

# Calcul neuronal si calcul evolutiv

octombrie 2004

## 1 Calculul inteligent si rezolvarea problemelor

Din punctul de vedere al rezolvării automate problemele pot fi clasificate în două categorii:

- Probleme "bine-puse": caracterizate prin faptul că li se poate asocia un model formal (de exemplu, un model matematic) pe baza căruia se poate dezvolta o metodă de rezolvare cu caracter algoritmic.
- Probleme "rău-puse": caracterizate prin faptul că nu pot fi descrise complet printr-un model formal, ci cel mult se cunosc răspunsuri pentru cazuri particulare ale problemei.

Rezolvarea unei probleme înseamnă stabilirea unei asocieri între datele de intrare (valori inițiale, ipoteze etc.) și răspunsul corect. În cazul problemelor bine-puse această asociere este o relație funcțională explicită construită pe baza modelului asociat problemei.

În cazul problemelor rău-puse, însă, o astfel de relație explicită nu poate fi pusă în evidență, rolul sistemului care rezolvă problema fiind de a dezvolta o relație de asociere întrebare-răspuns pe baza unor exemple. Procesul prin care sistemul își formează modelul propriu al problemei și pe baza acestuia relația de asociere se numește *învățare*.

Pe de altă parte, din punctul de vedere al complexității rezolvării și al relevanței răspunsului problemele pot fi clasificate în:

- Probleme pentru care este esențială obținerea unui răspuns exact indiferent de resursele implicate. Acestea necesită utilizarea unor tehnici exacte.
- Probleme pentru care este preferabil să se obțină un răspuns "aproximativ" folosind resurse "rezonabile" decât un răspuns exact dar folosind resurse foarte costisitoare.

*Calculul inteligent* este un domeniu al Inteligenței Artificiale care grupează tehnici de rezolvare a problemelor "rău-puse" sau a celor pentru care modelele formale conduc la algoritmi foarte costisitori.

Principalele direcții ale calculului inteligent sunt:

- *Calcul neuronal*. Este folosit în principal în rezolvarea problemelor de asociere, bazându-se pe extragerea, prin învățare, a unui model pornind de la exemple. Sursa de inspirație o reprezintă structura și funcționarea creierului.
- *Calcul evolutiv*. Este folosit în principal în rezolvarea problemelor bazate pe căutarea soluției într-un spațiu mare de soluții potențiale. Sursa de inspirație o reprezintă principiile evoluționismului darwinist.

- *Calculul fuzzy*. Este folosit atunci când datele problemei (și relațiile dintre acestea) nu pot fi descrise exact ci se caracterizează prin prezența unui grad de incertitudine ("fuzziness"). Ideea de bază este de a înlocui valorile exacte ("crisp") cu valori "fuzzy" descrise prin funcții de apartenență.

În fiecare dintre cele trei direcții majoritatea prelucrărilor care se efectuează au caracter numeric, fiind necesară o codificare numerică adecvată a problemei. Aceasta motivează prezența cuvântului *calcul* în denumirea domeniului. Pe de altă parte în fiecare dintre direcțiile de mai sus se încearcă simularea unor comportamente inteligente ceea ce motivează prezența termenului *inteligent*.

Principiul fundamental al calculului neuronal și al celui evolutiv este de a dezvolta sisteme de calcul inteligent pornind de la implementarea unor reguli simple, comportamentul complex al acestor sisteme derivând din aplicarea în paralel și în manieră interactivă a acestor reguli. Această abordare de tip "bottom-up" este în contrast cu abordarea de tip "top-down" specifică celorlalte domenii ale Inteligenței Artificiale.

Calculul neuronal și cel evolutiv fac parte din sfera mai largă a *calculului natural* al cărui principiu este de a prelua idei de rezolvare a problemelor din sistemele naturale (fizice, chimice, biologice, ecologice). Obiectivul principal al calculului natural este de a dezvolta metode de rezolvare a problemelor rău-puse și a celor pentru care metodele tradiționale nu sunt eficiente.

Pe lângă componentele amintite deja, calculul natural mai include *calculul molecular* (DNA Computing), *calculul cu membrane* (Membrane Computing) și *calculul cuantic* (Quantum Computing). Dacă primele două direcții (calculul neuronal și cel evolutiv) sunt deja tradiționale, ultimele trei sunt încă în primele faze de dezvoltare.

