 5. *(Finalizare) Se returnează punctul final $\boldsymbol{x}_f = \boldsymbol{x}_k$.*

Dacă funcția obiectiv f este pătratică şi $\boldsymbol{H}_0 = \nabla^2 f(\boldsymbol{x}_0)$ este pozitiv definită, metoda lui Newton conduce la punctul de minim într-un singur pas. Dacă funcția f nu este pătratică, atunci ar putea fi necesară efectuarea unui număr mai mare de iterații. Criteriul de stop poate fi oricare dintre cele prezentate anterior.

Dezavantajul major al metodei Newton îl reprezintă necesitatea calculării inversei matricei Hessian operație care, pe lângă faptul că presupune un număr mare de operații, poate introduce probleme numerice datorită erorilor de rotunjire.

Exemplul A.2.2. Pentru a ilustra aplicarea metodei lui Newton considerăm forma pătratică dată prin ecuația (A.2.3)

$$f : \mathbb{R}^n \to \mathbb{R},\ f(\boldsymbol{x}) = \tfrac{1}{2}\boldsymbol{x}^T \boldsymbol{A}\,\boldsymbol{x} + \boldsymbol{b}^T \boldsymbol{x} + c,$$

pentru care matricea $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ este simetrică şi pozitiv definită ($\boldsymbol{A}^T = \boldsymbol{A} > 0$), $\boldsymbol{b} \in \mathbb{R}^n$ şi $c \in \mathbb{R}$, utilizată şi în exemplul A.2.1. Aşa cum a fost prezentat în exemplul A.1.3, funcția f admite unic punct de minim, $\boldsymbol{x}^* = -\boldsymbol{A}^{-1}\boldsymbol{b}$.

Aplicând metoda lui Newton, la iterația $k = 0$ se porneşte din punctul inițial $\boldsymbol{x}_0$ şi se calculează gradientul $\boldsymbol{g}_0 = \nabla f(\boldsymbol{x}_0) = \boldsymbol{A}\,\boldsymbol{x}_0 + \boldsymbol{b}$. Presupunem că $\boldsymbol{x}_0 \neq \boldsymbol{x}^*$, astfel că $\boldsymbol{g}_0 \neq 0$. Matricea Hessian $\boldsymbol{H}_0 = \nabla^2 f(\boldsymbol{x}_0) = \boldsymbol{A}$ este pozitiv definită. La pasul 2 al algoritmului se rezolvă ecuația (A.2.7), care devine

$$\boldsymbol{A}\boldsymbol{s}_0 = -(\boldsymbol{A}\boldsymbol{x}_0 + \boldsymbol{b}),$$

astfel că se obține $\boldsymbol{s}_0 = -\boldsymbol{x}_0 - \boldsymbol{A}^{-1}\boldsymbol{b}$. Utilizând această expresie la pasul 3 pentru actualizare rezultă

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 + \boldsymbol{s}_0 = -\boldsymbol{A}^{-1}\boldsymbol{b} = \boldsymbol{x}^*.$$

Aceasta arată că, aplicând metoda lui Newton pentru minimizarea unei funcții pătratice se ajunge în punctul de minim într-o singură iterație. ■

A.2.3. Metoda Gauss-Netwton

În numeroase aplicații se ajunge la rezolvarea problemei celor mai mici pătrate în formă neliniară, si anume minimizarea unei funcții de forma

$$f : \mathbb{R}^n \to \mathbb{R},\ \ f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{r}^T(\boldsymbol{x})\cdot\boldsymbol{r}(\boldsymbol{x}) = \frac{1}{2}\sum_{k=1}^{m}\left[r_k(\boldsymbol{x})\right]^2, \qquad \text{(A.2.8)}$$

unde $\boldsymbol{r} : \mathbb{R}^n \to \mathbb{R}^m$, $\boldsymbol{r}(\boldsymbol{x}) = \left[r_1(\boldsymbol{x})\ r_2(\boldsymbol{x})\ ...\ r_m(\boldsymbol{x})\right]^T$, este o funcție neliniară. Dacă funcția $\boldsymbol{r}(\boldsymbol{x})$ este liniară, $\boldsymbol{r}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$, se ajunge la estimatorul celor mai mici pătrate prezentat în secțiunea A.1.3. La o asemenea problemă se poate ajunge, de exemplu, dacă dorim să rezolvăm un sistem de m ecuații neliniare

$$r_i(\boldsymbol{x}) = 0,\ \ i = 1, 2, ..., m.$$

Pentru a aplica metoda lui Newton de minimizare a funcției (A.2.8) este nevoie să calculăm mai întâi gradientul funcției f într-un punct oarecare. Pentru aceasta calculăm

$$\frac{\partial f}{\partial x_i}(\boldsymbol{x}) = \sum_{k=1}^{m} r_k(\boldsymbol{x}) \cdot \frac{\partial r_k}{\partial x_i}(\boldsymbol{x}) = \left[\frac{\partial \boldsymbol{r}}{\partial x_i}(\boldsymbol{x})\right]^T \cdot \boldsymbol{r}(\boldsymbol{x})$$

unde

$$\frac{\partial \boldsymbol{r}}{\partial x_i}(\boldsymbol{x}) = \left[\frac{\partial r_1}{\partial x_i}(\boldsymbol{x}) \; \frac{\partial r_2}{\partial x_i}(\boldsymbol{x}) \; \ldots \; \frac{\partial r_m}{\partial x_i}(\boldsymbol{x})\right]^T, \; i = \overline{1,n}.$$

Se introduce matricea

$$\boldsymbol{J}(\boldsymbol{x}) = \left[\frac{\partial \boldsymbol{r}}{\partial x_1}(\boldsymbol{x}) \; \ldots \; \frac{\partial \boldsymbol{r}}{\partial x_n}(\boldsymbol{x})\right] = \begin{bmatrix} \frac{\partial r_1}{\partial x_1}(\boldsymbol{x}) & \cdots & \frac{\partial r_1}{\partial x_n}(\boldsymbol{x}) \\ \vdots & \ddots & \vdots \\ \frac{\partial r_m}{\partial x_1}(\boldsymbol{x}) & \cdots & \frac{\partial r_m}{\partial x_n}(\boldsymbol{x}) \end{bmatrix} \in \mathbb{R}^{m \times n}, \tag{A.2.9}$$

care este numită *matricea Jacobian* a funcției vectoriale $\boldsymbol{r}$. Cu această notație, gradientul funcției f (A.2.8) devine:

$$\nabla f(\boldsymbol{x}) = \boldsymbol{J}(\boldsymbol{x})^T \cdot \boldsymbol{r}(\boldsymbol{x}). \tag{A.2.10}$$

Pentru calcularea matricei Hessian a funcției f este nevoie de derivatele parțiale de ordinul 2 ale lui f

$$\frac{\partial^2 f}{\partial x_i \partial x_j}(\boldsymbol{x}) = \frac{\partial}{\partial x_j}\left[\sum_{k=1}^{m} r_k(\boldsymbol{x}) \cdot \frac{\partial r_k}{\partial x_i}(\boldsymbol{x})\right] = \sum_{k=1}^{m} \frac{\partial r_k}{\partial x_j}(\boldsymbol{x}) \cdot \frac{\partial r_k}{\partial x_i}(\boldsymbol{x}) + \sum_{k=1}^{m} r_k(\boldsymbol{x}) \cdot \frac{\partial^2 r_k}{\partial x_i \partial x_j}(\boldsymbol{x}),$$

ceea ce conduce la

$$\boldsymbol{H}(\boldsymbol{x}) = \boldsymbol{J}(\boldsymbol{x})^T \cdot \boldsymbol{J}(\boldsymbol{x}) + \boldsymbol{S}(\boldsymbol{x}). \tag{A.2.11}$$

unde matricea $\boldsymbol{S}(\boldsymbol{x}) = \left[s_{ij}(\boldsymbol{x})\right] \in \mathbb{R}^{n \times n}$ are elementele

$$s_{ij}(\boldsymbol{x}) = \sum_{k=1}^{m} r_k(\boldsymbol{x}) \cdot \frac{\partial^2 r_k}{\partial x_i \partial x_j}(\boldsymbol{x}), \; i, j = \overline{1,n}. \tag{A.2.12}$$

Cu relațiile (A.2.10) şi (A.2.11), formula de actualizate (A.2.6) care intervine în aplicarea metodei lui Newton devine:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \left[\boldsymbol{J}(\boldsymbol{x}_k)^T \boldsymbol{J}(\boldsymbol{x}_k) + \boldsymbol{S}(\boldsymbol{x}_k)\right]^{-1} \boldsymbol{J}(\boldsymbol{x}_k)^T \boldsymbol{r}(\boldsymbol{x}_k), \tag{A.2.13}$$

În multe aplicații matricea $\boldsymbol{S}(\boldsymbol{x})$ poate fi neglijată în formula (A.2.13) deoarece elementele sale au valori foarte mici. Se obține astfel *metoda Gauss-Newton* de rezolvare a problemei celor mai mici pătrate în formă neliniară care utilizează formula de actualizare:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \left[\boldsymbol{J}(\boldsymbol{x}_k)^T \boldsymbol{J}(\boldsymbol{x}_k)\right]^{-1} \boldsymbol{J}(\boldsymbol{x}_k)^T \boldsymbol{r}(\boldsymbol{x}_k). \tag{A.2.14}$$

Facem precizarea că pentru a putea aplica această relație este necesar ca matricea Jacobian $\boldsymbol{J}(\boldsymbol{x}_k)$ să fie de rang complet pe coloane, adică $\operatorname{rang} \boldsymbol{J}(\boldsymbol{x}_k) = n$ (având implicit $n \le m$).

În concluzie, putem formula în continuare algoritmul Gauss-Newton de minimizare a funcției (A.2.8).

Algoritmul GN (*algoritmul Gauss-Newton*).

Pas 0. *(Inițializare) Pentru* $k = 0$ *se precizează punctul inițial* $\boldsymbol{x}_0 \in \mathbb{R}^n$, *valoarea de prag* $\varepsilon > 0$ *şi numărul maxim de iterații* $N \in \mathbb{N}$.

Pas 1. *(Testare criterii de stop) Se calculează* $\boldsymbol{r}_k = \boldsymbol{r}(\boldsymbol{x}_k)$, $\boldsymbol{J}_k = \boldsymbol{J}(\boldsymbol{x}_k)$ *şi* $\boldsymbol{g}_k = \boldsymbol{J}_k^T \boldsymbol{r}_k$.
1.1 Dacă $\|\boldsymbol{g}_k\| < \varepsilon$, *treci la Pasul 5.*
1.2. Dacă $k = N$, *treci la Pasul 5.*

Pas 2. *(Determinarea pasului) Se determină vectorul* $\boldsymbol{s}_k$ *soluție a ecuației*

$$\boldsymbol{J}_k^T \boldsymbol{J}_k \boldsymbol{s} = -\boldsymbol{g}_k .$$

Pas 3. *(Actualizare) Se calculează următorul punct*

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k .$$

Pas 4. *(Ciclare) Se setează* $k := k+1$ *şi se trece la Pasul 1.*

Pas 5. *(Finalizare) Se returnează punctul final* $\boldsymbol{x}_f = \boldsymbol{x}_k$.

A.2.4. Metode de tip Newton modificate

Metoda clasica a lui Newton, definită prin ecuația (A.2.6), este o metodă locală. Dacă punctul curent este departe de punctul de minim nu este garantat faptul că matricea Hessian este pozitiv definită, astfel că ecuația (A.2.7) poate să nu furnizeze direcția de descreştere a funcției f. Aceasta se datorează neglijării termenilor de grad mare din dezvoltarea în serie Taylor a funcției obiectiv.

