Inteligenţă artificială
Laboratorul 13

Antrenarea rețelelor neuronale

1. Modelul neuronului artificial
2. Rețele neuronale tip perceptron
3. Rețele neuronale MLP
   3.1. Metode de accelerare a învățării
   3.2. Considerente practice
4. Matlab Neural Network Toolbox
5. Aplicații

1. Modelul neuronului artificial

Reţelele neuronale feedforward sunt sisteme statice, adică sisteme ce realizează un transfer instantaneu intrare-ieşire. Acest transfer poate fi descris prin compunerea unor funcţii dependente de topologia (arhitectura) reţelei. Termenul de „feedforward" (însemnând „cu alimentare înainte" – semnalele propagându-se numai în sensul de la intrare către ieşire) marchează diferenţa faţă de *reţelele neuronale recurente*, sau cu feedback, care sunt sisteme dinamice, posedând căi de reacţie ce permit propagarea semnalelor şi de la ieşire către intrare. Deosebirea dintre cele două clase de reţele neuronale, introdusă prin terminologie, se regăseşte şi în descrierea matematică a transferului intrare-ieşire. Reţelele neuronale recurente sunt sisteme dinamice şi, drept urmare descrierea lor se realizează prin ecuaţii diferenţiale (în cazul reţelelor cu timp continuu) şi prin ecuaţii cu diferenţe (pentru reţelele cu timp discret). Elementul de bază în structura unei reţele neuronale (feedforward sau recurente) îl constituie *neuronul artificial*. Pentru construirea unei reţele neuronale, neuronii artificiali se conectează în unul sau mai multe straturi, definind astfel, *arhitectura reţelei*.

În figura de mai jos este prezentată schema bloc a unui neuron cu mai multe intrări $\boldsymbol{x} = [x_1, \dots, x_n]^T \in \mathbb{R}^n$ şi ieşirea scalară y. Constantele $\boldsymbol{w} = [w_1, \dots, w_n] \in \mathbb{R}^{1\times n}$ se numesc *ponderi* (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 $\boldsymbol{x}$, valoarea cu care este alimentat blocul b este tot timpul 1. Referirea comună a constantelor $\boldsymbol{w}, b$ se face prin termenul de "*parametrii neuronului*". În utilizarea neuronului, valorile parametrilor sunt *ajustabile*. Pe ajustarea adecvată a acestor valori se bazează capacitatea de învăţare a neuronului artificial.

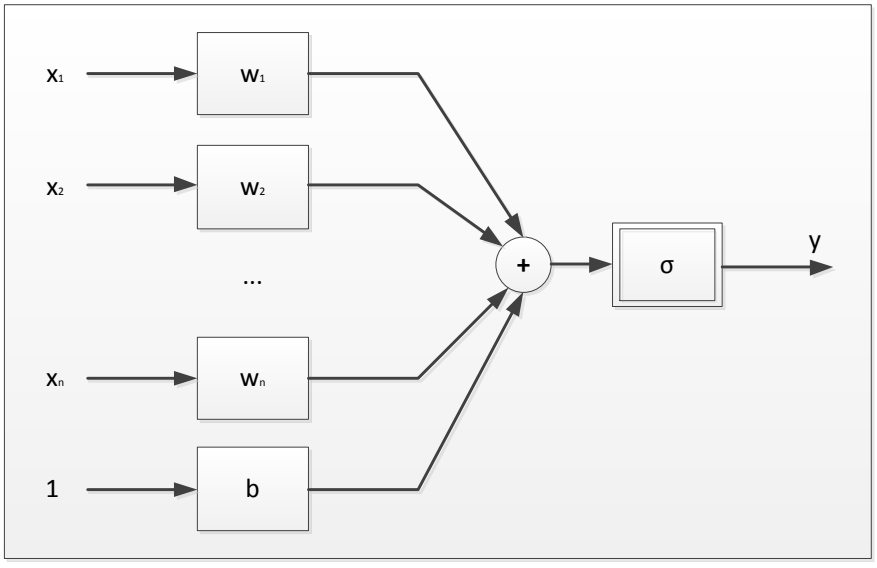
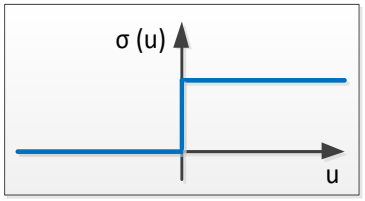
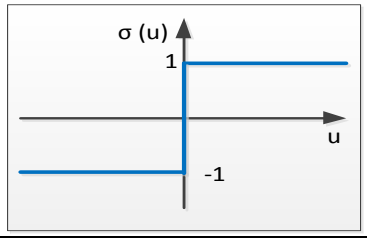
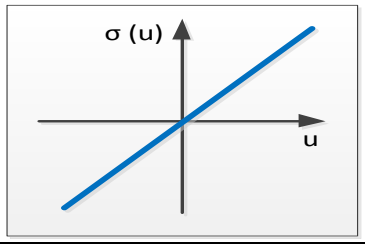
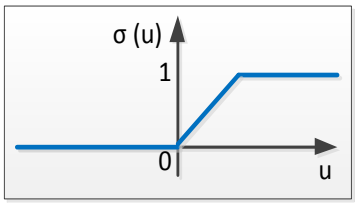
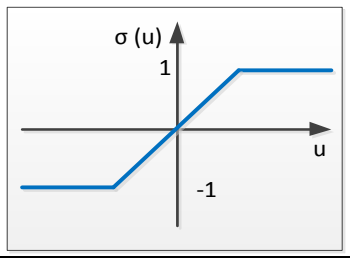
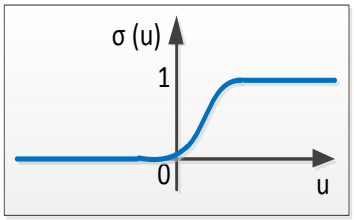
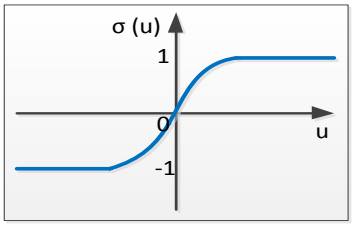