Sursa de inspirație	Model de calcul (soft)	Implementare (hard)
Creier	Calcul neuronal	Electronică
ADN	Calcul evolutiv	Electronică
ADN	Calcul molecular	Biologică
Celula	Calcul membranar	Electronică, biologică
Sistem fizic	Calcul cuantic	Nanoelectronică

## 2 Specificul calculului neuronal

Sistemele cu care se operează în calculul neuronal sunt *rețelele neuronale*. Acestea sunt sisteme artificiale concepute inițial pornind de la structura și funcționarea creierului fiind constituite din mai multe *unități funcționale interconectate*. Orice rețea neuronală este caracterizată prin:

- Tipul unităților funcționale componente.
- Arhitectura.
- Algoritmul de funcționare.
- Algoritmul de învățare.

**Unități funcționale.** O unitate funcțională dintr-o rețea neuronală (numită și neuron sau element de procesare) este un automat simplu care primește semnale de intrare (în formă numerică) și produce un semnal de ieșire tot în formă numerică (fig. 1). Funcționarea unităților depinde de un set de parametri ajustabili, numiți *ponderi*.

*Exemplu de unitate funcțională.* Fie  $x_1, \dots, x_n$  datele de intrare, iar  $w_0, w_1, \dots, w_n$  parametrii unității. Atunci răspunsul unității va fi:

$$y = f\left(\sum_{j=1}^n w_j x_j - w_0\right) = f\left(\sum_{j=0}^n w_j x_j\right) \quad \text{cu } x_0 = -1. \quad (1)$$

În relația de mai sus  $f$  este o funcție numită de *activare* sau de *transfer*, iar parametrii  $(w_j)_{j=0, \dots, n}$  reprezintă ponderile asociate conexiunilor. Parametrul  $w_0$  are o semnificație specială nefiind asociat unei conexiuni reale ci uneia fictive căruia îi corepunde semnalul de intrare  $x_0 = -1$ . În realitate el reprezintă *pragul de activare* al unității. Funcția care transformă vectorul semnalelor de intrare într-un scalar, se numește funcție de integrare (sau de agregare). În exemplul de mai sus ea este specificată prin  $g(x_0, x_1, \dots, x_n) = \sum_{j=0}^n w_j x_j$ .

Funcțiile de activare pot fi liniare ( $f(u) = u$ ) sau neliniare. Câteva exemple de funcții neliniare de activare sunt:

$$\begin{aligned} f_1(u) &= \begin{cases} 1, & \text{dacă } u > 0 \\ -1, & \text{dacă } u \leq 0 \end{cases} & f_2(u) &= \frac{\exp(2u) - 1}{\exp(2u) + 1} \\ f_3(u) &= \begin{cases} 1, & \text{dacă } u > 0 \\ 0, & \text{dacă } u \leq 0 \end{cases} & f_4(u) &= \frac{1}{1 + \exp(-u)} \end{aligned} \quad (2)$$

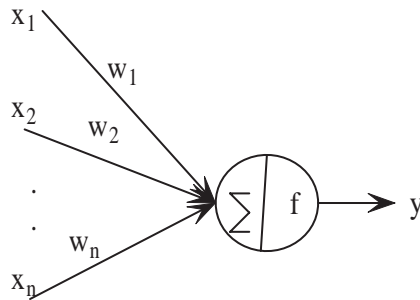


Figura 1: Exemplu de unitate funcțională

**Arhitectura.** Se referă la modul în care sunt amplasate și interconectate unitățile funcționale. Din punctul de vedere al arhitecturii, o rețea neuronală este un graf orientat etichetat în ale cărui noduri sunt amplasate unitățile funcționale și ale cărui arce descriu modul în care sunt conectate unitățile (cum se transferă informația între ele). Etichetele arcelor sunt ponderile conexiunilor către fiecare unitate și reprezintă principalele elemente adaptabile (supuse procesului de învățare) ale rețelei.

Din punctul de vedere al rolului pe care îl au unitățile funcționale în cadrul rețelei, ele pot fi descompuse în trei categorii principale:

- *Unități de intrare.* Primesc semnale din partea mediului. În cazul în care primesc semnale doar din exterior nu au alt rol decât de a retransmite semnalul primit către alte unități din rețea. În această situație nu sunt unități funcționale propriu-zise întrucât nu realizează nici o prelucrare asupra semnalului primit.

- *Unități ascunse.* Sunt conectate doar cu alte unități ale rețelei fără a comunica direct cu mediul extern. Rolul lor este de a colecta semnale, de a le prelucra și de a distribui semnalul de ieșire către alte unități.
- *Unități de ieșire.* Colectează semnale de la alte unități, le prelucreză și transmit semnalul pe care îl obțin mediului extern.