Metoda lui Newton cu pas adaptiv

O modificare simplă a metodei Newton constă în adaptarea pasului de căutare la fiecare iterație, modificând ecuația (A.2.6) în

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \eta_k \boldsymbol{H}_k^{-1} \boldsymbol{g}_k \tag{A.2.15}$$

unde factorul $\eta_k > 0$ este selectat în aşa fel încât să fie minimizată funcția f.

Concret, metoda Newton cu pas adaptiv pentru minimizarea unei funcții obiectiv $f : \mathbb{R}^n \to \mathbb{R}$, de clasă $C^2_{\mathbb{R}^n}$, este următoarea:

Algoritmul NwLS (*metoda lui Newton & Line Search*).

Pas 0. *(Inițializare) Pentru* $k = 0$ *se precizează punctul inițial* $\boldsymbol{x}_0 \in \mathbb{R}^n$, *valoarea de prag* $\varepsilon > 0$ *şi numărul maxim de iterații* $N \in \mathbb{N}$.

Pas 1. *(Testare criterii de stop) Fie* $\boldsymbol{g}_k = \nabla f(\boldsymbol{x}_k)$.
1.1 Dacă $\|\boldsymbol{g}_k\| < \varepsilon$, *treci la Pasul 6.*
1.2. Dacă $k = N$, *treci la Pasul 6.*

Pas 2. *(Determinarea direcției de căutare) Se calculează* $\boldsymbol{H}_k = \nabla^2 f(\boldsymbol{x}_k)$ *şi se determină vectorul* $\boldsymbol{s}_k$ *soluție a ecuației*

$$\boldsymbol{H}_k \boldsymbol{s} = -\boldsymbol{g}_k .$$

Pas 3. *(Determinarea lungimii pasului) Se determină factorul* η_k *astfel încât*

$$f\left(\boldsymbol{x}_k + \eta_k \boldsymbol{s}_k\right) = \min_{\eta>0} f\left(\boldsymbol{x}_k + \eta\, \boldsymbol{s}_k\right). \tag{A.2.16}$$

Pas 4. *(Actualizare) Se calculează următorul punct*

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \eta_k \boldsymbol{s}_k .$$

Pas 5. *(Ciclare) Se setează* $k := k+1$ *şi se trece la Pasul 1.*

Pas 6. *(Finalizare) Se returnează punctul final* $\boldsymbol{x}_f = \boldsymbol{x}_k$.

Pentru rezolvarea problemei (A.2.16) se poate utiliza o metodă de căutare liniară, de exemplu metoda secțiunii de aur.

În practică, şi metoda Gauss-Newton este frecvent combinată cu o metodă de căutare liniară, utilizând în locul relației de actualizare (A.2.14) o formulă de tipul (A.2.15), şi anume

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \eta_k \left[\boldsymbol{J}(\boldsymbol{x}_k)^T \boldsymbol{J}(\boldsymbol{x}_k)\right]^{-1} \boldsymbol{J}(\boldsymbol{x}_k)^T \boldsymbol{r}(\boldsymbol{x}_k),$$

unde factorul $\eta_k > 0$ este selectat în aşa fel încât să fie minimizată funcția f (A.2.8).

O altă posibilitate de adaptare a pasului la fiecare iterație este aceea de a înjumătăți valoarea factorului η la fiecare iterație,

$$\eta_{k+1} = \frac{1}{2}\eta_k ,\ k = 0,1,2,\ldots,$$

cu η_0 ales 1 sau puțin mai mic. Este demonstrat faptul că metoda de înjumătățire a pasului converge către punctul de minim local.

Metoda Levenberg-Marquardt

Dacă matricea Hessian nu este pozitiv definită direcția Newton poate să indice către un punct de maxim sau către un punct în şa. Pentru a evita asemenea situații, se aduna la matricea Hessian $\boldsymbol{H}$ o matrice pozitiv definită $\boldsymbol{P}$ pentru a obține o matrice pozitiv definită. Aceasta metodă a fost introdusă pentru prima dată de Levenberg şi Marquardt în problema celor mai mici pătrate. Dacă se alege $\boldsymbol{P} = \lambda \boldsymbol{I}$, cu $\boldsymbol{I}$ matricea unitate, pentru valori suficient de mari ale factorului $\lambda > 0$ matricea $\boldsymbol{H} + \lambda \boldsymbol{I}$ este pozitiv definită.

Astfel, ecuația utilizată la pasul de actualizare este

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - (\boldsymbol{H}_k + \lambda_k \boldsymbol{I})^{-1} \boldsymbol{g}_k . \tag{A.2.17}$$

În practică, la fiecare iterație se inițializează λ_k cu o valoare mică ce este apoi crescută până când este satisfăcută condiția de descreştere $f(\boldsymbol{x}_{k+1}) < f(\boldsymbol{x}_k)$.

Pentru $\lambda \to 0$ se obține metoda clasică a lui Newton în timp ce pentru $\lambda \to \infty$ rezultă metoda steepest-descent.

Cele două metode pot fi combinate, considerând relația de actualizare de forma

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \eta_k (\boldsymbol{H}_k + \lambda \boldsymbol{I})^{-1} \boldsymbol{g}_k . \tag{A.2.18}$$

A.2.5. Metode quasi-Newton

În cadrul metodelor de tip quasi-Newton se construieşte iterativ o aproximare $\boldsymbol{M}$ a matricei Hessian $\boldsymbol{H}$ sau a inversei sale $\boldsymbol{H}^{-1}$ pe măsură ce algoritmul progresează. În continuare ilustrăm pe scurt cel de-al doilea tip de metode.

Cu notaţiile uzuale, în vecinătatea punctului de minim se consideră aproximarea

$$\boldsymbol{g}_{k+1} - \boldsymbol{g}_k \approx \boldsymbol{H}_{k+1}\left(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\right).$$

Pornind de la relaţia

$$\Delta\boldsymbol{x}_k = \boldsymbol{M}_{k+1}\,\Delta\boldsymbol{g}_k, \tag{A.2.19}$$

unde $\Delta\boldsymbol{x}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$, $\Delta\boldsymbol{g}_k = \boldsymbol{g}_{k+1} - \boldsymbol{g}_k$, se construieşte aproximarea $\boldsymbol{M}_{k+1}$ a matricei $\boldsymbol{H}_{k+1}^{-1}$. Iniţial se consideră $\boldsymbol{M}_0 = \boldsymbol{I}$. Metodele bazate pe acest principiu diferă între ele prin relaţiile de recurenţă utilizate pentru actualizarea matricei $\boldsymbol{M}_k$ la fiecare iteraţie. Două dintre metodele frecvent utilizate sunt Davidson-Fletcher-Powell (DFP) şi Broyden-Fletcher-Golfarb-Shanno (BFGS) care utilizează relaţiile:

(DFP)
$$\boldsymbol{M}_{k+1} = \boldsymbol{M}_k + \frac{\Delta\boldsymbol{x}_k \cdot \Delta\boldsymbol{x}_k^T}{\Delta\boldsymbol{x}_k^T \cdot \Delta\boldsymbol{g}_k} - \frac{\boldsymbol{M}_k \cdot \Delta\boldsymbol{g}_k \cdot \Delta\boldsymbol{g}_k^T \cdot \boldsymbol{M}_k}{\Delta\boldsymbol{g}_k^T \cdot \boldsymbol{M}_k \cdot \Delta\boldsymbol{g}_k}, \tag{A.2.20}$$

(BFGS)
$$\boldsymbol{M}_{k+1} = \left(\boldsymbol{I} - \frac{\Delta\boldsymbol{x}_k \cdot \Delta\boldsymbol{g}_k^T}{\Delta\boldsymbol{x}_k^T \cdot \Delta\boldsymbol{g}_k}\right)\boldsymbol{M}_k\left(\boldsymbol{I} - \frac{\Delta\boldsymbol{g}_k \cdot \Delta\boldsymbol{x}_k^T}{\Delta\boldsymbol{x}_k^T \cdot \Delta\boldsymbol{g}_k}\right) + \frac{\Delta\boldsymbol{x}_k \cdot \Delta\boldsymbol{x}_k^T}{\Delta\boldsymbol{x}_k^T \cdot \Delta\boldsymbol{g}_k}. \tag{A.2.21}$$

Structura generală a unui algoritm de tip quasi-Newton este următoarea:

Algoritmul qNw (*algoritm general quasi-Newton*).

Pas 0. *(Iniţializare) Pentru* $k = 0$ *se precizează punctul iniţial* $\boldsymbol{x}_0 \in \mathbb{R}^n$, *aproximarea iniţială* $\boldsymbol{M}_0 \in \mathbb{R}^{n\times n}$, *valoarea de prag* $\varepsilon > 0$ *şi numărul maxim de iteraţii* $N \in \mathbb{N}$.

Pas 1. *(Testare criterii de stop) Fie* $\boldsymbol{g}_k = \nabla f(\boldsymbol{x}_k)$.
1.1 Dacă $\|\boldsymbol{g}_k\| < \varepsilon$, *treci la Pasul 7.*
1.2. Dacă $k = N$, *treci la Pasul 7.*

Pas 2. *(Determinarea direcţiei de căutare) Se calculează vectorul* $\boldsymbol{s}_k$

$$\boldsymbol{s}_k = -\boldsymbol{M}_k\boldsymbol{g}_k.$$

Pas 3. *(Determinarea lungimii pasului) Se determină factorul* η_k *astfel încât*

$$f\left(\boldsymbol{x}_k + \eta_k\boldsymbol{s}_k\right) = \min_{\eta>0} f\left(\boldsymbol{x}_k + \eta\,\boldsymbol{s}_k\right).$$

Pas 4. *(Actualizare* $\boldsymbol{x}_{k+1}$*) Se calculează următorul punct*

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \eta_k\boldsymbol{s}_k.$$

Pas 5. *(Actualizare* $\boldsymbol{M}_{k+1}$*) Se calculează matricea* $\boldsymbol{M}_{k+1}$ *astfel încât să fie satisfăcută ecuaţia quasi-Newton (A.2.19).*

Pas 6. *(Ciclare) Se setează* $k := k+1$ *şi se trece la Pasul 1.*

Pas 7. *(Finalizare) Se returnează punctul final* $\boldsymbol{x}_f = \boldsymbol{x}_k$.

A.3. Metode bazate pe direcții conjugate

A.3.1. Metoda direcțiilor conjugate

Metodele de gradient conjugat reprezintă un compromis între metodele de tip Newton şi metoda de descreştere maximă. Metodele de gradient conjugat necesită numai derivatele de ordinul 1 nefiind necesară calcularea derivatelor de ordinul 2, aşa cum se întâmplă în cazul metodelor de tip Newton. Metodele bazate pe direcții conjugate sunt mai răspândite decât metoda steepest descent, dar mai lente decât metodele de tip Newton. Factorul decisiv asupra eficienței unei metode iterative de căutare este direcția de căutare actualizată la fiecare iterație.

Introducem în continuare câteva noțiuni necesare în dezvoltarea teoretică.

Fie $\boldsymbol{Q} \in \mathbb{R}^{n\times n}$ o matrice simetrică şi pozitiv definită. Se spune că direcțiile $\boldsymbol{d}_0$, $\boldsymbol{d}_1$, ..., $\boldsymbol{d}_m$, $m \le n-1$, sunt $\boldsymbol{Q}$-conjugate dacă $\boldsymbol{d}_i^T \boldsymbol{Q} \boldsymbol{d}_j = 0$, pentru orice $i \ne j$. În particular, pentru $\boldsymbol{Q} = \boldsymbol{I}_n$, se obține definiția unei mulțimi de vectori ortogonali.

Se poate demonstra că dacă direcțiile $\boldsymbol{d}_0$, $\boldsymbol{d}_1$, ..., $\boldsymbol{d}_m \in \mathbb{R}^n \setminus \{0\}$, $m \le n-1$, sunt $\boldsymbol{Q}$-conjugate, atunci sunt liniar independente.