Figura 1. Neuronul artificial cu mai multe intrări

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

Funcţia de activare	**Expresia analitică**	**Reprezentarea grafică**
Treaptă unipolară **hardlim**	$\sigma(u) = \begin{cases} 0, u < 0 \\ 1, u \geq 0 \end{cases}$	
Treaptă bipolară **hardlims**	$\sigma(u) = \begin{cases} -1, u < 0 \\ 1, u \geq 0 \end{cases}$	
Funcţie liniară **purelin**	$\sigma(u) = u$	
Liniară cu saturaţie **satlin**	$\sigma(u) = \begin{cases} 0, u < 0 \\ u, 0 \leq u \leq 1 \\ 1, u > 1 \end{cases}$	
Liniară cu saturaţie, simetrică **satlins**	$\sigma(u) = \begin{cases} -1, u < -1 \\ u, -1 \leq u \leq 1 \\ 1, u > 1 \end{cases}$	
Sigmoid unipolar **logsig**	$\sigma(u) = \frac{1}{1 + e^{-u}}$	
Sigmoid bipolar **tansig**	$\sigma(u) = \frac{e^{u} - e^{-u}}{e^{u} + e^{-u}}$	

Neuronul artificial cu mai multe intrări 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 funcţionare este descrisă prin aplicaţia:

$$f: \mathbb{R} \to \mathbb{R}$$

$$y = f(\boldsymbol{x}) = \sigma\big(u(\boldsymbol{x})\big) = \sigma(r(\boldsymbol{x}) + b) = \sigma(\boldsymbol{w}\boldsymbol{x} + b) = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

Pentru cazul particular $n = 1$, se obține un neuron cu o singură intrare.

2. Rețele neuronale de tip perceptron

Reţelele neuronale feed-forward cu funcţii de activare treaptă poartă denumirea de **reţele perceptron**. Asemenea reţele pot fi utilizate numai în probleme de clasificare liniară şi necesită algoritmi de antrenare specifici, care exploatează forma particulară a funcţiilor de activare.

Se consideră un perceptron cu n intrări, cu funcție de activare de tip treaptă unipolară. Se observă că spaţiul de intrare, poate fi separat în două clase în funcție de valorile ieşirii perceptronului:

$$\mathcal{C}_0 = \{\boldsymbol{x} \in \mathbb{R}^n | \, y(\boldsymbol{x}) = 0\}$$
$$\mathcal{C}_1 = \{\boldsymbol{x} \in \mathbb{R}^n | \, y(\boldsymbol{x}) = 1\}$$

Suprafaţa de separaţie dintre cele două clase este hiperplanul de ecuaţie $u(\boldsymbol{x}) = 0$.

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* aparţin că aparţin la două clase, notate $\mathcal{C}_0$ şi $\mathcal{C}_1$, să se determine parametrii unui perceptron care să realizeze separarea dorită. Dacă această problemă admite soluţie (dacă există un hiperplan care să separe punctele ce aparţin unei clase de punctele celeilalte clase) se spune că cele două clase sunt liniar separabile. În cazurile simple, parametrii unui perceptron pot fi calculaţi în mod direct astfel încât să se realizeze separarea spaţiului de intrare în modul dorit, aşa cum se prezintă în continuare.

O reţea neuronală cu un singur strat cu *n* intrări, *p* ieşiri (neuroni) şi funcţii de activare treaptă (unipolară sau bipolară) posedă proprietatea de a partiţiona spaţiul vectorilor de intrare $\mathbb{R}^n$ în 2^p mulţimi prin *p hiperplane*. O asemenea reţea va putea fi utilizată în probleme de clasificare numai dacă regiunile de decizie sunt mulţimi din $\mathbb{R}^n$ separabile liniar.

Algoritmul de antrenare, învăţare, sau instruire (eng. training, learning) constă în actualizarea parametrilor reţelei, adică a ponderilor $\boldsymbol{W} \in \mathbb{R}^{p \times n}$ (elementele matricei $\boldsymbol{W}$) şi a deplasărilor $\boldsymbol{b}$ (elementele vectorului coloană $\boldsymbol{b} \in \mathbb{R}^p$), cu scopul de a satisface cele *N* egalităţi $f(\boldsymbol{x}_i) = \boldsymbol{z_i}$.

I. Etapa de inițializare

Se organizează vectorii prototip $\boldsymbol{x}_i \in \mathbb{R}^n, i = \overline{1, N}$ şi cei ţintă $\boldsymbol{z}_i \in \mathbb{R}^p$ sub forma matricelor: $\boldsymbol{X} = [\boldsymbol{x_1}, \dots, \boldsymbol{x_N}] \in \mathbb{R}^{n \times N}$, respective $\boldsymbol{Z} = [\boldsymbol{z_1}, \dots, \boldsymbol{z_N}] \in \mathbb{R}^{p \times N}$. Se inițializează ponderile și deplasările cu valori numerice aleatoare din intervalul $[-0.5; 0.5]$ (sau cu valori nule). Pentru algoritmul de antrenare se stabileşte de către utilizator un număr maxim de iteraţii sau epoci. La fiecare iteraţie a algoritmului de antrenare se parcurg etapele descrise mai jos. Valorile parametrilor reţelei la începerea unei iteraţii sunt notate prin $\boldsymbol{W_{old}}$ pentru ponderi) şi $\boldsymbol{b_{old}}$ (pentru deplasări).