În unele rețele cele trei categorii de unități formează mulțimi distincte. Cazul cel mai frecvent întâlnit este cel al rețelelor organizate pe nivele: un nivel de unități de intrare, unul sau mai multe nivele de unități ascunse și un nivel de ieșire. O situație particulară o reprezintă rețelele ce nu conțin unități ascunse ci doar un nivel de unități de intrare și un nivel de unități de ieșire.

În alte rețele nu se face distincție netă între unitățile de intrare și cele de ieșire: toate unitățile preiau semnale din mediu, le prelucreză și transmit rezultatul atât unităților din rețea cât și mediului.

Modul de amplasare a unităților determină *topologia* rețelei. Din punctul de vedere al acesteia există:

- Rețele în care nu are importanță (din punctul de vedere al algoritmilor de funcționare și/sau de învățare) poziția geometrică a unităților. Astfel de topologii sunt asociate rețelelor organizate pe nivele (fig. 2) și rețelelor Hopfield (fig. 3). În reprezentările schematice ale rețelelor organizate pe nivele unitățile aceluiași nivel sunt reprezentate grupat deși poziția lor nu are semnificație pentru procesul de funcționare și cel de învățare.
- Rețele în care este esențială organizarea geometrică, relațiile de vecinătate dintre unități intervenind în algoritmul funcționare sau în cel de învățare. Astfel de topologii sunt cele asociate rețelelor Kohonen (fig. 4, fig. 5) sau rețelelor celulare (fig. 6). Esențial în acest caz este definirea unei relații de vecinătate între unități.

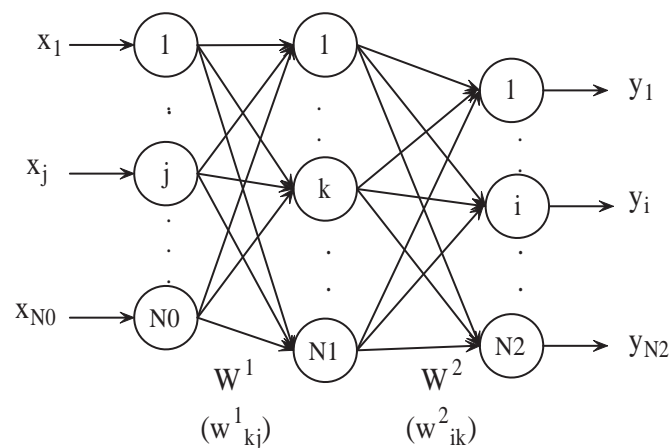


Figura 2: Exemplu de rețea cu un nivel ascuns

Modul de interconectare a unităților determină *fluxul de semnale* prin rețea fiind un factor esențial în stabilirea algoritmului de funcționare. Din perspectiva prezenței conexiunilor inverse în graful asociat, rețelele pot fi clasificate în:

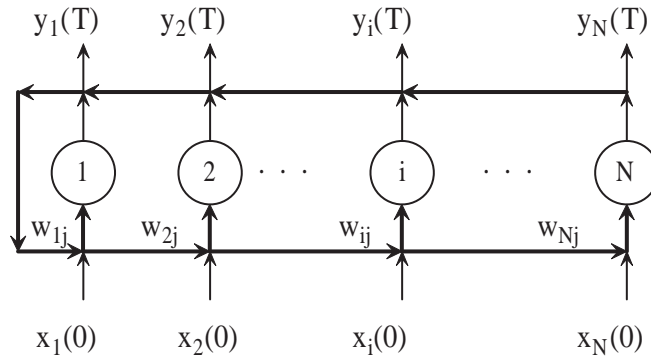


Figura 3: Exemplu de rețea cu conectivitate totală (rețea de tip Hopfield)

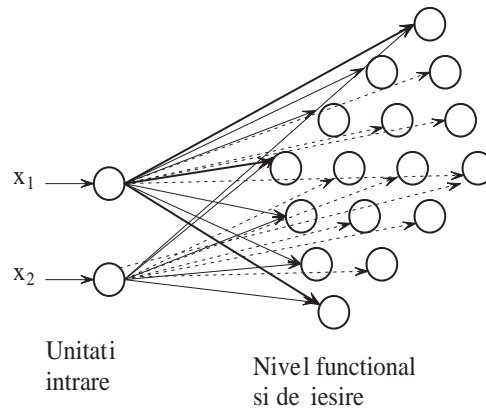


Figura 4: Exemplu de rețea cu organizare geometrică a nivelului de ieșire (rețea Kohonen)

- *Rețele de tip "feed-forward"*. Nu există conexiuni inverse, fluxul informațional fiind unidirecțional dinspre setul unităților de intrare către cel al unităților de ieșire. Conectivitatea între nivele poate fi totală sau locală. Exemple de astfel de rețele sunt ilustrate în fig. 7, 8, 9.
- *Rețele recurente*. Graful asociat conține conexiuni inverse directe (bucle) sau indirecte (circuite). Exemple de rețele cu conexiuni inverse sunt ilustrate în fig. 3 și fig. 10 .