Algoritmul general de minimizare a unei funcții $f : \mathbb{R}^n \to \mathbb{R}$ prin metoda direcțiilor conjugate este prezentat în continuare.

Algoritmul GCDM (*metoda generală a direcțiilor conjugate*).

Pas 0. *(Inițializare) Pentru* $k = 0$ *se precizează punctul inițial* $\boldsymbol{x}_0 \in \mathbb{R}^n$, *matricea pozitiv definită* $\boldsymbol{Q} > 0$ *şi valoarea de prag* $\varepsilon > 0$.
Se calculează $\boldsymbol{g}_0 = \nabla f(\boldsymbol{x}_0)$ *şi se alege* $\boldsymbol{d}_0$ *astfel încât* $\boldsymbol{d}_0^T \boldsymbol{g}_0 < 0$

Pas 1. *(Testare criteriu de stop) Fie* $\boldsymbol{g}_k = \nabla f(\boldsymbol{x}_k)$. *Dacă* $\|\boldsymbol{g}_k\| < \varepsilon$, *treci la Pasul 6.*

Pas 2. *(Determinarea lungimii pasului) Se determină factorul* η_k *astfel încât*

$$f(\boldsymbol{x}_k + \eta_k \boldsymbol{d}_k) = \min_{\eta} f(\boldsymbol{x}_k + \eta \boldsymbol{d}_k). \quad \text{(A.3.1)}$$

Pas 3. *(Actualizare) Se calculează următorul punct*

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \eta_k \boldsymbol{d}_k. \quad \text{(A.3.2)}$$

Pas 4. *(Determinarea următorului pas) Se calculează direcția* $\boldsymbol{d}_{k+1}$ *care este* $\boldsymbol{Q}$*-conjugată direcțiilor* $\boldsymbol{d}_0$, $\boldsymbol{d}_1$, ..., $\boldsymbol{d}_k$, *adică:*

$$\boldsymbol{d}_{k+1}^T \boldsymbol{Q} \boldsymbol{d}_j = 0,\ j = 0,1,...,k. \quad \text{(A.3.3)}$$

Pas 5. *(Ciclare) Se setează* $k := k+1$ *şi se trece la Pasul 1.*

Pas 6. *(Finalizare) Se returnează punctul final* $\boldsymbol{x}_f = \boldsymbol{x}_k$.

Exemplul A.3.1. Pentru a ilustra aplicarea metodei bazate pe direcții conjugate considerăm forma pătratică precizată prin relația

$$f:\mathbb{R}^n\to\mathbb{R},\ f(\boldsymbol{x})=\tfrac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}+\boldsymbol{b}^T\boldsymbol{x}+c, \tag{A.3.4}$$

în care matricea $\boldsymbol{A}\in\mathbb{R}^{n\times n}$ este simetrică şi pozitiv definită ($\boldsymbol{A}^T=\boldsymbol{A}>0$), $\boldsymbol{b}\in\mathbb{R}^n$ şi $c\in\mathbb{R}$, utilizată şi în exemplele A.2.1 şi A.2.2. Această funcție admite un punctul de minim unic, $\boldsymbol{x}^*=-\boldsymbol{A}^{-1}\boldsymbol{b}$. Pentru minimizarea funcției f vom utiliza direcții $\boldsymbol{A}$-conjugate.

Pentru $k=0$ se alege $\boldsymbol{x}_0$, se calculează $\boldsymbol{g}_0=\nabla f(\boldsymbol{x}_0)=\boldsymbol{A}\boldsymbol{x}_0+\boldsymbol{b}$. Presupunând că $\boldsymbol{g}_0\neq 0$ se alege direcția $\boldsymbol{d}_0$ astfel încât $\boldsymbol{d}_0^T\boldsymbol{g}_0<0$, ceea ce arată că $\boldsymbol{d}_0$ este o direcție de descreştere pentru funcția f pornind din $\boldsymbol{x}_0$.

La iterația $k\geq 0$ se calculează

$$\eta_k=\arg\min_{\eta} f(\boldsymbol{x}_k+\eta\boldsymbol{d}_k).$$

Pentru aceasta procedăm în acelaşi mod ca în exemplul A.2.1 şi consideră funcția

$$\varphi_k(\eta)=f\left(\boldsymbol{x}_k+\eta\boldsymbol{d}_k\right)=\tfrac{1}{2}\boldsymbol{d}_k^T\boldsymbol{A}\boldsymbol{d}_k\eta^2+\boldsymbol{g}_k^T\boldsymbol{d}_k\eta+f\left(\boldsymbol{x}_k\right),$$

unde $\boldsymbol{g}_k=\nabla f(\boldsymbol{x}_k)=\boldsymbol{A}\boldsymbol{x}_k+\boldsymbol{b}$. Funcția $\varphi_k(\eta)$ are gradul 2 în η şi coeficientul termenului dominant pozitiv ($\tfrac{1}{2}\boldsymbol{d}_k^T\boldsymbol{A}\boldsymbol{d}_k>0$ datorită faptului că matricea $\boldsymbol{A}$ este pozitiv definită), astfel că admite minim în

$$\eta_k=-\frac{\boldsymbol{g}_k^T\boldsymbol{d}_k}{\boldsymbol{d}_k^T\boldsymbol{A}\boldsymbol{d}_k}. \tag{A.3.5}$$

Astfel, la iterația k a algoritmului direcțiilor conjugate se calculează punctul

$$\boldsymbol{x}_{k+1}=\boldsymbol{x}_k-\frac{\boldsymbol{g}_k^T\boldsymbol{d}_k}{\boldsymbol{d}_k^T\boldsymbol{A}\boldsymbol{d}_k}\boldsymbol{d}_k. \tag{A.3.6}$$

Se calculează apoi următoarea direcție de căutare $\boldsymbol{d}_{k+1}$ care se alege astfel încât să fie $\boldsymbol{A}$-conjugată direcțiilor $\boldsymbol{d}_0$, $\boldsymbol{d}_1$, …, $\boldsymbol{d}_k$, adică:

$$\boldsymbol{d}_{k+1}^T\boldsymbol{A}\boldsymbol{d}_j=0,\ j=0,1,...,k.$$

Arătăm în continuare că pornind dintr-un punct oarecare $\boldsymbol{x}_0$, se ajunge în punctul de minim în cel mult n iterații, adică $\boldsymbol{x}_n=\boldsymbol{x}^*$.

Deoarece direcțiile $\boldsymbol{A}$-conjugate $\boldsymbol{d}_0,\boldsymbol{d}_1,\ldots,\boldsymbol{d}_{n-1}$, sunt liniar independente, formează o bază în $\mathbb{R}^n$, în care vectorul $\boldsymbol{x}^*-\boldsymbol{x}_0$ admite descompunerea

$$\boldsymbol{x}^*-\boldsymbol{x}_0=\alpha_0\boldsymbol{d}_0+\alpha_1\boldsymbol{d}_1+\ldots+\alpha_{n-1}\boldsymbol{d}_{n-1} \tag{A.3.7}$$

Deoarece $\boldsymbol{d}_k^T\boldsymbol{A}\boldsymbol{d}_j=0$ pentru $j\neq k$, înmulțind la dreapta relația (A.3.3) cu $\boldsymbol{d}_k^T\boldsymbol{A}$ rezultă

$$\boldsymbol{d}_k^T\boldsymbol{A}\left(\boldsymbol{x}^*-\boldsymbol{x}_0\right)=\alpha_k\boldsymbol{d}_k^T\boldsymbol{A}\boldsymbol{d}_k. \tag{A.3.8}$$

Pe de altă parte, din relația de recurență (A.3.2) scrisă în forma $\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \eta_{k-1}\boldsymbol{d}_{k-1}$, se obține

$$\boldsymbol{x}_k = \boldsymbol{x}_0 + \eta_0\boldsymbol{d}_0 + \ldots + \eta_{k-1}\boldsymbol{d}_{k-1},\ k = 1,2,\ldots,$$

relație ce arată că vectorul $\boldsymbol{x}_k - \boldsymbol{x}_0$ este conținut în subspațiul liniar generat de vectorii $\boldsymbol{d}_0, \boldsymbol{d}_1, \ldots, \boldsymbol{d}_{k-1}$, astfel că $\boldsymbol{d}_k^T \boldsymbol{A}(\boldsymbol{x}_k - \boldsymbol{x}_0) = 0$. Scriind $\boldsymbol{x}^* - \boldsymbol{x}_0$ sub forma $\boldsymbol{x}^* - \boldsymbol{x}_0 = (\boldsymbol{x}^* - \boldsymbol{x}_k) + (\boldsymbol{x}_k - \boldsymbol{x}_0)$, se obține

$$\boldsymbol{d}_k^T \boldsymbol{A}(\boldsymbol{x}_k - \boldsymbol{x}_0) = \boldsymbol{d}_k^T \boldsymbol{A}(\boldsymbol{x}^* - \boldsymbol{x}_k) = -\boldsymbol{d}_k^T(\boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{b}) = -\boldsymbol{d}_k^T \boldsymbol{g}_k \qquad \text{(A.3.9)}$$

deoarece $\boldsymbol{A}\boldsymbol{x}^* = -\boldsymbol{b}$ şi $\boldsymbol{g}_k = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{b}$. Din relațiile (A.3.8) şi (A.3.9) rezultă

$$\alpha_k \boldsymbol{d}_k^T \boldsymbol{A}\boldsymbol{d}_k = -\boldsymbol{d}_k^T \boldsymbol{g}_k \Rightarrow \alpha_k = -\frac{\boldsymbol{d}_k^T \boldsymbol{g}_k}{\boldsymbol{d}_k^T \boldsymbol{A}\boldsymbol{d}_k},$$

care, prin comparație cu (A.3.5) revine la $\alpha_k = \eta_k$. Înlocuind în (A.3.7) se obține

$$\boldsymbol{x}^* = \boldsymbol{x}_0 + \eta_0\boldsymbol{d}_0 + \eta_1\boldsymbol{d}_1 + \ldots + \eta_{n-1}\boldsymbol{d}_{n-1},$$

adică $\boldsymbol{x}^* = \boldsymbol{x}_n$. ■

Algoritmul general de minimizare a unei funcții $f : \mathbb{R}^n \to \mathbb{R}$ prin metoda direcțiilor conjugate este prezentat în continuare.

Pe baza rezultatelor prezentate în exemplul A.3.1, algoritmul de minimizare a funcției pătratice (A.3.4)

$$f : \mathbb{R}^n \to \mathbb{R},\ f(\boldsymbol{x}) = \tfrac{1}{2}\boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{x} + c,$$

cu $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ matrice simetrică şi pozitiv definită ($\boldsymbol{A}^T = \boldsymbol{A} > 0$), $\boldsymbol{b} \in \mathbb{R}^n$ şi $c \in \mathbb{R}$, poate fi formulat în modul următor:

Algoritmul QCDM (*metoda direcțiilor conjugate pentru forme pătratice*).

Pas 0. *(Inițializare) Pentru* $k = 0$ *se precizează punctul inițial* $\boldsymbol{x}_0 \in \mathbb{R}^n$ *şi direcțiile* $\boldsymbol{A}$*-conjugate* $\boldsymbol{d}_0, \boldsymbol{d}_1, \ldots, \boldsymbol{d}_{n-1}$.