II. Etapa de prezentare

Pentru valorile curente ale parametrilor rețelei se calculează ieșirile rețelei:

$$\boldsymbol{y_i} = \boldsymbol{f}(\boldsymbol{x_i}) = \boldsymbol{\sigma}(\boldsymbol{W_{old}}\boldsymbol{x_i} + \boldsymbol{b_{old}}), i = \overline{1, N}$$

organizate sub forma matricei $\boldsymbol{Y} = [\boldsymbol{y_1}, \dots, \boldsymbol{y_N}] \in \mathbb{R}^{p \times N}$.

III. Etapa de verificare

Se calculează vectorii erorilor ca diferențe între vectorii țintă și vectorii ieșire la iterația curentă:

$$\boldsymbol{e_i} = \boldsymbol{z_i} - \boldsymbol{y_i} \in \mathbb{R}^p, i = \overline{1, N}$$

Algoritmul se oprește dacă este îndeplinită una din următoarele condiții:

i) toți vectorii eroare sunt nuli $\boldsymbol{e_i} \equiv \boldsymbol{0}, i = \overline{1, N}$;
ii) a fost atins numărul maxim de iterații.

La final, algoritmul va furniza valorile parametrilor rețelei rezultate din antrenare, care vor fi notate prin $\boldsymbol{W_f}$ (ponderi) și $\boldsymbol{b_f}$ (deplasări). Dacă nici una din condițiile de stop nu este îndeplinită, se continuă cu etapa următoare a iterației curente.

IV. Etapa de actualizare a parametrilor

Se construiește matricea erorilor:

$$\boldsymbol{E} = [\boldsymbol{e_1}, \dots, \boldsymbol{e_N}] = \boldsymbol{Z} - \boldsymbol{Y} \in \mathbb{R}^{p \times N}$$

cu ajutorul căreia se calculează matricele de actualizare a parametrilor:

i) pentru ponderi: $\boldsymbol{D_W} = \boldsymbol{E} \cdot \boldsymbol{X^T} \in \mathbb{R}^{p \times n}$
ii) pentru deplasări: $\boldsymbol{D_b} = \boldsymbol{E} \cdot [\boldsymbol{1} \dots \boldsymbol{1}]^{\boldsymbol{T}} \in \mathbb{R}^p$.

Se calculează noile valori ale parametrilor:

i) pentru ponderi: $\boldsymbol{W_{new}} = \boldsymbol{W_{old}} + \boldsymbol{D_w}$
ii) pentru deplasări: $\boldsymbol{b_{new}} = \boldsymbol{b_{old}} + \boldsymbol{D_b}$.

Cu aceste calcule se încheie iterația curentă și se trece la o nouă iterație (Etapa II) efectuând mai întâi atribuirile: $\boldsymbol{W_{old}} \leftarrow \boldsymbol{W_{new}}$ și $\boldsymbol{b_{old}} \leftarrow \boldsymbol{b_{new}}$.

Exemplu

Ilustrăm aplicarea algoritmului de antrenare prezentat anterior în cazul determinării parametrilor unui perceptron ($p = 1$) cu două intrări ($n = 2$) ce implementează funcția logică SAU.

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 $\boldsymbol{Z} = [0 \quad 1 \quad 1 \quad 1]$. Pentru simplitatea calculelor, inițializăm parametrii perceptronului cu valori nule: $\boldsymbol{W_{old}} = [0 \quad 0], \boldsymbol{b_{old}} = 0$

Iterația 1:
$\boldsymbol{Y} = [1 \quad 1 \quad 1 \quad 1]$
$\boldsymbol{E} = [-1 \quad 0 \quad 0 \quad 0]$
$\boldsymbol{D_W} = [0 \quad 0], \boldsymbol{D_b} = -1$
$\boldsymbol{W_{new}} = [0 \quad 0], \boldsymbol{b_{new}} = -1.$

Iterația 2:
$\boldsymbol{W_{old}} = [0 \quad 0], \boldsymbol{b_{old}} = -1$
$\boldsymbol{Y} = [0 \quad 0 \quad 0 \quad 0]$
$\boldsymbol{E} = [0 \quad 1 \quad 1 \quad 1]$
$\boldsymbol{D_W} = [2 \quad 2], \boldsymbol{D_b} = \mathbf{3}$
$\boldsymbol{W_{new}} = [2 \quad 2], \boldsymbol{b_{new}} = 2.$

Iterația 3:
$\boldsymbol{W_{old}} = [2 \quad 2], \boldsymbol{b_{old}} = 2$
$\boldsymbol{Y} = [1 \quad 1 \quad 1 \quad 1]$
$\boldsymbol{E} = [-1 \quad 0 \quad 0 \quad 0]$
$\boldsymbol{D_W} = [0 \quad 0], \boldsymbol{D_b} = -1$
$\boldsymbol{W_{new}} = [2 \quad 2], \boldsymbol{b_{new}} = 1.$

Iterația 4:
$\boldsymbol{W_{old}} = [2 \quad 2], \boldsymbol{b_{old}} = 1$
$\boldsymbol{Y} = [1 \quad 1 \quad 1 \quad 1]$
$\boldsymbol{E} = [-1 \quad 0 \quad 0 \quad 0]$
$\boldsymbol{D_W} = [0 \quad 0], \boldsymbol{D_b} = -1$
$\boldsymbol{W_{new}} = [2 \quad 2], \boldsymbol{b_{new}} = 0.$

Iterația 5:
$\boldsymbol{W_{old}} = [2 \quad 2], \boldsymbol{b_{old}} = 0$
$\boldsymbol{Y} = [1 \quad 1 \quad 1 \quad 1]$
$\boldsymbol{E} = [-1 \quad 0 \quad 0 \quad 0]$
$\boldsymbol{D_W} = [0 \quad 0], \boldsymbol{D_b} = -1$
$\boldsymbol{W_{new}} = [2 \quad 2], \boldsymbol{b_{new}} = -1.$

Iterația 6:
$\boldsymbol{W_{old}} = [2 \quad 2], \boldsymbol{b_{old}} = -1$
$\boldsymbol{Y} = [0 \quad 1 \quad 1 \quad 1]$
$\boldsymbol{E} = [0 \quad 0 \quad 0 \quad 0]$
STOP.

