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 întrare (valori iniţiale, ipoteze etc.) şi răspunsul corect. În cazul problemelor bine-puse această asociere este o relaţie funcţională explicită construită pe baza modelului asociat problemei.

În cazul problemelor rău-puse, însă, o astfel de relaţie explicită nu poate fi pusă în evidenţă, rolul sistemului care rezolvă problema fiind de a dezvolta o relaţie de asociere întrebare-răspuns pe baza unor exemple. Procesul prin care sistemul îsi formează modelul propriu al problemei şi pe baza acestuia relaţia de asociere se numeşte *învăţare*.

Pe de altă parte, din punctul de vedere al complexităţii rezolvării şi al relevanţei răspunsului problemele pot fi clasificate în:

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

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

Principalele direcţii ale calculului inteligent sunt:

- *Calcul neuronal.* Este folosit în principal în rezolvarea problemelor de asociere, bazându-se pe extragerea, prin învăţare, a unui model pornind de la exemple. Sursa de inspiraţie o reprezintă structura şi funcţionarea creierului.
- *Calcul evolutiv.* Este folosit în principal în rezolvarea problemelor bazate pe căutarea soluţiei într-un spaţiu mare de soluţii potenţiale. Sursa de inspiraţie o reprezintă principiile evoluţionismului darwinist.

- *Calcul fuzzy.* Este folosit atunci când datele problemei (şi relaţiile dintre acestea) nu pot fi descrise exact ci se caracterizează prin prezenţa unui grad de incertitudine ("fuzziness"). Ideea de bază este de a înlocui valorile exacte ("crisp") cu valori "fuzzy" descrise prin funcţii de apartenenţă.

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

Principiul fundamental al calculului neuronal şi 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 şi 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 Inteligenţei Artificiale.

Calculul neuronal şi 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 şi a celor pentru care metodele tradiţionale nu sunt eficiente.

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

Sursa de inspiraţie	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 *reţelele neuronale*. Acestea sunt sisteme artificiale concepute iniţial pornind de la structura şi funcţionarea creierului fiind constituite din mai multe *unităţi funcţionale interconectate*. Orice reţea neuronală este caracterizată prin:

- Tipul unităţilor funcţionale componente.
- Arhitectura.
- Algoritmul de funcţionare.
- Algoritmul de învăţare.

Unităţi funcţionale. O unitate funcţională dintr-o reţea neuronală (numită şi neuron sau element de procesare) este un automat simplu care primeşte semnale de intrare (în formă numerică) şi produce un semnal de ieşire tot în formă numerică (fig. 1). Funcţionarea unităţilor depinde de un set de parametri ajustabili, numiţi *ponderi*.

Exemplu de unitate funcţională. Fie $x_1,\ldots,x_n$ datele de intrare, iar $w_0,w_1,\ldots,w_n$ parametrii unităţii. Atunci răspunsul unităţii va fi:

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

În relaţia de mai sus f este o funcţie numită de *activare* sau de *transfer*, iar parametrii $(w_j)_{j=\overline{0,n}}$ reprezintă ponderile asociate conexiunilor. Parametrul w_0 are o semnificaţie 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 unităţii. Funcţia care transformă vectorul semnalelor de intrare într-un scalar, se numeşte funcţie de integrare (sau de agregare). În exemplul de mai sus ea este specificată prin $g(x_0, x_1, \ldots, x_n) = \sum_{j=0}^{n} w_j x_j$.

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

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

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

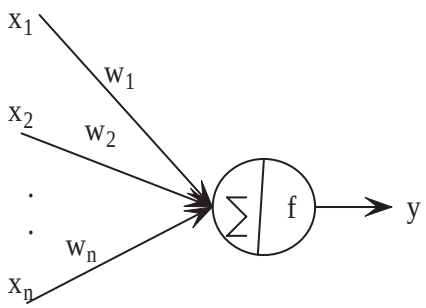

Figura 1: Exemplu de unitate functionala

Arhitectura. Se referă la modul în care sunt amplasate şi interconectate unităţile funcţionale. Din punctul de vedere al arhitecturii, o reţea neuronală este un graf orientat etichetat în ale cărui noduri sunt amplasate unităţile funcţionale şi ale cărui arce descriu modul în care sunt conectate unităţile (cum se transferă informaţia între ele). Etichetele arcelor sunt ponderile conexiunilor către fiecare unitate şi reprezintă principalele elemente adaptabile (supuse procesului de învăţare) ale reţelei.