Pas 1. *(Testare criteriu de stop). Dacă* $k = n$, *treci la Pasul 5.*

Pas 2. *(Determinarea lungimii pasului) Se calculează* $\boldsymbol{g}_k = \nabla f(\boldsymbol{x}_k) = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{b}$ *şi*

$$\eta_k = -\frac{\boldsymbol{g}_k^T \boldsymbol{d}_k}{\boldsymbol{d}_k^T \boldsymbol{A}\boldsymbol{d}_k}.$$

Pas 3. *(Actualizare) Se calculează următorul punct*

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \eta_k \boldsymbol{d}_k.$$

Pas 4. *(Ciclare) Se setează* $k := k+1$ *şi se trece la Pasul 1.*

Pas 5. *(Finalizare) Se returnează punctul final* $\boldsymbol{x}_f = \boldsymbol{x}_n$.

Aplicând algoritmul direcțiilor conjugate pentru minimizarea unei funcții pătratice este satisfăcută următoarea proprietate:

$$\boldsymbol{g}_{k+1}^T \boldsymbol{d}_i = 0, \tag{A.3.10}$$

pentru orice $0 \le k \le n-1$ şi $0 \le i \le k$. Această relație arată că $\boldsymbol{g}_{k+1}$ este ortogonal pe subspațiul generat de vectorii $\boldsymbol{d}_0, \boldsymbol{d}_1, \ldots, \boldsymbol{d}_k$. În plus, in afara faptului că $f(\boldsymbol{x}_{k+1}) = \min_{\eta} f(\boldsymbol{x}_k + \eta \boldsymbol{d}_k)$ (relație utilizată pentru determinarea punctului $\boldsymbol{x}_{k+1}$) se poate demonstra că

$$f(\boldsymbol{x}_{k+1}) = \min_{\alpha_0,\ldots,\alpha_k} f\left(\boldsymbol{x}_0 + \sum_{i=0}^{k} \alpha_i \boldsymbol{d}_i\right).$$

A.3.2. Metoda gradientului conjugat

În algoritmul direcțiilor conjugate prezentat în secțiunea precedentă nu este precizată nici o regulă pentru alegerea direcțiilor conjugate, acestea putând fi specificate din start. Singura condiție impusă este aceea ca direcțiile să fie conjugate în raport cu o matrice dată.

Spre deosebire de acesta, în *algoritmul gradientului conjugat* direcțiile de căutare sunt calculate pe măsură ce algoritmul progresează. La fiecare pas, direcția de căutare este considerată ca o combinație liniară a direcțiilor precedente de căutare şi a gradientului în punctul curent astfel încât toate direcțiile să fie conjugate. Metoda gradientului conjugat a fost propusă în anii 1950 de Hestenes şi Stiefel pentru rezolvarea sistemelor lineare şi extinsă apoi în anii 1960 de Fletcher şi Reeves la cazul minimizării unei funcții generale. În prezent metodele de gradient conjugat sunt utilizate pentru rezolvarea problemelor de optimizare de dimensiuni mari.

Exemplul A.3.2. Ilustrăm în continuare aplicarea algoritmului gradientului conjugat pentru minimizarea formei pătratice

$$f: \mathbb{R}^n \to \mathbb{R},\ f(\boldsymbol{x}) = \tfrac{1}{2} \boldsymbol{x}^T \boldsymbol{A}\, \boldsymbol{x} + \boldsymbol{b}^T \boldsymbol{x} + c, \tag{A.3.11}$$

în care matricea $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ este simetrică şi pozitiv definită ($\boldsymbol{A}^T = \boldsymbol{A} > 0$), $\boldsymbol{b} \in \mathbb{R}^n$ şi $c \in \mathbb{R}$, utilizată şi în exemplele A.2.1, A.2.2 şi A.3.1.

Pentru $k = 0$, se consideră punctul inițial $\boldsymbol{x}_0$ şi se alege direcția $\boldsymbol{d}_0$ de descreştere maximă a lui f, adică

$$\boldsymbol{d}_0 = -\boldsymbol{g}_0,$$

unde $\boldsymbol{g}_0 = \nabla f(\boldsymbol{x}_0) = \boldsymbol{A}\, \boldsymbol{x}_0 + \boldsymbol{b}$.

Ținând cont de cele prezentate în exemplul A.3.1, se calculează

$$\eta_0 = \arg\min_{\eta} f(\boldsymbol{x}_0 + \eta \boldsymbol{d}_0) = -\frac{\boldsymbol{g}_0^T \boldsymbol{d}_0}{\boldsymbol{d}_0^T \boldsymbol{A}\, \boldsymbol{d}_0}$$

şi se consideră

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 + \eta_0 \boldsymbol{d}_0.$$

La pasul următor alegem o direcție $\boldsymbol{d}_1$, combinație liniară între $\boldsymbol{d}_0$ şi $\boldsymbol{g}_1 = \nabla f(\boldsymbol{x}_1)$, de forma $\boldsymbol{d}_1 = -\boldsymbol{g}_1 + \beta_0 \boldsymbol{d}_0$, care să fie $\boldsymbol{A}$-conjugată pentru $\boldsymbol{d}_0$, adică $\boldsymbol{d}_1^T \boldsymbol{A}\, \boldsymbol{d}_0 = 0$.

În general, la iterația k alegem direcția de căutare $\boldsymbol{d}_{k+1}$, ca o combinație liniară între $\boldsymbol{d}_k$ și $\boldsymbol{g}_{k+1}$ de forma

$$\boldsymbol{d}_{k+1} = -\boldsymbol{g}_{k+1} + \beta_k \boldsymbol{d}_k, \quad k = 0,1,2,\ldots \tag{A.3.12}$$

Coeficientul β_k este calculat astfel încât $\boldsymbol{d}_{k+1}$ să fie $\boldsymbol{A}$-conjugată direcției $\boldsymbol{d}_k$. Aceasta conduce la

$$\boldsymbol{d}_{k+1}^T \boldsymbol{A}\boldsymbol{d}_k = -\boldsymbol{g}_{k+1}^T \boldsymbol{A}\boldsymbol{d}_k + \beta_k \boldsymbol{d}_k^T \boldsymbol{A}\boldsymbol{d}_k = 0,$$

de unde rezultă

$$\beta_k = \frac{\boldsymbol{g}_{k+1}^T \boldsymbol{A}\boldsymbol{d}_k}{\boldsymbol{d}_k^T \boldsymbol{A}\boldsymbol{d}_k}. \tag{A.3.13}$$

Apoi se calculează noul punct utilizând relația

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \eta_k \boldsymbol{d}_k.$$

Se poate demonstra că direcțiile $\boldsymbol{d}_0, \boldsymbol{d}_1, \ldots, \boldsymbol{d}_{n-1}$, obținute în modul prezentat mai sus sunt $\boldsymbol{A}$-conjugate. În primul rând direcția $\boldsymbol{d}_1$ este selectată astfel încât să fie $\boldsymbol{A}$-conjugată direcției $\boldsymbol{d}_0$. În continuare, presupunem că direcțiile $\boldsymbol{d}_0, \boldsymbol{d}_1, \ldots, \boldsymbol{d}_k$, $k < n-1$, sunt $\boldsymbol{A}$-conjugate. Din relația (A.3.12) rezultă $\boldsymbol{g}_{k+1}^T \boldsymbol{d}_i = 0$, pentru orice $0 \le i \le k$. Cum fiecare $\boldsymbol{d}_i$ este ales de forma

$$\boldsymbol{d}_i = -\boldsymbol{g}_i + \beta_{i-1} \boldsymbol{d}_{i-1}, \quad i = 1,2,\ldots,k,$$

rezultă că

$$\boldsymbol{g}_{k+1}^T \boldsymbol{d}_i = -\boldsymbol{g}_{k+1}^T \boldsymbol{g}_i + \beta_{i-1} \boldsymbol{g}_{k+1}^T \boldsymbol{d}_{i-1},$$

astfel că

$$\boldsymbol{g}_{k+1}^T \boldsymbol{g}_i = 0, \; i = 1,2,\ldots k.$$

Cum $\boldsymbol{d}_0 = -\boldsymbol{g}_0$, are loc și $\boldsymbol{g}_{k+1}^T \boldsymbol{g}_0 = 0$.

În continuare demonstrăm că

$$\boldsymbol{d}_{k+1}^T \boldsymbol{A}\boldsymbol{d}_i = 0, \; i = 0,1,\ldots k-1.$$

Ținând cont de (A.3.12), avem $\boldsymbol{d}_{k+1}^T \boldsymbol{A}\boldsymbol{d}_i = \left(-\boldsymbol{g}_{k+1} + \beta_k \boldsymbol{d}_k\right)^T \boldsymbol{A}\boldsymbol{d}_i$.

În ipoteza considerată, $\boldsymbol{d}_k^T \boldsymbol{A}\boldsymbol{d}_i = 0$, $i = 0,1,\ldots k$, de unde rezultă $\boldsymbol{d}_{k+1}^T \boldsymbol{A}\boldsymbol{d}_i = -\boldsymbol{g}_{k+1}^T \boldsymbol{A}\boldsymbol{d}_i$, $i = 0,1,\ldots,k-1$. Dar $\boldsymbol{g}_{i+1} = \boldsymbol{A}\boldsymbol{x}_{i+1} + \boldsymbol{b} = \boldsymbol{A}(\boldsymbol{x}_i + \eta_i \boldsymbol{d}_i) + \boldsymbol{b} = \boldsymbol{g}_i + \eta_i \boldsymbol{A}\boldsymbol{d}_i$, astfel că $\boldsymbol{A}\boldsymbol{d}_i = \frac{1}{\eta_i}(\boldsymbol{g}_{i+1} - \boldsymbol{g}_i)$. Cum $\boldsymbol{g}_{k+1}^T \boldsymbol{g}_i = 0$, $i = 1,2,\ldots k$, rezultă în continuare că $\boldsymbol{d}_{k+1}^T \boldsymbol{A}\boldsymbol{d}_i = 0$ pentru $i = 0,1,\ldots,k-1$. Pe de altă parte, $\boldsymbol{d}_{k+1}$ a fost calculată astfel încât să fie $\boldsymbol{A}$-conjugată direcției $\boldsymbol{d}_k$, deci este satisfăcută și relația $\boldsymbol{d}_{k+1}^T \boldsymbol{A}\boldsymbol{d}_k = 0$.

Prin inducție matematică am demonstrat că dacă direcțiile $\boldsymbol{d}_0, \boldsymbol{d}_1, \ldots, \boldsymbol{d}_k$, $k < n-1$, sunt $\boldsymbol{A}$-conjugate, atunci direcția $\boldsymbol{d}_{k+1}$ este $\boldsymbol{A}$-conjugată acestora. Rezultă în final că direcțiile $\boldsymbol{d}_0, \boldsymbol{d}_1, \ldots, \boldsymbol{d}_{n-1}$ sunt $\boldsymbol{A}$-conjugate.

Pe baza celor prezentate pentru metoda generală a direcțiilor conjugate, algoritmul gradientului conjugat pentru minimizarea unei funcții pătratice se va termina în cel mult *n* iterații. Mai exact, se poate demonstra că algoritmul gradientului conjugat se va termina în cel mult *m* iterații, unde *m* reprezintă numărul de valori proprii distincte ale matricei $\boldsymbol{A}$. ■

În continuare poate fi formulat algoritmul de minimizare a unei forme pătratice $f:\mathbb{R}^n \to \mathbb{R}$, $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T \boldsymbol{A}\,\boldsymbol{x} + \boldsymbol{b}^T \boldsymbol{x} + c$, cu $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, $\boldsymbol{A}^T = \boldsymbol{A} > 0$, $\boldsymbol{b} \in \mathbb{R}^n$ şi $c \in \mathbb{R}$, prin metoda gradientului conjugat.

Algoritmul QCGM (*metoda gradientului conjugat pentru forme pătratice*).

Pas 0. *(Inițializare) Pentru* $k = 0$ *se precizează punctul inițial* $\boldsymbol{x}_0 \in \mathbb{R}^n$.