Algoritmul se încheie furnizând valorile parametrilor rețelei rezultate din antrenare, $\boldsymbol{W_f} = [2 \quad 2], \boldsymbol{b_f} = -1$. Ecuația dreptei de separație este $d(x) = 2x_1 + 2x_2 - 1 = 0$.

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 și 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. Rețele neuronale MLP

Perceptronul multistrat (MLP) este o rețea neuronală feedforward ce conține unul sau mai multe straturi ascunse. De obicei, rețeaua este formată dintr-un strat de intrare conținând neuronii sursă, cel puțin un strat ascuns conținând neuroni computaționali și un strat de ieșire conținând neuroni computaționali. Semnalele de intrare se propagă de la intrare spre ieșirea rețelei strat după strat.

Ca orice rețea neuronală, o rețea cu propagare înapoi este caracterizată de conexiunile dintre neuroni (care formează arhitectura rețelei), de funcțiile de activare folosite de neuroni și de algoritmul de învățare ce specifică procedura folosită pentru ajustarea ponderilor. De obicei o rețea neuronală cu propagare înapoi este o rețea multistrat ce conține trei sau patru straturi complet conectate. Fiecare neuron își calculează ieșirea similar cu perceptronul. Apoi valoarea de intrare este transmisă funcției de activare. Spre deosebire de perceptron, neuronii din rețelele cu propagare înapoi au funcții de activare de tip sigmoid. Derivata acesteia este foarte ușor de calculat și garantează ieșirea în intervalul $[0, 1]$.

Fiecare strat dintr-o rețea neuronală MLP îndeplinește o funcție specifică. Stratul de intrare acceptă semnale de intrare și rareori conține neuroni computaționali de aceea nu procesează tiparele de intrare. Stratul de ieșire acceptă semnale de ieșire (stimuli proveniți din stratul ascuns) și stabilește ieșirea rețelei în funcție de acesta. Neuronii din stratul ascuns detectează trăsăturile, iar

ponderile acestora reprezintă trăsăturile ascunse de tiparele de intrare. Aceste trăsături sunt folosite apoi de stratul de ieşire pentru a determina tiparul de ieşire.

Un strat ascuns ascunde ieşirea dorită. Neuronii din stratul ascuns nu pot fi observaţi prin intermediul comportamentului intrare-ieşire al reţelei. Nu există un mod evident de a cunoaşte care trebuie să fie ieşirea dorită a stratului ascuns. Ieşirea dorită va fi determinată de însuşi stratul ascuns.

Folosind un singur strat ascuns putem reprezenta orice funcţie continuă a semnalelor de intrare, iar cu două straturi ascunse chiar şi funcţii cu discontinuităţi.

Reţelele neuronale au capacitatea de a învăţa, însă modalitatea concretă în care se realizează acest proces este dată de algoritmul folosit pentru antrenare. O reţea se consideră antrenată dacă aplicarea unui vector de intrare conduce la obţinerea unei ieşiri dorite, sau foarte apropiate de aceasta. Antrenarea constă în aplicarea secvenţială a diferiţi vectori de intrare şi ajustarea ponderilor din reţea în raport cu o procedură predeterminată. În acest timp, ponderile conexiunilor converg gradual spre anumite valori pentru care fiecare vector de intrare produce vectorul de ieşire dorit.

Învăţarea supervizată presupune aplicarea unei perechi vector de intrare – vector de ieşire dorit. După aplicarea unei intrări, se compară ieşirea calculată cu ieşirea dorită, după care diferenţa este folosită pentru modificarea ponderilor cu scopul minimizării erorii la un nivel acceptabil.

Într-o rețea neuronală cu propagare înapoi, algoritmul de învățare cuprinde două etape: faza de prezentare a tiparelor de antrenare pentru stratul de intrare. Reţeaua propagă strat după strat tiparele de antrenare până la generarea tiparului de ieşire. Dacă acesta este diferit de tiparul ţintă dorit, se va calcula eroarea şi va fi propagată înapoi, de la ieşire spre intrare. Ponderile sunt actualizate simultan cu propagarea erorii.

Algoritmul back-propagation este cel mai cunoscut şi utilizat algoritm de învăţare supervizată. Numit şi *algoritmul delta generalizat*, el se bazează pe minimizarea diferenţei dintre ieşirea dorită şi ieşirea reală, prin metoda gradientului descendent. Gradientul ne spune cum variază funcţia în diferite direcţii. Ideea algoritmului este găsirea minimului funcţiei de eroare în raport cu ponderile conexiunilor. Eroarea este dată bineînţeles de diferenţa dintre ieşirea dorită şi ieşirea efectivă a reţelei. Cea mai utilizată funcţie de eroare este eroarea pătratică medie. Dacă avem în setul de antrenare K modele iar reţeaua are N ieşiri, atunci:

$$E = \frac{1}{K \cdot N} \cdot \sum_k \sum_n \left(y_{kn} - o_{kn} \right)^2 ,$$

unde y_{kn} este valoarea dorită pe ieşirea n a reţelei pentru modelul k, iar o_{kn} este ieşirea efectivă a reţelei.

O epocă reprezintă prezentarea tuturor exemplelor din setul de antrenare. În majoritatea cazurilor, antrenarea reţelei presupune mai multe epoci de antrenare.

Pentru a păstra rigoarea matematică, ponderile vor fi ajustate numai după ce *toţi* vectorii de test vor fi aplicaţi reţelei. În acest scop, gradienţii ponderilor trebuie memoraţi şi ajustaţi după fiecare model din setul de antrenare, iar la sfârşitul unei epoci de antrenare, se vor modifica ponderile *o singură dată* (există şi varianta „on-line”, mai simplă, în care ponderile sunt actualizate direct; în acest caz, poate conta ordinea în care sunt prezentaţi reţelei vectorii de test).