Din punctul de vedere al rolului pe care îl au unităţile funcţionale în cadrul reţelei, ele pot fi descompuse în trei categorii principale:

- *Unităţi 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 unităţi din reţea. În această situaţie nu sunt unităţi funcţionale propriu-zise întrucât nu realizează nici o prelucrare asupra semnalului primit.

- *Unităţi ascunse.* Sunt conectate doar cu alte unităţi ale reţelei fără a comunica direct cu mediul extern. Rolul lor este de a colecta semnale, de a le prelucra şi de a distribui semnalul de ieşire către alte unităţi.

- *Unităţi de ieşire.* Colectează semnale de la alte unităţi, le prelucrează şi transmit semnalul pe care îl obţin mediului extern.

În unele reţele cele trei categorii de unităţi formează mulţimi distincte. Cazul cel mai frecvent întâlnit este cel al reţelelor organizate pe nivele: un nivel de unităţi de intrare, unul sau mai multe nivele de unităţi ascunse şi un nivel de ieşire. O situaţie particulară o reprezintă reţelele ce nu conţin unitaţi ascunse ci doar un nivel de unităţi de intrare şi un nivel de unităţi de ieşire.

În alte reţele nu se face distincţie netă între unităţile de intrare şi cele de ieşire: toate unităţile preiau semnale din mediu, le prelucrează şi transmit rezultatul atât unităţilor din reţea cât şi mediului.

Modul de amplasare a unităţilor determină *topologia* reţelei. Din punctul de vedere al acesteia există:

- Reţele în care nu are importanţă (din punctul de vedere al algoritmilor de funcţionare şi/sau de învăţare) poziţia geometrică a unităţilor. Astfel de topologii sunt asociate reţelelor organizate pe nivele (fig. 2) şi reţelelor Hopfield (fig. 3). În reprezentările schematice ale reţelelor organizate pe nivele unităţile aceluiaşi nivel sunt reprezentate grupat deşi poziţia lor nu are semnificaţie pentru procesul de funcţionare şi cel de învăţare.

- Reţele în care este esenţială organizarea geometrică, relaţiile de vecinătate dintre unităţi intervenind în algoritmul funcţionare sau în cel de învăţare. Astfel de topologii sunt cele asociate reţelelor Kohonen (fig. 4, fig. 5) sau reţelelor celulare (fig. 6). Esenţial în acest caz este definirea unei relaţii de vecinătate între unităţi.

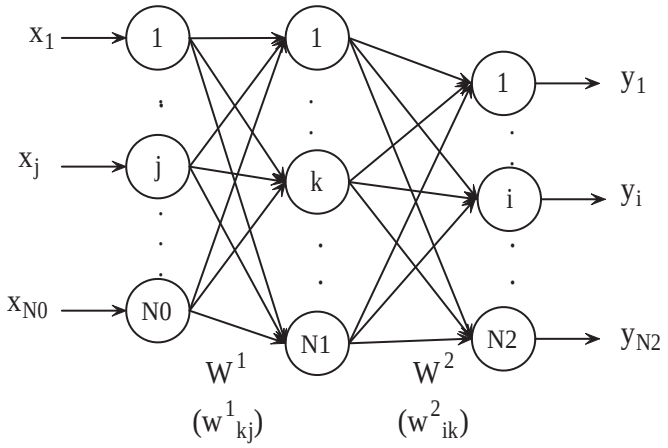

Figura 2: Exemplu de reţea cu un nivel ascuns

Modul de interconectare a unităţilor determină *fluxul de semnale* prin reţea fiind un factor esenţial în stabilirea algoritmului de funcţionare. Din perspectiva prezenţei conexiunilor inverse în graful asociat, reţelele pot fi clasificate în:

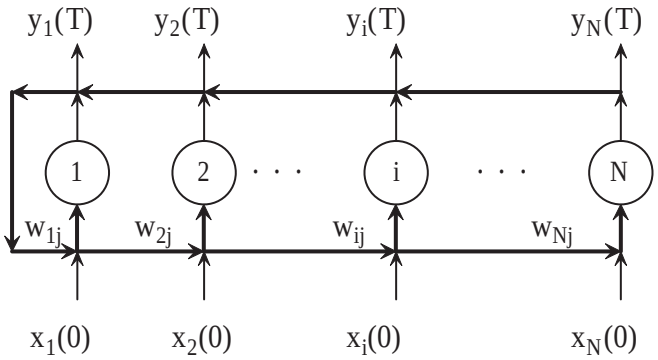

Figura 3: Exemplu de reţea cu conectivitate totală (reţea de tip Hopfield)