**Funcționare.** Funcționarea se referă la modul în care rețeaua transformă un semnal de intrare,  $X$ , într-un semnal de ieșire,  $Y$ . Ea depinde atât de modul în care funcționează unitățile cât și de modul în care sunt interconectate. Unul dintre parametrii cei mai importanți ai funcționării este ansamblul ponderilor asociate tuturor conexiunilor ( $W$ ). La prima vedere, o rețea neuronală poate fi văzută ca o cutie neagră care primește date de intrare și produce un rezultat (figura 11).

În funcție de specificul fluxului informațional există două moduri principale de funcționare: *Funcționare neiterativă*. În cazul rețelelor în care fluxul informațional este de tip feedforward, semnalul de ieșire,  $Y$ , se poate obține prin aplicarea unei funcții,  $F_W$  (care depinde de parametrii rețelei), asupra semnalului de intrare,  $X$  (figura 12).

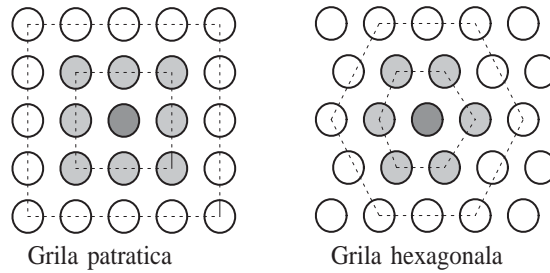


Figura 5: Tipuri de grile bidimensionale utilizate in algoritmi de învățare de la rețelele Kohonen

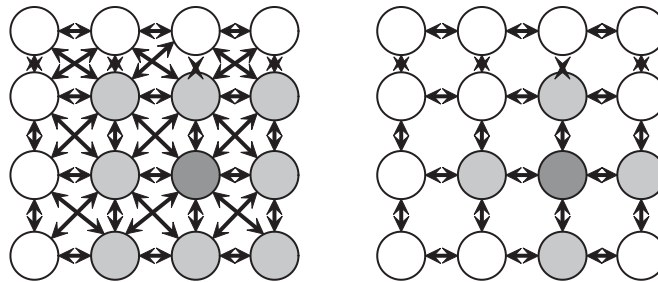


Figura 6: Exemple de rețele celulare

*Funcționare iterativă.* În cazul rețelelor în care sunt prezente conexiuni inverse nu este suficientă o singură "trecere" a semnalului de intrare,  $X$ , prin rețea pentru a obține semnalul de ieșire. Dimpotrivă, în acest caz funcționarea se desfășoară în timp, putând fi descrisă printr-un proces iterativ (fig. 13) de forma:

$$X(0) = X, \quad X(t + 1) = F_W(X(t)), \quad t \geq 0.$$

Semnalul de ieșire se consideră ca fiind limita lui  $X(t)$  ( $X^* = \lim_{t \rightarrow \infty} X(t)$ ). În implementări, limita se aproximează prin  $X(T)$ ,  $T$  fiind momentul în care este oprit procesul iterativ. Calitatea acestei aproximări depinde atât de proprietățile lui  $F_W$  cât și de  $X(0)$ .

**Învățare.** Învățarea poate fi văzută ca fiind un proces prin care un sistem își îmbunătățește performanțele prin achiziție de cunoaștere.

Pentru rețelele neuronale învățarea se referă la orice modificare a mulțimii parametrilor (ponderile asociate conexiunilor și eventual pragurile asociate unităților) care asigură o mai bună adecvare a comportării rețelei la problema pentru care a fost proiectată. Prin procesul de învățare se poate modifica și arhitectura rețelei (numărul de unități funcționale, modul de interconectare între ele etc.).

Capacitatea de a învăța este una dintre cele mai importante calități ale unei rețele neuronale, prin care aceasta poate surclasa alte metode. Posibilitatea învățării din exemple permite evitarea formalizării în detaliu a problemei de rezolvat, un avantaj important mai ales în cazul problemelor pentru care o astfel de formalizare nu există. Prin procesul de învățare în parametrii rețelei este înglobată implicit o formalizare a problemei, aceasta putând fi utilizată în faza de funcționare.