Dacă s-a încheiat o epocă, se testează dacă s-a îndeplinit criteriul de terminare ($E < E_{\max}$ sau s-a atins un număr maxim de epoci de antrenare). Dacă nu, se trece la o nouă epocă. Dacă da, algoritmul se termină.

3.1. Metode de accelerare a învăţării

Antrenarea unui perceptron multistrat este deseori destul de lentă, necesitând mii sau zeci de mii de epoci pentru probleme complexe. Cele mai cunoscute metode de accelerare a învăţării sunt: metoda momentului şi aplicarea unei rate de învăţare variabile.

Metoda momentului (engl. „momentum") propune adăugarea unui termen la ajustarea ponderilor. Acest termen este proporţional cu ultima modificare a ponderii, adică valorile cu care se ajustează ponderile sunt memorate şi influenţează în mod direct toate ajustările ulterioare.

Metoda ratei de învăţare variabile este o tehnică simplă de accelerare a învăţării şi constă în utilizarea unei rate de învăţare individuale pentru fiecare pondere şi adaptarea acestor parametri în fiecare iteraţie, în funcţie de semnele succesive ale gradienţilor. Dacă pe parcursul antrenării eroarea începe să crească, în loc să scadă, ratele de învăţare se resetează la valorile iniţiale şi apoi se continuă procesul.

Notă: mai multe detalii despre algoritmul back-propagation pot fi găsite în cursul 11 şi laboratorul opţional 17.

3.2. Considerente practice

În implementarea algoritmului apar o serie de probleme practice, legate în special de alegerea parametrilor antrenării şi a configuraţiei reţelei.

În primul rând, o rată de învăţare mică determină o convergenţă lentă a algoritmului, pe când una prea mare poate cauza eşecul (algoritmul va „sări" peste soluţie). Pentru probleme relativ simple o rată de învăţare $\eta = 0.7$ este acceptabilă, însă în general se recomandă rate de învăţare în jur de 0.2. Pentru accelerarea prin metoda momentului, o valoare satisfăcătoare pentru α este 0.9. În cazul ratei de învăţare variabile, valori tipice care funcţionează bine în cele mai multe situaţii sunt: $u = 1.2$ şi $d = 0.8$.

Alegerea funcţiei de activare pentru stratul de ieşire al reţelei depinde de natura problemei care trebuie rezolvată.

Pentru neuronii din straturile ascunse, se preferă funcţii sigmoide, deoarece au avantajul atât al neliniarităţii cât şi al diferenţiabilităţii (condiţie necesară pentru aplicarea algoritmului backpropagation). Influenţa cea mai mare a unei sigmoide asupra performanţelor algoritmului se pare că o are simetria faţă de origine. Sigmoida bipolară este simetrică faţă de origine, în timp ce sigmoida unipolară este simetrică faţă de punctul (0, 0.5), ceea ce scade viteza de convergenţă.

Pentru neuronii de ieşire se recomandă funcţii de activare adaptate distribuţiei datelor de ieşire. Astfel, pentru probleme de categorizare binară (0/1), este potrivită sigmoida. Pentru clasificarea cu *n* clase, fiecare corespunde unei ieşiri binare a reţelei (de exemplu, într-o aplicaţie de recunoaştere optică a caracterelor).

Pentru valori continue se poate realiza o preprocesare şi o postprocesare a datelor, astfel încât reţeaua să opereze cu valori scalate, de exemplu în intervalul [–0.9, 0.9] pentru sigmolida bipolară (tangenta hiperbolică).

De asemenea, pentru valori continue funcţia de activare a neuronilor de ieşire poate fi liniară, mai ales dacă nu există limite cunoscute pentru intervalul în care se pot găsi acestea.

O altă problemă caracteristică acestui mod de antrenare este dată de minimele locale. Într-un minim local, gradienţii erorii devin 0 şi învăţarea nu mai continuă. O soluţie este încercarea mai multor antrenări independente, cu ponderi iniţializate diferit la început, ceea ce creşte probabilitatea găsirii minimului global. Pentru probleme mari, acest lucru poate fi greu de realizat şi atunci pot fi acceptate şi minime locale, cu condiţia ca erorile să fie suficient de mici. De asemenea, se pot încerca diferite configuraţii ale reţelei, cu un număr mai mare de neuroni în stratul ascuns sau cu mai multe straturi ascunse, care în general conduc la minime locale mai mici. Totuşi, deşi minimele locale sunt într-adevăr o problemă, în practică nu reprezintă dificultăţi de nesoluţionat.

O chestiune importantă este alegerea unei configuraţii cât mai bune pentru reţea din punct de vedere al numărului de neuroni în straturile ascunse. În multe situaţii, un singur strat ascuns este suficient. Nu există nişte reguli precise de alegere a numărului de neuroni. În general, reţeaua poate fi văzută ca un sistem în care numărul vectorilor de test înmulţit cu numărul de ieşiri reprezintă numărul de ecuaţii iar numărul ponderilor reprezintă numărul necunoscutelor. Ecuaţiile sistemului sunt în general neliniare şi foarte complexe şi deci este foarte dificilă rezolvarea lor exactă prin

mijloace convenţionale. Algoritmul de antrenare urmăreşte tocmai găsirea unor soluţii aproximative care să minimizeze erorile.