Pas 1. *(Testare criteriu de stop). Se calculează* $\boldsymbol{g}_0 = \nabla f(\boldsymbol{x}_0) = \boldsymbol{A}\,\boldsymbol{x}_0 + \boldsymbol{b}$.

Dacă $\boldsymbol{g}_0 = 0$, *atunci se returnează* $\boldsymbol{x}_f = \boldsymbol{x}_0$. *STOP.*

Altfel, se setează $\boldsymbol{d}_0 = -\boldsymbol{g}_0$.

Pas 2. *(Determinarea lungimii pasului) Se calculează*

$$\eta_k = -\frac{\boldsymbol{g}_k^T \boldsymbol{d}_k}{\boldsymbol{d}_k^T \boldsymbol{A}\,\boldsymbol{d}_k}.$$

Pas 3. *(Actualizare) Se calculează următorul punct*

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \eta_k \boldsymbol{d}_k .$$

Pas 4. *Se calculează* $\boldsymbol{g}_{k+1} = \nabla f(\boldsymbol{x}_{k+1}) = \boldsymbol{A}\,\boldsymbol{x}_{k+1} + \boldsymbol{b}$.

Dacă $\boldsymbol{g}_{k+1} = 0$, *atunci se returnează* $\boldsymbol{x}_f = \boldsymbol{x}_{k+1}$. *STOP.*

Pas 5. *(Determinarea următoarei direcții de căutare) Se calculează*

$$\beta_k = \frac{\boldsymbol{g}_{k+1}^T \boldsymbol{A}\,\boldsymbol{d}_k}{\boldsymbol{d}_k^T \boldsymbol{A}\,\boldsymbol{d}_k}$$

$$\boldsymbol{d}_{k+1} = -\boldsymbol{g}_{k+1} + \beta_k \boldsymbol{d}_k$$

Pas 6. *(Ciclare) Se setează* $k := k+1$ *şi se trece la Pasul 2.*

Pentru minimizarea unei funcții care nu este pătratică prin metoda gradientului conjugat se porneşte de la ideea aproximării funcției de minimizat prin termenii de ordin cel mult 2 din dezvoltarea acesteia în serie Taylor în jurul punctului curent. Pentru a evita calcularea matricei Hessian a funcției respective (care, pentru o funcție pătratică este constantă) la fiecare iterație, trebuie modificate formulele care precizează valorile lui η_k şi β_k astfel încât să nu mai apară matricea hessian.

Pentru calcularea lui η_k se poate utiliza o metodă de căutare liniară care să permită evaluarea

$$\eta_k = \arg\min_{\eta} f(\boldsymbol{x}_k + \eta \boldsymbol{d}_k).$$

În ceea ce priveşte calcularea factorului β_k, relația (A.3.13), este modificată astfel încât să nu mai apară matricea $\boldsymbol{A}$. Printre formulele cele mai frecvent utilizate sunt:

Fletcher-Reeves: $$\beta_k = \frac{\boldsymbol{g}_{k+1}^T \boldsymbol{g}_{k+1}}{\boldsymbol{g}_k^T \boldsymbol{g}_k} \tag{A.3.14}$$

Polak-Ribiere-Polyak: $$\beta_k = \frac{\boldsymbol{g}_{k+1}^T (\boldsymbol{g}_{k+1} - \boldsymbol{g}_k)}{\boldsymbol{g}_k^T \boldsymbol{g}_k} \tag{A.3.15}$$

Dixon: $$\beta_k = \frac{\boldsymbol{g}_{k+1}^T \boldsymbol{g}_{k+1}}{\boldsymbol{d}_k^T \boldsymbol{g}_k} \tag{A.3.16}$$

Dai-Yuan: $$\beta_k = \frac{\boldsymbol{g}_{k+1}^T \boldsymbol{g}_{k+1}}{\boldsymbol{d}_k^T (\boldsymbol{g}_{k+1} - \boldsymbol{g}_k)} \tag{A.3.17}$$

Din punct de vedere istoric, prima formulă utilizată în acest sens este formula Hestenes-Stiefel:

$$\beta_k = \frac{\boldsymbol{g}_{k+1}^T (\boldsymbol{g}_{k+1} - \boldsymbol{g}_k)}{\boldsymbol{d}_k^T (\boldsymbol{g}_{k+1} - \boldsymbol{g}_k)}. \tag{A.3.18}$$

Toate aceste formule sunt echivalente între ele în cazul în care funcția minimizată este pătratică.

Aplicarea acestor formule prezintă avantajul că nu necesită explicitarea matricei hessian la fiecare iterație, fiind nevoie numai de valorile gradientului. Pentru probleme nepătratice însă, algoritmul nu mai converge în *n* paşi, iar proprietatea de „conjugare“ a direcțiilor de căutare tinde să se piardă. De aceea o tehnică uzuală este de a reinițializa direcția de căutare cu direcția de descreştere maximă (opusă gradientului) la fiecare *n* (sau *n*+1) iterații. Algoritmul de va opri atunci când unul din criteriile de stop este satisfăcut.

Pentru simplitate prezentăm în continuare algoritmul Fletcher-Reeves cu reinițializare pentru minimizarea unei funcții $f : \mathbb{R}^n \to \mathbb{R}$.

Algoritmul RFRCGM (*Restart Fletcher-Reeves Conjugate Gradient Method*).

Pas 0. *(Inițializare) Pentru* $k = 0$ *se precizează punctul inițial* $\boldsymbol{x}_0 \in \mathbb{R}^n$, *valoarea de prag* $\varepsilon > 0$ *şi numărul maxim de iterații N. Se calculează* $\boldsymbol{g}_0 = \nabla f(\boldsymbol{x}_0) = \boldsymbol{A}$ *şi se setează* $\boldsymbol{d}_0 = -\boldsymbol{g}_0$.

Pas 1. *(Testare criterii de stop)*
1.1 Dacă $\|\boldsymbol{g}_k\| < \varepsilon$, *treci la Pasul 8.*
1.2. Dacă $k = N$, *treci la Pasul 8.*

Pas 2. *(Determinarea lungimii pasului) Se determină factorul* η_k *astfel încât* $f(\boldsymbol{x}_k + \eta_k \boldsymbol{d}_k) = \min_{\eta} f(\boldsymbol{x}_k + \eta \boldsymbol{d}_k)$.

Pas 3. *(Actualizare) Se calculează următorul punct* $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \eta_k \boldsymbol{d}_k$ şi $\boldsymbol{g}_{k+1} = \nabla f(\boldsymbol{x}_{k+1})$.

Pas 4. *Dacă* $k \equiv 0 \pmod{n}$ *atunci* $\beta_k = 0$,

altfel $\beta_k = \dfrac{\boldsymbol{g}_{k+1}^T \boldsymbol{g}_{k+1}}{\boldsymbol{g}_k^T \boldsymbol{g}_k}$.

Pas 5. *(Determinarea următoarei direcții de căutare) Se calculează*

$\boldsymbol{d}_{k+1} = -\boldsymbol{g}_{k+1} + \beta_k \boldsymbol{d}_k$

Pas 6. *Dacă* $\boldsymbol{d}_{k+1}^T \boldsymbol{g}_{k+1} > 0$, *atunci* $\boldsymbol{d}_{k+1} = -\boldsymbol{g}_{k+1}$.

Pas 7. *(Ciclare) Se setează* $k := k+1$ *şi se trece la Pasul 1.*

Pas 8. *(Finalizare) Se returnează punctul final* $\boldsymbol{x}_f = \boldsymbol{x}_k$.

Anexa B

PROBLEMATICA CLASIFICĂRII FORMELOR

B.1. Tipuri de clasificatori

Cum neuronul artificial a fost conceput pentru a modela (imita) neuronul biologic, rețelelor neurale artificiale le-au fost atribuite diverse sarcini specifice sistemului nervos.
Dintre acestea, operația de *recunoaştere a formelor* (eng. *pattern recognition*) sau *clasificare* (eng. *classification*) este caracteristică tuturor ființelor vii. Natura fizică a obiectelor clasificate poate fi foarte diversă (imagini, sunete, semnale electrice, etc) astfel încât recunoaşterea formelor a devenit parte integrantă a sistemelor de decizie bazate pe inteligența artificială. În prezent, recunoaşterea formelor are aplicații în domenii variate precum: vedere artificială, recunoaşterea optică a caracterelor (litere sau cifre) (eng. *Optical Character Recognition – OCR*), recunoaşterea sunetelor, diagnosticarea pacienților cu ajutorul computerului (eng. *computer aided diagnosis*) pe baza procesării unor date medicale variate (imagini obținute prin expunerea la raze X, tomografii, EKG, EEG), recunoaşterea vorbirii (comanda verbală a sistemelor), procesarea de imagini în astronomie, geologie, aplicații militare (recunoaşterea automată a țintelor).

Pentru a determina clasa căreia îi aparține un anumit obiect sunt evaluate mai întâi anumite *trăsături* (eng. *feature*) caracteristice ale acestuia; decizia privind clasificarea este luată apoi în funcție de valorile trăsăturilor. În general, un obiect poate fi caracterizat prin mai multe trăsături. În funcție de scopul clasificării, în această operație sunt utilizate numai unele dintre aceste trăsături, efectuându-se mai întâi operația de *extragere a trăsăturilor* (eng. *feature extraction*) (selectare a măsurătorilor relevante, evaluarea unor noi trăsături care vor fi utilizate efectiv în clasificare pe baza măsurătorilor existente).

Un concept de bază în problemele de clasificare a formelor este cel de *cluster* (eng. *cluster*). Un cluster constă dintr-un număr de obiecte (forme) similare care sunt grupate împreună. Operația de clasificare înseamnă de fapt împărțirea formelor în clustere şi stabilirea pentru fiecare cluster în parte a centrului şi a frontierei sale.

Cunoaşterea numărului de clustere şi a localizării lor (măcar aproximative) în etapa de proiectare a clasificatorului simplifică operația de clasificare. În acest caz se poate utiliza o procedură de *antrenare supervizată* a clasificatorului care presupune existența unui număr de *eşantioane* (*prototipuri* – eng. *patterns*) pentru care sunt cunoscute clasele cărora le aparțin. Clasificatorul trebuie, în primul rând, învățat să clasifice în mod corect aceste eşantioane. Apoi, la prezentarea unei noi forme, clasificatorul o va atribui uneia dintre clasele învățate, în funcție de trăsăturile sale.

Dacă asemenea cunoştințe nu sunt disponibile, se aplică un procedeu de *antrenare nesupervizată* a clasificatorului, care, în plus față de antrenarea supervizată, permite şi determinarea numărului de clase în care pot fi clasificate eşantioanele considerate.