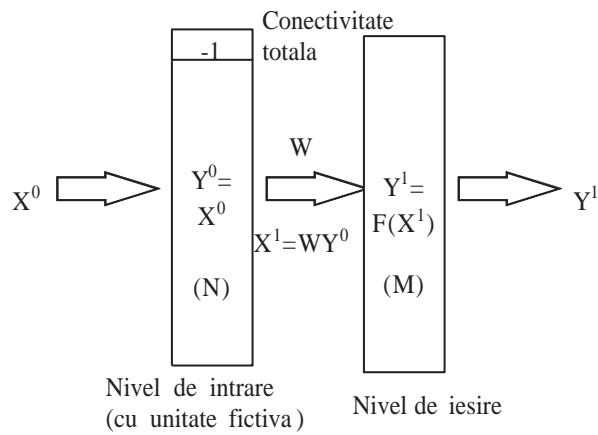


Figura 7: Reprezentare schematică a unei rețele cu un nivel de intrare ( $N+1$  unități) și un nivel de ieșire ( $M$  unități) și conectivitate totală între nivelul de intrare și cel de ieșire.

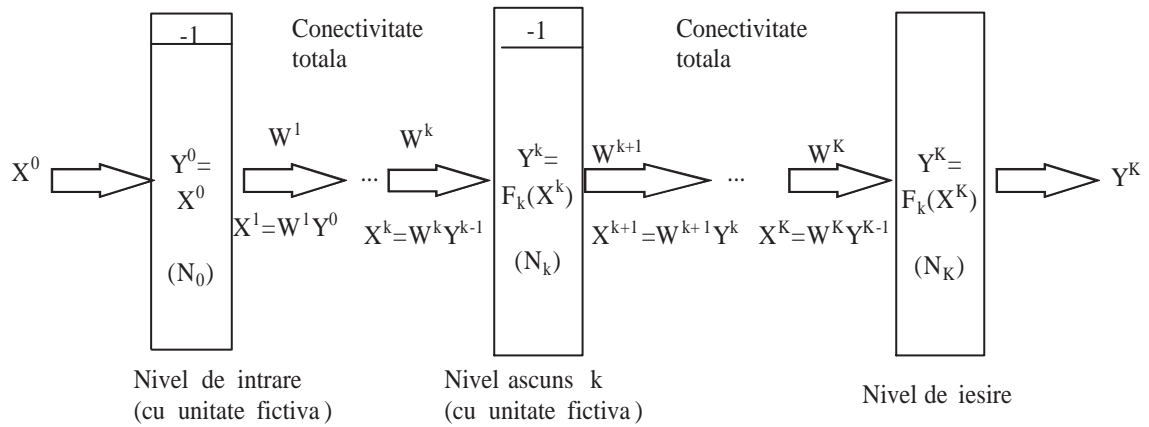


Figura 8: Reprezentare schematică a unei rețele cu un  $K$  nivele de unități funcționale și conectivitate totală între nivelele consecutive.

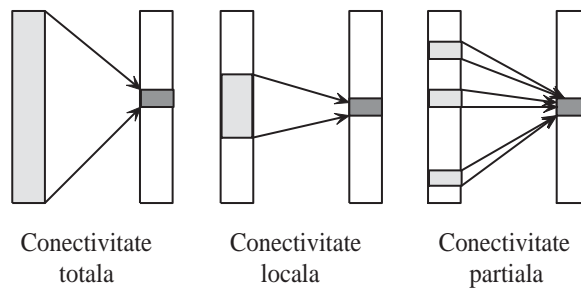


Figura 9: Variante de conectare a unităților de pe două nivele

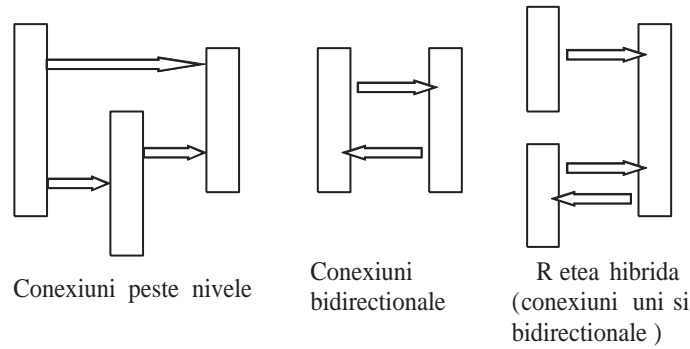


Figura 10: Variante de conectare între două sau mai multe nivele

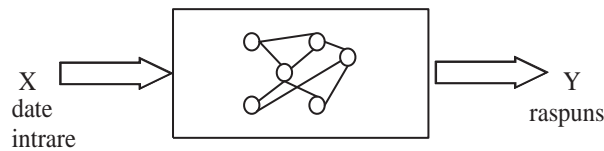


Figura 11: Rețeaua neuronală ca sistem de transformare a semnalelor de intrare în semnale de ieșire.