O reţea neuronală trebuie să fie capabilă de generalizare. Dacă reţeaua aproximează bine setul de antrenare, aceasta nu este o garanţie că va găsi soluţii la fel de bune şi pentru datele din alt set, cel de test. Generalizarea presupune existenţa în date a unor regularităţi, a unui model care *poate fi* învăţat. În analogie cu sistemele liniare clasice, aceasta ar însemna nişte ecuaţii redundante. Astfel, dacă numărul de ponderi este mai mic decât numărul de vectori de test, pentru o aproximare corectă reţeaua trebuie să se bazeze pe modelele intrinseci din date, modele care se vor regăsi şi în datele de test. O regulă euristică afirmă că numărul de ponderi trebuie să fie în jur sau sub o zecime din produsul dintre numărul de vectori de antrenare şi numărul de ieşiri. În unele situaţii însă (de exemplu, dacă datele de antrenare sunt relativ puţine), numărul de ponderi poate fi chiar jumătate din produs.

Pentru un perceptron multistrat se consideră că numărul de neuroni dintr-un strat trebuie să fie suficient de mare pentru ca acest strat să furnizeze trei sau mai multe laturi pentru fiecare regiune convexă identificată de stratul următor. Deci numărul de neuroni dintr-un strat trebuie să fie de peste trei ori mai mare decât cel din stratul următor.

După cum am menţionat, un număr insuficient de ponderi conduce la „sub-potrivire” (engl. “under-fitting”), în timp ce un număr prea mare de ponderi conduce la „supra-potrivire” (engl. “over-fitting”), fenomene prezentate în figura 1. Acelaşi lucru apare dacă numărul de epoci de antrenare este prea mic sau prea mare.

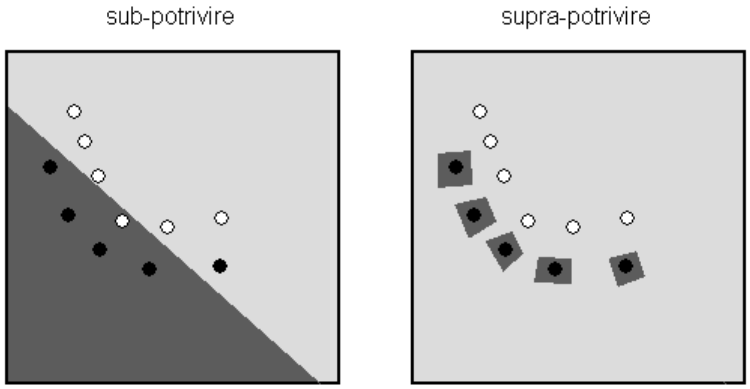


Figura 2. Capacitatea de aproximare a unei reţele neuronale în funcţie de numărul de ponderi

O metodă de rezolvare a acestei probleme este oprirea antrenării în momentul în care se atinge cea mai bună generalizare. Pentru o reţea suficient de mare, s-a verificat experimental faptul că eroarea de antrenare scade în mod continuu pe măsură ce creşte numărul epocilor de antrenare. Totuşi, pentru date diferite de cele din setul de antrenare, se constată că eroarea scade la început până la un punct în care începe din nou să crească. De aceea, oprirea antrenării trebuie făcută în momentul când eroarea pentru setul de *validare* este minimă. Acest lucru se face împărţind datele de antrenare în două: între 67-90% din date vor fi utilizate pentru antrenarea propriu-zisă iar restul, numit set de validare este folosit pentru măsurarea erorii. Antrenarea se opreşte atunci când începe să crească eroarea pentru setul de validare, moment numit „punct de maximă generalizare”.

În funcţie de performanţele reţelei în acest moment, se pot încerca apoi diferite configuraţii, crescând sau micşorând numărul de neuroni din stratul (sau straturile) intermediar(e).

4. Matlab Neural Network Toolbox

newff – funcţia construieşte o reţea MLP (utilizată până la versiunea R2010a NNET 6.0.4)
Sintaxă

net = newff(P,T,S)
net = newff(P,T,S,TF,BTF,BLF,PF,IPF,OPF,DDF)

Descriere

newff(P,T,S) primește ca parametri:

P - matricea de dimensiuni RxQ1 care conține cele mai reprezentative Q1 elemente ale celor R vectori prototip.

T - matricea de dimensiuni SNxQ2 care conține cele mai reprezentative Q2 elemente ale celor SN vectori țintă.

Si - Dimensiunile celor N-1 straturi ascunse, S1 to S(N-1), implicit []. Dimesiunea stratului de ieșire este determinată din T.

și returnează o rețea feed-forward ce folosește algoritmul backpropagation.

newff(P,T,S,TF,BTF,BLF,PF) primește următoarele intrări opționale,

TFi – Funcția de transfer a stratului i. Valoarea implicită este 'tansig' pentru straturile ascunse și 'purelin' pentru stratul de ieșire. Funcția de transfer TF{i} poate fi orice funcție derivabilă, de exemplu: TANSIG, LOGSIG sau PURELIN.

BTF – funcția de antrenare a rețelei, cu valoarea implicită 'trainlm'. Funcția de antrenare poate fi una dintre funcțiile: TRAINLM, TRAINBFG, TRAINRP sau TRAINGD.

BLF – Funcția de învățare , cu valoarea implicită 'learngdm'. Funcția de învățare BLF poate fi una din următoarele funcții: LEARNGD sau LEARNGDM.

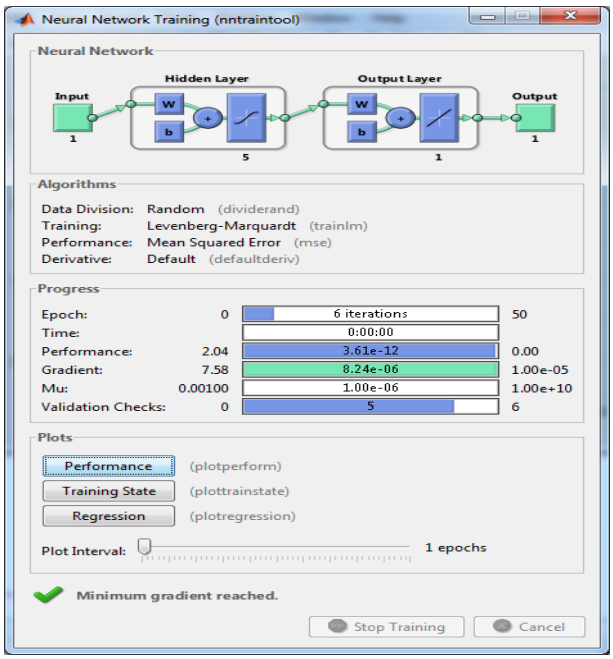
PF - funcția criteriu pentru evaluarea performanței, cu valoarea implicită 'mse'.