Următoarele metode de clasificare sunt frecvent utilizate în aplicații practice:

- **Funcțiile de decizie** (eng. *decision functions*) pot fi utilizate dacă numărul de clase este cunoscut apriori şi există o separare geometrică între clase. De exemplu , pentru separarea a două clase $\mathscr{C}_1$ şi $\mathscr{C}_2$ in $\mathbb{R}^n$ considerăm o funcție $d:\mathbb{R}^n \to \mathbb{R}$ şi clasificăm o forma $\boldsymbol{x} \in \mathbb{R}^n$ după semnul funcției d: dacă $d(x)>0$ atunci $x \in \mathscr{C}_1$ şi dacă $d(x)<0$ atunci $x \in \mathscr{C}_2$. Suprafața din $\mathbb{R}^n$ a cărei ecuație este $d(\boldsymbol{x})=0$ reprezintă *frontiera de decizie* (eng. *decision boundary*).
- Clasificarea bazată pe **distanța minimă** (eng. *minimum distance*) poate fi aplicată în cazul în care fiecare clasă poate fi reprezentată printr-un singur prototip, denumit *centrul* clusterului. Clasificatorul bazat pe distanța minimă (pe cel mai apropiat vecin – eng. *nearest neighbor*) clasifică o nouă formă măsurând distanțele de la aceasta la centrele clusterelor considerate şi atribuind forma respectivă celei mai apropiate clase. Această metodă a fost extinsă la cazul în care este cunoscut numai numărul de clase, dar nu se cunoaşte modul de clasificare a eşantioanelor din setul de antrenare. Centrele clusterelor pot fi găsite iterativ prin minimizarea unui criteriu de performanță (de exemplu algoritmii C-Means sau ISODATA).
- În situația în care există posibilitatea de suprapunere a unor clase, problema clasificării poate fi abordată **statistic**. Clasificatorul determină probabilitatea de apartenență a unei forme la fiecare din clasele considerate. De asemenea un clasificator statistic trebuie sa evalueze gradul de risc asociat clasificării (probabilitatea de clasificare eronată a unei forme).
- **Rețele neurale** reprezintă un instrument frecvent utilizat în proiectarea clasificatorilor. Există atât metode de antrenare supervizată, cât şi metode de antrenare nesupervizată, care permit ajustarea valorilor ponderilor şi deplasărilor neuronilor până se obține o clasificare corectă (este minimizat un indicator de performanță).

- **Logica fuzzy** poate fi utilizată dacă există incertitudinii asupra modului de clasificare a prototipurilor (formele din setul de antrenare) sau o anumită formă poate aparține într-o măsură oarecare mai multor clase. Clasificatorul fuzzy determină în final gradul de apartenență a unei forme la fiecare clasă considerată.

B.2. Clasificarea pe baza funcțiilor de decizie

B.2.1. Separarea în două clase

Pentru simplitate, considerăm situația în care se dorește separarea în două clase , notate $\mathscr{C}_1$ și $\mathscr{C}_2$, a unor forme caracterizate prin două trăsături. Presupunem că fiecare eșantion poate fi reprezentat printr-un vector bidimensional $\boldsymbol{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T \in \mathbb{R}^2$ și reprezentat grafic în planul x_1Ox_2 prin simbolul ∘ (dacă $x \in \mathscr{C}_1$) sau prin □ (dacă $x \in \mathscr{C}_2$).

În cazul prezentat în figura B.2.1, cele două clase pot fi separate printr-o dreaptă, a cărei ecuație este:

$$d(\boldsymbol{x}) = w_1 x_1 + w_2 x_2 + b = 0 . \tag{B.2.1}$$

Semnele coeficienților $w_1, w_2, b \in \mathbb{R}$ pot fi alese astfel încât să fie satisfăcute condițiile:

$$\begin{cases} d(\boldsymbol{x}) > 0, & \forall \boldsymbol{x} \in \mathscr{C}_1, \\ d(\boldsymbol{x}) < 0, & \forall \boldsymbol{x} \in \mathscr{C}_2. \end{cases}$$

Pentru a clasifica o nouă formă $\tilde{\boldsymbol{x}} \in \mathbb{R}^2$ despre care se știe că aparține la una și numai una din clasele $\mathscr{C}_1$ și $\mathscr{C}_2$, se calculează valoarea funcției d în $\tilde{\boldsymbol{x}}$ și decizia asupra clasificării lui $\tilde{\boldsymbol{x}}$ se ia în funcție de semnul lui $d(\tilde{\boldsymbol{x}})$ astfel:

$$\begin{aligned} &\tilde{\boldsymbol{x}} \in \mathscr{C}_1, \text{ dacă } d(\tilde{\boldsymbol{x}}) > 0, \\ &\tilde{\boldsymbol{x}} \in \mathscr{C}_2, \text{ dacă } d(\tilde{\boldsymbol{x}}) < 0. \end{aligned} \tag{B.2.2}$$

Funcția $d : \mathbb{R}^2 \to \mathbb{R}$, $d(\boldsymbol{x}) = w_1 x_1 + w_2 x_2 + b$, pe baza căreia se realizează clasificarea reprezintă o *funcție de decizie liniară*.

Acest concept poate fi extins la cazul general al clasificării unor forme caracterizate printr-un număr oarecare de trăsături. Presupunem că forme *n*-dimensionale reprezentate prin vectori $\boldsymbol{x} = \begin{bmatrix} x_1 & x_2 & \ldots & x_n \end{bmatrix}^T \in \mathbb{R}^n$ pot fi separate în două clase $\mathscr{C}_1$ și $\mathscr{C}_2$, utilizând o *funcție de decizie liniară* de forma

$$d : \mathbb{R}^n \to \mathbb{R}, \; d(\boldsymbol{x}) = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b = \boldsymbol{W}\boldsymbol{x} + b , \tag{B.2.3}$$

unde $\boldsymbol{W} \triangleq \begin{bmatrix} w_1 & w_2 & \ldots & w_n \end{bmatrix} \in \mathbb{R}^{1 \times n}$ și $b \in \mathbb{R}$, în conformitate cu (B.2.2). Geometric cele două clase din $\mathbb{R}^n$ pot fi separate prin hiperplanul de ecuație $d(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x} + b = 0$.

Mai mult, funcția liniară (B.2.3) poate fi generalizată în cazul în care are forma unei combinații liniare de funcții

$$d : \mathbb{R}^n \to \mathbb{R}, \; d(\boldsymbol{x}) = w_1 f_1(\boldsymbol{x}) + \ldots + w_p f_p(\boldsymbol{x}) + b , \tag{B.2.4}$$

cu $f_i:\mathbb{R}^n\to\mathbb{R}$, $i=\overline{1,p}$, în general funcții neliniare de n variabile, și $w_1,w_2,\dots w_p,b\in\mathbb{R}$. Cu notațiile $\boldsymbol{W}=\begin{bmatrix}w_1 & \dots, & w_p\end{bmatrix}^{1\times p}$, $\boldsymbol{F}:\mathbb{R}^n\to\mathbb{R}^p$, $\boldsymbol{F}(x)=\begin{bmatrix}f_1(\boldsymbol{x}) & \dots & f_p(\boldsymbol{x})\end{bmatrix}^T$, funcția (B.2.4) poate fi adusă la forma $d(\boldsymbol{x})=\boldsymbol{W}\boldsymbol{F}(\boldsymbol{x})+b$.

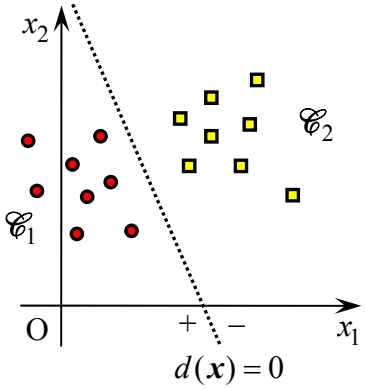


Figura B.2.1. *Două clase liniar separabile.*

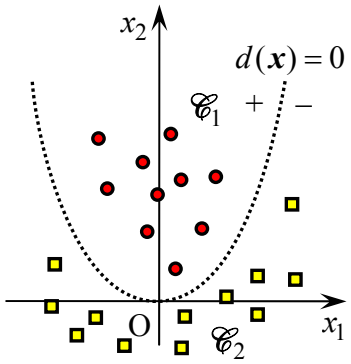


Figura B.2.2. *Două clase neliniar separabile.*

În cazul cel mai general funcția de decizie poate să nu fie liniară. Figura B.2.2 prezintă cazul a două clase din $\mathbb{R}^2$ ce pot fi separate utilizând funcția de decizie $d(\boldsymbol{x})=-x_1^2+x_2$. Reprezentarea grafică a curbei de decizie $d(\boldsymbol{x})=0$ este o parabolă. Clasificarea unei noi forme $\tilde{\boldsymbol{x}}\in\mathbb{R}^2$ se face aplicând (B.2.2).

B.2.2. Separarea în m *clase*

În cazul cel mai general se dorește separarea unui număr m de clase, notate $\mathscr{C}_1$, $\mathscr{C}_2$, …, $\mathscr{C}_m$, de forme din $\mathbb{R}^n$.

Dacă fiecărei clase de forme $\mathscr{C}_i$ îi poate fi atașată o funcție de decizie $d_i:\mathbb{R}^n\to\mathbb{R}$ care realizează separarea dintre clasa $\mathscr{C}_i$ și restul claselor $\mathscr{C}_j$, $j\neq i$, conform

$$\begin{cases} d_i(\boldsymbol{x})>0, & \forall\boldsymbol{x}\in\mathscr{C}_i,\\ d_i(\boldsymbol{x})<0, & \forall\boldsymbol{x}\in\mathscr{C}_j,\ j\neq i,\end{cases} \tag{B.2.5}$$

atunci se spune că $\mathscr{C}_1,\mathscr{C}_2,\dots,\mathscr{C}_m$ sunt *clase absolut separabile* (eng. *absolutely separable*).

Dacă $\mathscr{C}_1$, $\mathscr{C}_2$, …, $\mathscr{C}_m$, sunt clase absolut separabile din $\mathbb{R}^n$ cu ajutorul funcțiilor de decizie d_1, d_2, …, d_m, mulțimea

$$\mathscr{D}_i=\left\{\boldsymbol{x}\in\mathbb{R}^n\middle|\,d_i(\boldsymbol{x})>0,d_j(\boldsymbol{x})<0,j\neq i\right\},\ i=1,2,\dots m, \tag{B.2.6}$$

se numește *regiune de decizie* corespunzătoare clasei $\mathscr{C}_i$. Utilizând criteriul (B.2.5), orice punct din $\mathscr{D}_i$ este clasificat ca aparținând clasei $\mathscr{C}_i$.

Figura B.2.3 prezintă cazul a trei clase $\mathscr{C}_1$, $\mathscr{C}_2$ şi $\mathscr{C}_3$ absolut separabile în $\mathbb{R}^2$ prin funcții de decizie liniare. Regiunile de decizie corespunzătoare sunt prezentate în figura B.2.4. Se observă că regiunile de decizie ale celor trei clase nu acoperă întreg spațiul $\mathbb{R}^2$. O formă $\boldsymbol{x} \in \mathbb{R}^2 \setminus \left(\mathscr{D}_1 \cup \mathscr{D}_2 \cup \mathscr{D}_3 \right)$ nu poate fi clasificată în nici una dintre clasele $\mathscr{C}_1$, $\mathscr{C}_2$, $\mathscr{C}_3$ utilizând funcțiile de decizie considerate.