Un alt aspect important al rețelelor neuronale este *capacitatea de generalizare* adică de a produce răspunsuri și pentru date pentru care nu a fost antrenată.

Procesul de învățare se bazează pe două elemente:

- o mulțime de informații;
- un algoritm de adaptare la informațiile primite.

În funcție de natura informațiilor primite, învățarea poate fi de una dintre categoriile:

1. *Nesupervizată (auto-organizare)*. Sistemul primește doar semnale de intrare din partea mediului și pe baza acestora descoperă distribuția datelor de intrare construindu-și o reprezentare codificată în ponderi a mediului. Această reprezentare poate fi ulterior utilizată pentru alți stimuli care provin din partea aceluiași mediu. Din punct de vedere algoritmic o asemenea

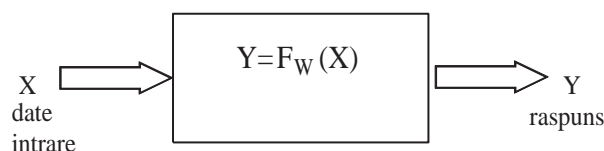


Figura 12: Funcționarea unei rețele cu flux de tip feedforward

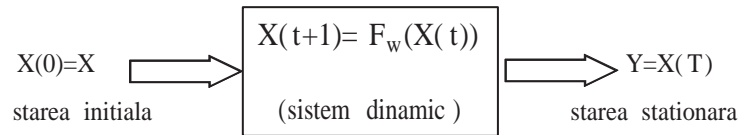


Figura 13: Funcționarea unei rețele recurente

metodă de adaptare constă într-un algoritm de construire a ponderilor care poate fi iterativ sau nu. Aceste metode sunt utilizate pentru aplicații de memorare asociativă, grupare a datelor (clustering), analiza componentelor principale.

2. *Supervizată*. Se dispune de un *set de antrenare* (mulțime de exemple) care conține perechi de forma  $(i, d)$  cu  $i$  reprezentând semnalul de intrare, iar  $d$  răspunsul corect (în cazul învățării supervizate propriu-zise) sau un indicator de corectitudine (în cazul celei de tip recompensă/penalizare). Pe baza setului de antrenare ponderile se construiesc iterativ urmărind maximizarea unui indice de performanță sau minimizarea unei funcții de eroare. Relativ la această metodă de adaptare apar și următoarele probleme:
  - (a) Testarea rețelei "antrenate" : se face de regulă prin reținerea din setul de antrenare a unui subset de testare (care nu este utilizat în determinarea ponderilor).
  - (b) Asigurarea unei bune capacități de generalizare: se menține un nivel *acceptabil* de eroare pe setul de antrenare în scopul evitării *supraînvățării* (învățarea detaliilor nesemnificative din cadrul exemplilor).

**Rezolvarea unei probleme folosind rețele neuronale.** Rezolvarea clasică (cu algoritmi bine precizați pentru fiecare clasă de probleme) a problemelor necesită cunoașterea a suficiente date despre problemă pentru a o putea descompune în unități logice elementare și pentru a elabora un algoritm care va rămâne "înghețat" în structura lui, modificându-se doar datele pe care le prelucrează. Dacă datele despre problemă nu sunt suficiente atunci problema nu poate fi formalizată și metoda de mai sus nu poate fi aplicată. În aceste situații pot fi utilizate rețele neuronale, etapele de rezolvare fiind:

1. Stabilirea unei arhitecturi inițiale care să fie compatibilă cu problema (de exemplu structura nivelului de intrare în rețea trebuie să fie compatibilă cu volumul datelor de intrare ale problemei) și alegerea tipului de unități funcționale. Stabilirea gradului de maleabilitate al rețelei prin specificarea parametrilor ajustabili (o rețea va fi cu atât mai generală cu cât va avea mai mulți parametri ajustabili). Pentru fiecare instanțiere a parametrilor se obține o anumită funcție asociată rețelei (două rețele având aceeași arhitectură dar valori diferite ale parametrilor pot rezolva două probleme diferite). În anumite cazuri chiar și arhitectura (de exemplu numărul de unități) este maleabilă ea fiind stabilită prin procesul de învățare.
2. Alegerea unui algoritm de învățare potrivit cu arhitectura rețelei și cu cantitatea de informație de care se dispune despre problemă. În alegerea algoritmului de învățare trebuie să se țină cont de:
  - (a) funcția pe care o poate realiza rețeaua (deci de arhitectură);