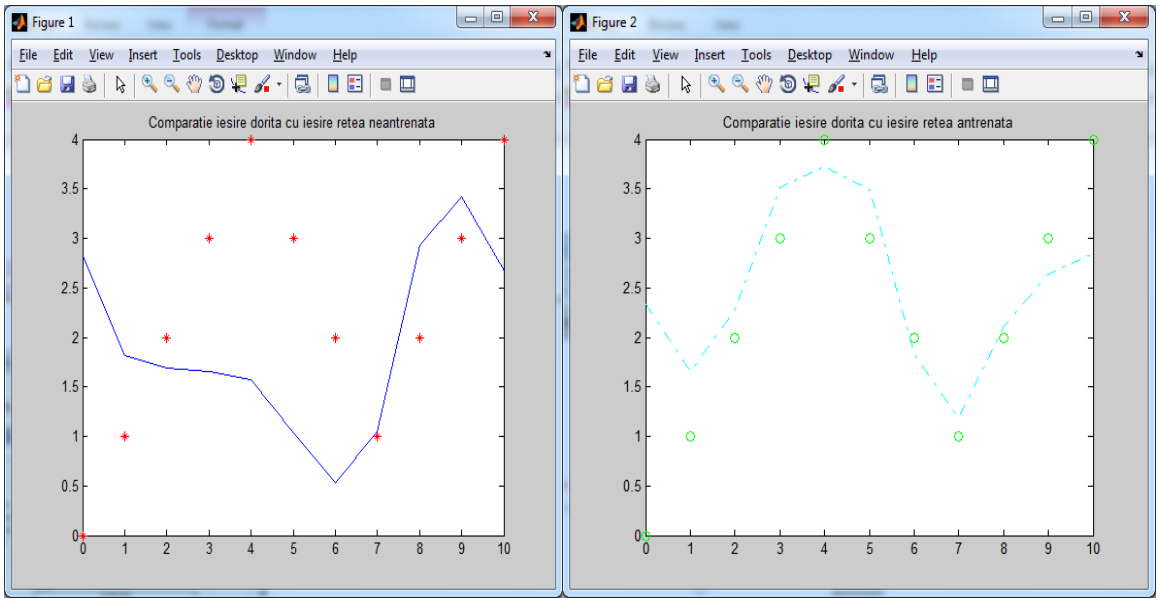
și returnează o rețea feed-forward ce folosește algoritmul backpropagation.

Exemplu

```
close all;
clear all;
clc;
% multimea vectorilor prototip
P = [0 1 2 3 4 5 6 7 8 9 10];

% multimea vetorilor tinta
T = [0 1 2 3 4 3 2 1 2 3 4];
% dimensiunile straturilor ascunse
% 1 strat ascuns cu 5 neuroni
S = [2 1];
net = newff(P, T, S);
net = newff([min(P) max(P)], S, {'tansig' 'purelin'}, 'traingd');
% evaluare retea neantrenata pe prototipurile din setul P
Y = sim(net, P);
figure(1);
plot(P, T, '*r', P, Y, '-b');
title('Comparatie iesire dorita cu iesire retea neantrenata');
% epoci de antrenare 100, 200, 500,...
net.trainParam.epochs = 50;
% eroarea 0.0001
net.trainParam.goal = 0.0001;
% rata de invatare 0.001, 0.1, 0.3
net.trainParam.lr = 0.01;
net = train(net, P, T);
Y = sim(net, P);
figure(2);
plot(P, T, 'og', P, Y, '-.c');
title('Comparatie iesire dorita cu iesire retea antrenata');
```





Parametrii rețelei antrenate:

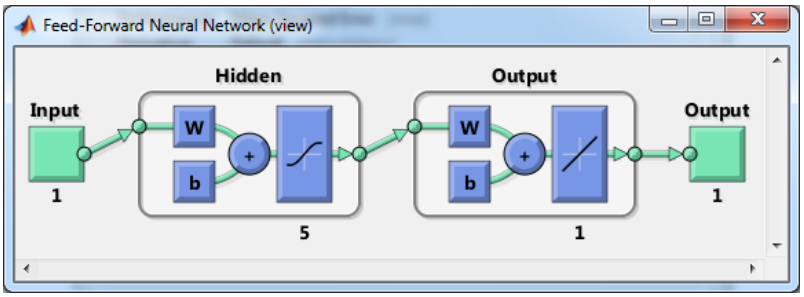
- ponderile: net.IW{1} = [-6.5669; -6.7676; -7.6076; 6.8881; -7.0563];
- deplasările: net.b{1} = [7.4293; 3.6983; 1.1361; 3.8130; -6.9153];
  net.b{2} = [0.6909];