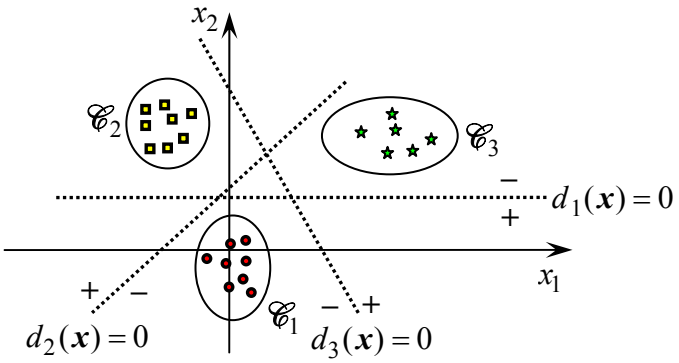


Figura B.2.3. *Trei clase absolut separabile în* $\mathbb{R}^2$ *prin funcții de decizie liniare.*

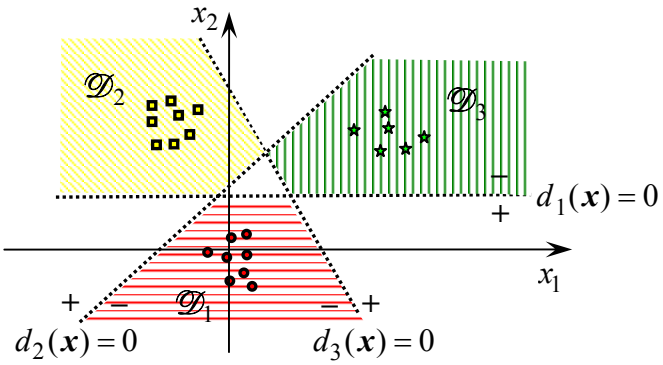


Figura B.2.4. *Regiunile de decizie corespunzătoare claselor din figura B.2.3.*

Fie clasele $\mathscr{C}_1$, $\mathscr{C}_2$, …, $\mathscr{C}_m$ de forme din $\mathbb{R}^n$, care nu sunt absolut separabile. Dacă fiecărei perechi de clase $\mathscr{C}_i$, $\mathscr{C}_j$, $j \neq i$, îi poate fi asociată o funcție $d_{ij} : \mathbb{R}^n \to \mathbb{R}$ astfel încât

$$\begin{cases} d_{ij}(\boldsymbol{x}) > 0, & \forall \boldsymbol{x} \in \mathscr{C}_i, \\ d_{ij}(\boldsymbol{x}) < 0, & \forall \boldsymbol{x} \in \mathscr{C}_j, \end{cases} \tag{B.2.7}$$

se spune că $\mathscr{C}_1$, $\mathscr{C}_2$, …, $\mathscr{C}_m$ sunt clase *separabile două câte două* (eng. *pairwise separable*).

Dacă clasele $\mathscr{C}_1$, $\mathscr{C}_2$, …, $\mathscr{C}_m$ sunt separabile două câte două, atunci pentru orice $j \neq i$ are loc $d_{ij}(\boldsymbol{x}) > 0$, $\forall \boldsymbol{x} \in \mathscr{C}_i$. De asemenea, funcțiile de decizie pot fi considerate astfel încât $d_{ji}(\boldsymbol{x}) = -d_{ij}(\boldsymbol{x})$, $\forall \boldsymbol{x} \in \mathbb{R}^n$.

Dacă $\mathscr{C}_1$, $\mathscr{C}_2$, …, $\mathscr{C}_m$, sunt clase din $\mathbb{R}^n$ separabile două câte două cu ajutorul funcțiilor de decizie d_{ij}, mulțimea

$$\mathscr{D}_i = \left\{ \boldsymbol{x} \in \mathbb{R}^n \,\middle|\, d_{ij}(\boldsymbol{x}) > 0, j \neq i \right\}, \tag{B.2.8}$$

se numește *regiune de decizie* corespunzătoare clasei $\mathscr{C}_i$, $i = 1,2,\ldots m$.

Figura B.2.5 prezintă cazul a trei clase $\mathscr{C}_1$, $\mathscr{C}_2$, $\mathscr{C}_3$ din $\mathbb{R}^2$ separabile două câte două prin funcții de decizie liniare. Așa cum se observă în figura B.2.6, regiunile de decizie ale celor trei clase nu acoperă întreg spațiul $\mathbb{R}^2$. O formă $\boldsymbol{x} \in \mathbb{R}^2 \setminus \left(\mathscr{D}_1 \cup \mathscr{D}_2 \cup \mathscr{D}_3 \right)$ nu poate fi clasificată în nici una dintre clasele $\mathscr{C}_1$, $\mathscr{C}_2$, $\mathscr{C}_3$ utilizând funcțiile de decizie considerate.

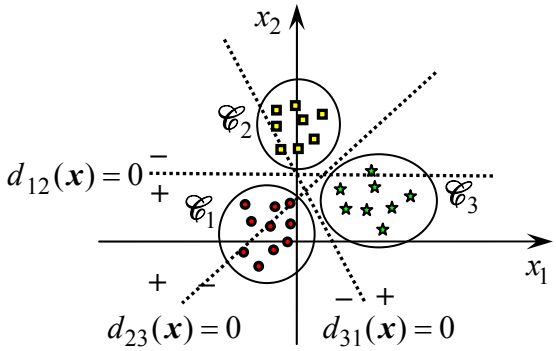


Figura B.2.5. *Trei clase separabile două câte două prin funcții de decizie liniare.*

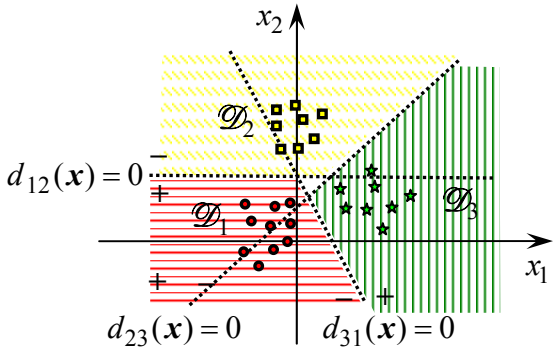


Figura B.2.6. *Regiunile de decizie corespunzătoare claselor din fig. B.2.5.*

B.3. Clasificarea pe baza distanței

B.3.1. Clase reprezentate printr-un singur prototip

Considerăm un număr *m* de clase, notate $\mathscr{C}_1$, $\mathscr{C}_2$, …, $\mathscr{C}_m$, de forme din $\mathbb{R}^n$. Presupunem mai întâi că fiecare clasă $\mathscr{C}_i$ este reprezentată printr-un unic prototip $\boldsymbol{y}_i = \left[y_{i1} \;\; y_{i2} \;\; \ldots \;\; y_{in} \right]^T \in \mathbb{R}^n$, $i = \overline{1,m}$. Distanța euclidiană de la o formă $\boldsymbol{x} \in \mathbb{R}^n$ la vectorul prototip $\boldsymbol{y}_i$ este

$$D_i = \|\boldsymbol{x} - \boldsymbol{y}_i\|_2 = \left[(\boldsymbol{x} - \boldsymbol{y}_i)^T \cdot (\boldsymbol{x} - \boldsymbol{y}_i) \right]^{1/2},\ i = \overline{1,m}. \tag{B.3.1}$$

Un clasificator bazat pe distanța euclidiană atribuie forma $\boldsymbol{x} \in \mathbb{R}^n$ clasei $\mathscr{C}_j$ (sau echivalent prototipul $\boldsymbol{y}_j$) situată la distanță minimă de $\boldsymbol{x}$, adică

$$\boldsymbol{x} \in \mathscr{C}_j \Leftrightarrow D_j = \min_{1 \le i \le m} D_i = \min_{1 \le i \le m} \|\boldsymbol{x} - \boldsymbol{y}_i\|_2. \tag{B.3.2}$$

Dacă valoarea minimă este atinsă pentru mai mulți indici j simultan, atunci forma $\boldsymbol{x}$ este fie clasificată în clasa corespunzătoare celui mai mic indice, fie nu este clasificată deloc.

O modalitate de clasificare echivalentă cu (B.3.2) este de a utiliza pătratele distanțelor D_i^2 în loc de D_i. Astfel o formă $\boldsymbol{x} \in \mathbb{R}^n$ este atribuită clasei C_j în modul următor :

$$\boldsymbol{x} \in \mathscr{C}_j \Leftrightarrow D_j^2 = \min_{1 \le i \le m} D_i^2 = \min_{1 \le i \le m} \|\boldsymbol{x} - \boldsymbol{y}_i\|_2^2. \tag{B.3.3}$$

Datorită faptului că

$$D_i^2 = \|\boldsymbol{x} - \boldsymbol{y}_i\|_2^2 = \left[(\boldsymbol{x} - \boldsymbol{y}_i)^T \cdot (\boldsymbol{x} - \boldsymbol{y}_i) \right] = \boldsymbol{x}^T \cdot \boldsymbol{x} - 2\boldsymbol{y}_i^T \cdot \boldsymbol{x} + \boldsymbol{y}_i^T \cdot \boldsymbol{y}_i,$$

expresie în care $\boldsymbol{x}^T \cdot \boldsymbol{x}$ are valoare constantă pentru o formă $\boldsymbol{x}$ dată, relația (B.3.3) devine

$$\boldsymbol{x} \in \mathscr{C}_j \Leftrightarrow j = \arg \min_{1 \le i \le m} \left(-2\boldsymbol{y}_i^T \cdot \boldsymbol{x} + \boldsymbol{y}_i^T \cdot \boldsymbol{y}_i \right) = \arg \max_{1 \le i \le m} \left(\boldsymbol{y}_i^T \cdot \boldsymbol{x} - \frac{1}{2} \boldsymbol{y}_i^T \cdot \boldsymbol{y}_i \right).$$

Astfel se poate utiliza funcția

$$d_i : \mathbb{R}^n \to \mathbb{R},\ d_i(\boldsymbol{x}) = \boldsymbol{y}_i^T \cdot \boldsymbol{x} - \frac{1}{2} \boldsymbol{y}_i^T \cdot \boldsymbol{y}_i,\ i = \overline{1,m}, \tag{B.3.4}$$

drept funcție de decizie, clasificarea făcându-se astfel:

$$\boldsymbol{x} \in \mathscr{C}_j \Leftrightarrow d_j(\boldsymbol{x}) > d_i(\boldsymbol{x}),\ j \ne i. \tag{B.3.5}$$

Se observă că funcțiile de decizie (B.3.4) sunt liniare, putând fi puse sub forma $d_i(\boldsymbol{x}) = \boldsymbol{W}_i \boldsymbol{x} + b_i$ cu $\boldsymbol{W}_i = [w_{i1}\ w_{i2}\ \ldots\ w_{in}] \in \mathbb{R}^{1 \times n}$, $w_{ik} = y_{ik}$, $k = \overline{1,n}$, și $b_i = -\frac{1}{2} \boldsymbol{y}_i^T \cdot \boldsymbol{y}_i$ pentru $i = \overline{1,m}$.

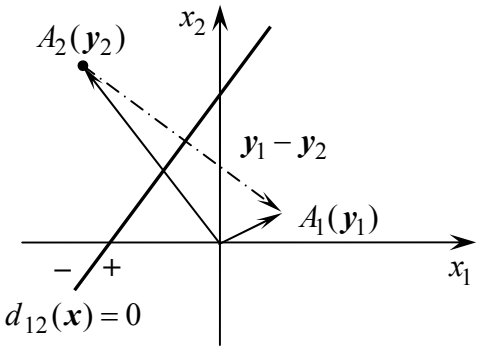


Figura B.3.1. *Frontiera de separare a două clase din* $\mathbb{R}^2$ *pe baza distanței minime.*

În particular, dacă se separă două clase $\mathscr{C}_1$, $\mathscr{C}_2$ în $\mathbb{R}^n$ cărora le corespund prototipurile $\boldsymbol{y}_1$, $\boldsymbol{y}_2$, utilizând funcțiile $d_1(\boldsymbol{x})$ și $d_2(\boldsymbol{x})$ definite conform (B.3.4) se observă că frontiera de decizie are ecuația:

$$d_{12}(\boldsymbol{x}) = d_1(\boldsymbol{x}) - d_2(\boldsymbol{x}) = (\boldsymbol{y}_1 - \boldsymbol{y}_2)^T \cdot \boldsymbol{x} - \frac{1}{2}\left(\boldsymbol{y}_1^T \cdot \boldsymbol{y}_1 + \boldsymbol{y}_2^T \cdot \boldsymbol{y}_2\right) = 0,$$

fiind un hiperplan normal pe vectorul $\boldsymbol{y}_1 - \boldsymbol{y}_2$.