- (b) specificul "mediului informațional" al problemei (de volumul și natura datelor despre problemă).
- 3. Antrenarea rețelei pentru a rezolva o anumită problemă. Antrenarea se realizează prin "amplasarea" rețelei în "mediul informațional" specific problemei și activarea algoritmului de învățare.
- 4. Testarea (validarea) rețelei presupune verificarea corectitudinii răspunsurilor pe care le dă rețeaua când primește date de intrare care nu aparțin setului de antrenare dar pentru care se cunoaște răspunsul corect.
- 5. Utilizarea propriu-zisă a rețelei.

Primele două etape se referă la proiectarea rețelei iar celelalte la adaptarea (antrenarea) și utilizarea ei.

**Domenii de aplicabilitate a rețelelor neuronale.** Câteva dintre problemele ce pot fi abordate cu ajutorul rețelelor neuronale sunt:

- Probleme de clasificare și recunoaștere (gruparea și clasificarea datelor, recunoașterea scrisului și a vorbirii etc.).
- Probleme de aproximare și estimare (extragerea dependenței funcționale dintre două mărimi măsurate experimental, estimarea parametrilor unor modele asociate seriilor temporale etc);
- Probleme de modelare și control (modelarea sistemelor neliniare, controlul dispozitivelor autonome de tipul roboților etc.).
- Probleme de optimizare (proiectarea de circuite, probleme de rutare etc.).
- Prelucrarea și analiza semnalelor (filtrare, extragere de informații din imagini etc.)

### 3 Specificul calculului evolutiv

Calculul evolutiv oferă mecanisme de căutare în spațiul soluțiilor bazate pe principiile evoluției naturale. Pentru găsirea soluției se utilizează o *populație* de "agenți" de rezolvare a problemei. Această populație este supusă unui proces de evoluție caracterizat prin: *selectie, încrucișare, mutație*.

În funcție de modul în care este construită populația și de modul în care este implementată evoluția, sistemele de calcul evolutiv se încadrează în una dintre următoarele categorii:

- *Algoritmi genetici*: populația este reprezentată de stări din spațiul problemei care reprezintă soluții potențiale. De regulă, elementele populației sunt codificate în formă binară. Operatorul principal este cel de încrucișare, mutația având un rol secundar. Algoritmii genetici au fost introduși de Holland (1960) inițial ca modele ale evoluției și adaptării la mediu a sistemelor naturale. Ulterior s-a dovedit ca pot fi și modele eficiente de calcul, în special în rezolvarea problemelor de optimizare combinatorială.
- *Programare genetică*: populația este reprezentată de programe care candidează la rezolvarea problemei. Acestea sunt descrise mai degrabă ca arbori de derivare a cuvântului pe care îl reprezintă în cadrul limbajului în care este scris și nu ca linii de cod. "Programe" extrem de simple asociate unui calcul sunt expresiile. De exemplu, expresia "a+b\*c" poate fi descrisă în notație prefixată prin (+ a (\* b c)). O astfel de structură este descrisă ușor în Lisp, astfel

că în elaborarea sistemelor bazate pe programarea genetică Lisp-ul a fost frecvent utilizat. Încrucișarea este realizată selectând aleator subarbori din arborele asociat programelor părinte și interschimbându-le). Mutația este folosită relativ rar. Cel care a pus bazele programării genetice este Koza (1990).

- *Strategii evolutive*: au fost concepute inițial (Rechenberg, 1970) pentru a rezolva probleme de optimizare în tehnică (proiectarea construcțiilor, proiectarea avioanelor etc.). Populația este reprezentată de vectori cu valori reale (reprezentate în virgulă flotantă) iar principalul operator este cel de mutație (văzut ca o perturbație aleatoare). Încrucișarea este o operație utilă însă nu este obligatorie. Pentru acești algoritmi au fost dezvoltate pentru prima dată metode de stabilire a unor parametri de control prin auto-adaptare.
- *Programare evolutivă*: populația este reprezentată de automate cu stări finite fiecare dintre ele candidând la rezolvarea problemei abordate. Operatorul principal este cel de mutație, constând în perturbarea diagramei de tranziție a stărilor. Această direcție a fost inițiată de către Fogel (1966). Ulterior tehnicile de programare evolutivă s-au orientat mai mult înspre rezolvarea problemelor de optimizare folosind ca instrument de explorare a spațiului soluțiilor potențiale doar un operator de mutație (fără a folosi recombinare).