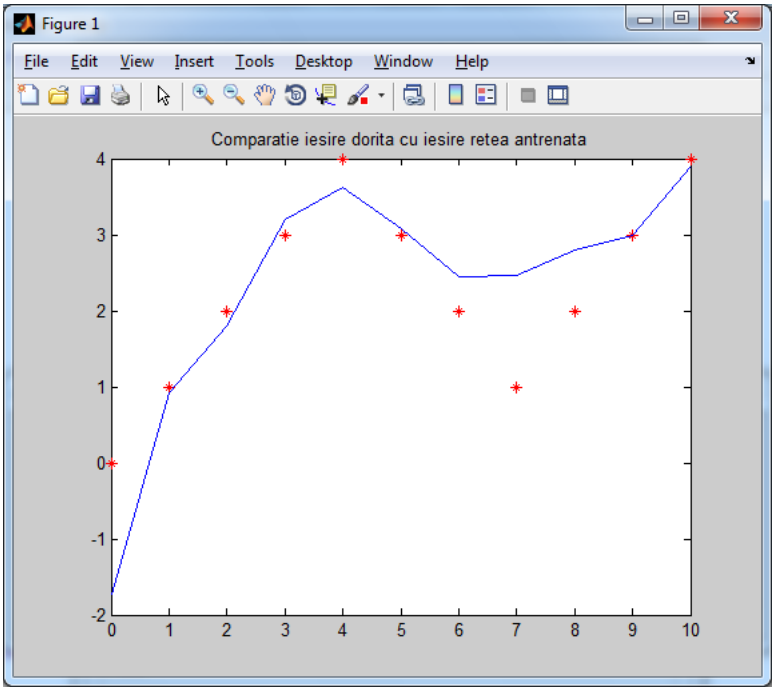
feedforwardnet – construiește o rețea neuronală feed-forward.
Sintaxă:
feedforwardnet(hiddenSizes,trainFcn) primește ca parametru vectorul linie cu dimensiunile celor N straturi ascunse și o funcție de antrenare specifică algoritmului backpropagation și returnează o rețea neuronală cu N+1 straturi. Intrările și ieșirile sunt setate pe 0, urmând a fi configurate cu datele specificate la apelarea funcției train sau manual prin apelul funcției configure. Argumentele implicite sunt (10, 'trainlm').

Exemplu

```
close all;
clear all;
clc;
% multimea vectorilor prototip
P = [0 1 2 3 4 5 6 7 8 9 10];
% multimea vetorilor tinta
T = [0 1 2 3 4 3 2 1 2 3 4];
% dimensiunile straturilor ascunse
% 1 strat ascuns cu 5 neuroni
S = [5];

net = feedforwardnet(S, 'trainlm');
net = train(net, P, T);
view(net)
Y = net(P);
perf = perform(net, P, Y);
figure(1);
plot(P, T, '*r', P, Y, '-b');
title('Comparatie iesire dorita cu iesire retea antrenata');
```





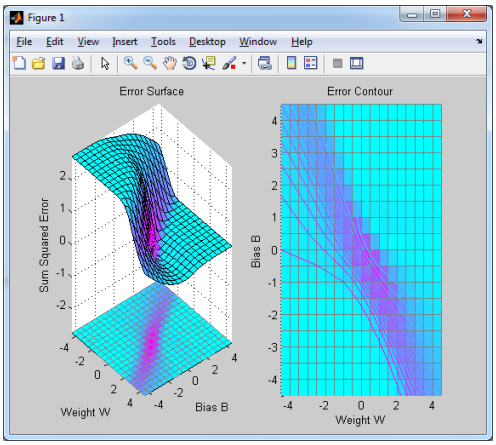
perf = 14.4846
net.IW{1} = [-7.0298; -6.9963; -6.9970; 6.8387; -6.7973];
net.b{1} = [6.9627; 3.4981;-0.0030; 3.7237; -7.2267];
net.b{2} = [-1.0255];

Utilizare plotes

```
close all;
clear all;
clc;
P = 0:pi/12:pi/2;
T = sin(P);
```

```
w_range = -4.5:0.5:4.5;
b_range = -4.5:0.5:4.5;

figure(1);
ES = errsurf(P, T, w_range, b_range, 'tansig')/length(T);
plotes(w_range, b_range, ES, [50 25]);
```



5. Aplicaţii

5.1.Implementaţi algoritmul back-propagation pentru o rețea neuronală de tip perceptron multistrat, cu un singur strat ascuns şi cu un număr variabil de unităţi în stratul de intrare şi în stratul ascuns pentru a aproxima funcția neliniară φ definită prin vectorii prototip $\boldsymbol{x_i} = \frac{(i-1)\pi}{12}, i = 1, \dots, 7$ și vectorii țintă: $z_i = \sin(x_i), i = 1, \dots, 7$.

Se vor preciza:

a) valorile inițiale ale parametrilor, reprezentarea grafică a aproximării realizate de rețea pe baza acestor valori şi eroarea corespunzătoare;

b) valorile finale ale parametrilor (rezultate din antrenare), reprezentarea grafică a aproximării realizate de rețea pe baza acestor valori şi eroarea corespunzătoare;

c) numărul total de iterații cât a durat antrenarea şi reprezentarea grafică a evoluției erorii ca funcție de numărul de iterații;

e) valorile utilizate pentru parametrii de antrenare comentate prin prisma convergenței algoritmului şi a formei suprafeței erorii. Se vor raporta şi explica eventualele eşecuri în antrenare, datorate alegerii neadecvate a parametrilor de antrenare.

5.2. Proiectați o rețea neuronală care să aproximeze funcția:

$z_i = \varphi(x_i) = 1 + e^{-x_i} \cdot \sin\left(2\pi x_i - \frac{\pi}{2}\right), i = 1, \dots, N$. Tiparele de intrare din setul de antrenare se aleg astfel: $u_i = \frac{i-1}{N-1}, i = 1, \dots N$. Se vor considera cazurile: N = 51 sau 101. Se va urmări ieşirea rețelei antrenate pe setul de date de testare $u = 0{:}\frac{0.5}{N}{:}1$. Se vor alege valori convenabile pentru numărul de neuroni ascunși (2, 4 sau 8), rata de învățare (0.01, 0.1 sau 0.3) și numărul de epoci de antrenare (100, 250 sau 500). Se vor nota rezultatele obținute.

5.3. Să se reia problema 5.2. pentru aproximarea funcției:

$\varphi(x_i) = 1 + e^{-x_i} \sin\left(8\pi x_i - \frac{\pi}{2}\right), N = 51.$