Acest raționament poate fi extins la cazul a *m* clase $\mathscr{C}_1$, $\mathscr{C}_2$, …, $\mathscr{C}_m$, din $\mathbb{R}^n$ ce pot fi separate două câte două cu ajutorul funcțiilor de decizie d_{ij},

$$d_{ij}(\boldsymbol{x}) = d_i(\boldsymbol{x}) - d_j(\boldsymbol{x}) = \left(\boldsymbol{y}_i - \boldsymbol{y}_j\right)^T \cdot \boldsymbol{x} - \frac{1}{2}\left(\boldsymbol{y}_i^T \cdot \boldsymbol{y}_i + \boldsymbol{y}_j^T \cdot \boldsymbol{y}_j\right),\ i \neq j.$$

Figura B.3.1 prezintă cazul separării a două clase din $\mathbb{R}^2$ reprezentate prin prototipurile $\boldsymbol{y}_1$, $\boldsymbol{y}_2$, cărora le corespund punctele A_1 și A_2 din plan. Frontiera de decizie, dreapta de ecuație $d_{12}(\boldsymbol{x}) = 0$, reprezintă mediatoarea segmentului A_1A_2. În figura B.3.2 este ilustrat cazul separării a trei clase din $\mathbb{R}^2$ reprezentate prin prototipurile $\boldsymbol{y}_1$, $\boldsymbol{y}_2$, $\boldsymbol{y}_3$, cărora le corespund punctele A_1, A_2 și A_3 din plan. Se observă că regiunile de decizie definite conform relației (B.2.8) pentru cele trei clase considerate acoperă întreg spațiul $\mathbb{R}^2$ (cu excepția punctelor situate pe semidreptele de separație a acestor regiuni).

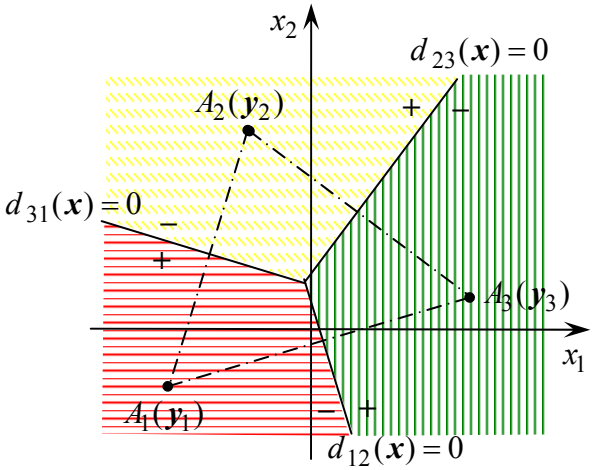


Figura B.3.2. *Regiunile de decizie corespunzătoare separării a trei clase din* $\mathbb{R}^2$ *pe baza distanței euclidiene minime.*

B.3.2. Clase reprezentate prin mai multe prototipuri

Sunt situații în care o clasă poate fi reprezentată prin mai multe prototipuri. Presupunem că un număr p_i de prototipuri notate $\boldsymbol{y}_i^{(1)}, \boldsymbol{y}_i^{(2)} \ldots \boldsymbol{y}_i^{(p_i)}$ reprezintă clasa $\mathscr{C}_i$, $i = \overline{1,m}$. În acest caz distanța de la o formă $\boldsymbol{x} \in \mathbb{R}^n$ la clasa $\mathscr{C}_i$ este definită prin

$$D_i = \min_{1 \le q \le p_i} \left\| \boldsymbol{x} - \boldsymbol{y_i}^{(q)} \right\|, \; i = \overline{1,m} \,. \tag{B.3.6}$$

În mod similar cazului în care fiecare clasă este reprezentată printr-un singur prototip, un clasificator bazat pe distanța (B.3.6) atribuie forma $\boldsymbol{x} \in \mathbb{R}^n$ clasei $\mathscr{C}_j$ situată la distanță minimă de $\boldsymbol{x}$, adică

$$\boldsymbol{x} \in \mathscr{C}_j \Leftrightarrow D_j = \min_{1 \le i \le m} D_i \,. \tag{B.3.7}$$

Dacă valoarea minimă este atinsă pentru mai mulți indici j simultan, atunci forma $\boldsymbol{x}$ este fie clasificată în clasa corespunzătoare celui mai mic indice, fie nu este clasificată deloc.

O formă echivalentă a acestui mod de clasificare a formelor din $\mathbb{R}^n$ este de a determina distanțele de la $\boldsymbol{x}$ la toți vectorii prototip $\boldsymbol{y}_1^{(1)}, \ldots, \boldsymbol{y}_1^{(p_1)}, \boldsymbol{y}_2^{(1)}, \ldots, \boldsymbol{y}_m^{(p_m)}$, și de a atribui $\boldsymbol{x}$ clasei căreia îi aparține prototipul situat la distanța minimă de $\boldsymbol{x}$. Acest algoritm poartă numele de *clasificare pe baza celui mai apropiat vecin* (eng. *nearest-neighbor classification*).

În afară de norma euclidiană (norma 2), în probleme de clasificare mai sunt frecvent utilizate și normele Hölder 1 (denumită și distanța Manhatan) și ∞ (sau distanța Chebychev) definite pentru doi vectori oarecare $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, $\boldsymbol{x} = \left[x_1 \; \ldots \; x_n \right]^T$, $\boldsymbol{y} = \left[y_1 \; \ldots \; y_n \right]^T$, prin:

$$d_1(\boldsymbol{x}, \boldsymbol{y}) = \left\| \boldsymbol{x} - \boldsymbol{y} \right\|_1 = \sum_{i=1}^{n} | \, x_i - y_i \, | \,, \tag{B.3.8}$$

respectiv,

$$d_\infty(\boldsymbol{x}, \boldsymbol{y}) = \left\| \boldsymbol{x} - \boldsymbol{y} \right\|_\infty = \max_{i=\overline{1,n}} \left(| \, x_i - y_i \, | \right). \tag{B.3.9}$$

De asemenea, este frecvent utilizată distanța Mahalanobis, definită prin

$$d_{\boldsymbol{A}}(\boldsymbol{x}, \boldsymbol{y}) = \left[\left(\boldsymbol{x} - \boldsymbol{y} \right)^T \boldsymbol{A} \left(\boldsymbol{x} - \boldsymbol{y} \right) \right]^{\frac{1}{2}}, \tag{B.3.10}$$

unde matricea $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ este simetrică și pozitiv definită.

Bibliografie

Agarwal, M., A systematic classification of neural-network-based control, *IEEE Control Systems Magazine*, vol. 17, no. 2, pp. 75-93, 1997.

Chong, E.K.P. and Żak, S.H., *An Introduction to Optimization*, Second Edition, John Wiley & Sons, Inc., New York, 2001.

Cybenko, G., 1989. "Approximation by superpositions of a sigmoidal function", *Mathematics of Control, Signals and Systems*, vol 2, pp. 303 – 314.

Demuth, H., Beale, M. and Hagan, M., *Neural Networks Toolbox* 5 – *User's Guide*, The MathWorks Inc., 2007.

Duch, W. and Jankowski, N., Survey of Neural Transfer Functions, *Neural Computing Surveys*, vol. 2, pp. 163-212, 1999.

Dumitrache, I., Constantin, N., Drăgoicea, M., *Reţele neurale. Identificarea şi Conducerea Proceselor*, MatrixRom, 1999.

Eberhart, R.C. and Yuhui Shi, *Computational Intelligence: Concepts to Implementations*, Morgan Kaufmann; 2007.

Graupe, D., *Principles of Artificial Neural Networks*, 2nd Edition, World Scientific Publishing Co. Pte. Ltd., 2007.

Gupta, M.M., Liang Jin and Homma, N., *Static and Dynamic Neural Networks: From Fundamentals to Advanced Theory*, John Wiley & Sons, Inc., Hoboken, NJ, USA, 2003

Hagan, M.T., Demuth, H.B. and Beale, M.H., *Neural Network Design*, PWS Pub. Co., 1996.

Haykin, S., *Neural Networks: A Comprehensive Foundation, 2nd Ed.*, Prentice-Hall, Pearsons Education, India, 2005.

Hu, Y.H. and Hwang, J.N. (Eds.), *Handbook of Neural Network Signal Processing*, CRC Press LLC, Boca Raton, FL, USA, 2002.

Hunt, K.J., Sbarbaro, D., Zbikowski, R. and Gawthrop, P.J., Neural Networks for Control System - A Survey, *Automatica*, vol. 28, pp. 1083-1112, 1992.

Jang, J.S.R., Sun, C.T. and Mizutani, E., *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*, Prentice-Hall Inc., Upper Saddle River, NJ, USA, 1997.

Kecman, V., *Learning And Soft Computing. Support Vector Machines, Neural Networks and Fuzzy Logic Models*, The MIT Press, Cambridge, MA, 2000.

Kloetzer, M., Ardelean, D., Păstrăvanu, O., Developing Simulink tools for teaching neural-network-based identification, *9th IEEE Mediterranean Conference on Control and Automation, MED 01*, June 27-29, 2001, Dubrovnik, Croatia.

Kloetzer, M., Păstrăvanu,O., Simulink blockset for neural predictive control, *Buletinul Ştiinţific al Universităţii Politehnica din Timişoara, Periodica Politechnica, Transactions On Automatic Control And Computer Science*, Vol. 49 (63), No.2, pp. 19-22, 2004.

Lazăr, C., *Conducerea predictivă a proceselor cu model cunoscut,* Ed. Matrix Rom, Bucureşti, 1999.

Levin, A.U. and Narendra, K.S, Control of nonlinear dynamical systems using neural networks. I. Controllability and stabilization, *IEEE Trans. Neural Networks*, vol. 4, iss. 3, pp. 192-206 1993.

Levin, A.U. and Narendra, K.S, Control of nonlinear dynamical systems using neural networks. II. Observability, identification, and control, *IEEE Trans. Neural Networks*, vol. 7, iss. 1, pp. 30-42, 1996.

Lewis, F.L., Jagannathan, S. and Yeildirek, A., *Neural Network Control of Robot Manipulators and Nonlinear Systems*, Taylor & Francis, New York, 1999.

Ljung, L., Sjöberg, J., Hjalmarson, H., "On neural network model structures in system identification" in Bittanti, S., Picci, G. (Eds.) *Identification, Adaptation, Learning*, NATO ASI Series, Springer, 1996.

Liu, G.P., Kadirkamanathan, V. and Billings, S.A., Predictive Control for Non-Linear Systems Using Neural Networks. *International Journal of Control*, nr. 71, 1998, pp. 1119-1132.

Masoud, M., Sarker, R.A. and Yao, X., *Computational Intelligence in Control*, Ideea Group Publishing, London, UK, 2003.

Narendra, K.S. and Lewis, F.L. (Eds.), Special issue on neural network feedback control, *Automatica*, vol. 37, no. 8, 2001.

K.S. Narendra, K. Parthasarathy, "Identification and control of dynamical systems using neural networks", *IEEE Trans. on Neural Networks*, vol. **1**, issue 1, pp. 4-27, 1990.

Nguyen, H.T., Prasad, N.R., Walker, C.L. and Walker, E.A., *A First Course in Fuzzy and Neural Control*, Chapman & Hall/CRC Press, Boca Ratton, Fl, 2003.

Rosenblatt, F., *Principles of Neurodynamics,* Spartan Press, Washington D.C., 1961.

Sarangapani, J., *Neural Network Control of Nonlinear Discrete-Time Systems*, CRC Press, Taylor & Francis Group, Boca Raton, FL, USA, 2006.

Sima, V. şi Varga, A., *Practica optimizării asistate de calculator*, Editura Tehnică. Bucureşti, 1986.

Sun, W. and Ya-Xiang Yuan, *Optimization Theory and Methods: Nonlinear Programming*, Springer, USA, 2006.

*** *Using Simulink*, The MathWorks Inc., 2007.

Widrow, B. and Lehr, M.A., 30 Years of adaptive neural networks: Perceptron, Madaline, and Backpropagation, *Proceedings of the IEEE*, vol. 78 no.9, September 1990.