Inițial aceste direcții au evoluat separat fără a se influența reciproc. În ultimii ani însă s-a realizat un transfer de idei între diversele variante de sisteme evolutive astfel încât la ora actuală distincția dintre ele nu mai este atât de netă. În continuare prin *algoritm evolutiv* vom înțelege orice algoritm în care intervin prelucrări specifice oricăreia dintre categoriile de mai sus.

În fiecare dintre variantele enumerate evoluția este controlată prin intermediul unei funcții de *performanță* ("fitness") care măsoară gradul de adecvare a fiecărui individ la *mediul* din care face parte. Un individ este cu atât mai adecvat cu cât se apropie mai mult de soluția problemei.

De exemplu, în cazul rezolvării unei probleme de optimizare funcția "fitness" este determinată de funcția obiectiv a problemei.

Cu cât gradul de adecvare a unui element este mai mare cu atât șansele ca el să fie selectat pentru a participa la constituirea noii generații, deci de a supraviețui direct sau prin urmași este mai mare. Majoritatea algoritmilor evolutivi au caracter iterativ (figura 14) constând în aplicarea succesivă a operatorilor specifici până când populația satisface anumite proprietăți (specifice problemei de rezolvat) sau a fost parcurs un număr maxim de iterații (generații).

---

Inițializarea populației  $P(0) = \{x_1(0), \dots, x_m(0)\}$

Inițializarea contorului de generație:  $t = 0$

REPETĂ

EVALUARE  $P(t)$

Selecție părinți  $P(t) \rightarrow P'(t)$

Recombinare părinți  $P'(t) \rightarrow P''(t)$

Mutație  $P''(t) \rightarrow P'''(t)$

Evaluare  $P'''(t)$

Selecție noua populație  $\{P'''(t), P(t)\} \rightarrow P(t)$

PÂNĂ CÂND < este satisfăcută o condiție de oprire >

---

Figura 14: Structura generală a unui algoritm evolutiv.

**Etapele rezolvării unei probleme utilizând algoritmi evolutivi.** Etapele generice ale proiectării unui algoritm evolutiv sunt:

1. Stabilirea modului de codificare a elementelor populației.
2. Stabilirea modului de evaluare a elementelor populației.
3. Alegerea operatorilor evolutivi și a parametrilor asociați.

**Domenii de aplicabilitate.** La fel ca și rețelele neuronale, metodele evolutive se utilizează atunci când problema nu este bine formalizată și/sau nu există altă strategie de rezolvare. Principalele aplicații ale calculului evolutiv sunt:

- Optimizare neliniară multidimensională (funcții obiectiv cu multe puncte de optim, pentru care nu trebuie impuse ipoteze de netezime).
- Probleme de optimizare combinatorială (de exemplu, problema comis voiajorului, problema colorării grafurilor, problema alocării resurselor etc. ) cu aplicații în tehnică (de exemplu proiectarea automată a circuitelor electronice).
- Probleme de căutare în volume mari de informații (de exemplu în ingineria genetică se pune problema găsirii unei proteine cu proprietăți specifice în cadrul unui lanț de aminoacizi)
- Probleme de predicție (de exemplu predicția evoluției unui activ financiar sau a cursului valutar).
- Probleme de planificare a activităților (generarea automată a orarelor sau planificarea sarcinilor ce vor fi desfășurate de către un dispozitiv industrial).
- Programare automată (generarea unor programe care să rezolve sarcini specifice și generarea unor structuri computaționale cum sunt automatele celulare și rețelele de sortare).
- Prelucrarea imaginilor (proiectarea filtrelor pentru imagini și analiza imaginilor).
- Proiectarea rețelelor neuronale (stabilirea arhitecturii și/sau a ponderilor).
- Modelarea unor sisteme sau procese din biologie și ecologie (modelarea sistemului imunitar și a fenomenelor ecologice de tip: simbioză și coevoluție în sisteme de tip gazdă-parazit, migrații etc.).
- Modelarea unor sisteme sociale și extrapolarea rezultatelor la sisteme multi-agent (comportamentul social al unor colonii de insecte, probleme de cooperare și comunicare).
- Simularea unor activități creative (generare de imagini, muzică etc.)