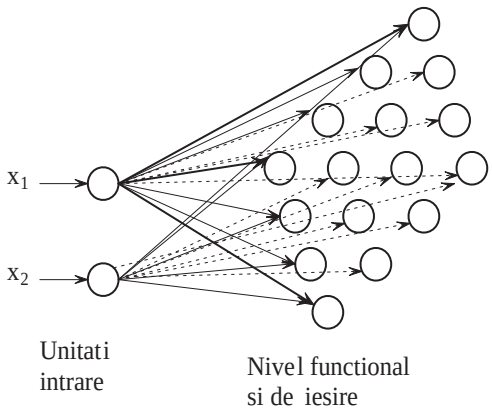

Figura 4: Exemplu de reţea cu organizare geometrică a nivelului de ieşire (reţea Kohonen)

- *Reţele de tip "feed-forward".* Nu există conexiuni inverse, fluxul informaţional fiind unidirecţional dinspre setul unităţilor de intrare către cel al unităţilor de ieşire. Conectivitatea între nivele poate fi totală sau locală. Exemple de astfel de reţele sunt ilustrate in fig. 7, 8, 9.
- *Reţele recurente.* Graful asociat conţine conexiuni inverse directe (bucle) sau indirecte (circuite). Exemple de reţele cu conexiuni inverse sunt ilustrate în fig. 3 şi fig. 10 .

Funcţionare. Funcţionarea se referă la modul în care reţeaua transformă un semnal de intrare, X, într-un semnal de ieşire, Y. Ea depinde atât de modul în care funcţionează unităţile cât şi de modul în care sunt interconectate. Unul dintre parametrii cei mai importanţi ai funcţionării este ansamblul ponderilor asociate tuturor conexiunilor (W). La prima vedere, o reţea neuronală poate fi văzută ca o cutie neagră care primeşte date de intrare şi produce un rezultat (figura 11).

În funcţie de specificul fluxului informaţional există două moduri principale de funcţionare: *Funcţionare neiterativă.* În cazul reţelelor în care fluxul informaţional este de tip feedforward, semnalul de ieşire, Y, se poate obţine prin aplicarea unei funcţii, F_W (care depinde de parametrii reţelei) , asupra semnalului de intrare, X (figura 12).

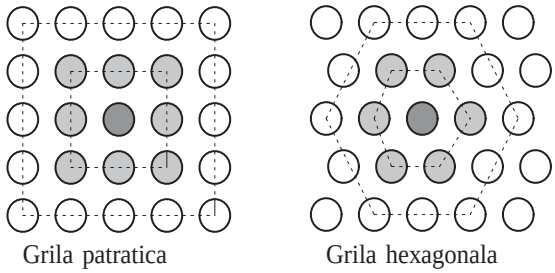

Figura 5: Tipuri de grile bidimensionale utilizate in algoritmii de învăţare de la reţelele Kohonen

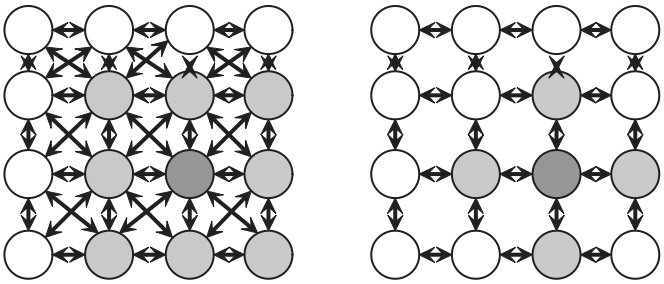

Figura 6: Exemple de reţele celulare

Funcţionare iterativă. În cazul reţelelor în care sunt prezente conexiuni inverse nu este suficientă o singură "trecere" a semnalului de intrare, X, prin reţea pentru a obţine semnalul de ieşire. Dimpotrivă, în acest caz funcţionarea se desfăşoară în timp, putând fi descrisă printr-un proces iterativ (fig. 13) de forma:

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

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

Învăţare. Învăţarea poate fi văzută ca fiind un proces prin care un sistem îşi îmbunătăţeşte performanţele prin achiziţie de cunoaştere.

Pentru reţelele neuronale învăţarea se referă la orice modificare a mulţimii parametrilor (ponderile asociate conexiunilor şi eventual pragurile asociate unităţilor) care asigură o mai bună adecvare a comportării reţelei la problema pentru care a fost proiectată. Prin procesul de învăţare se poate modifica şi arhitectura reţelei (numărul de unităţi funcţionale, modul de interconectare între ele etc.).

Capacitatea de a învăţa este una dintre cele mai importante calităţi ale unei reţele neuronale, prin care aceasta poate surclasa alte metode. Posibilitatea învăţării 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 învăţare în parametrii reţelei este înglobată implicit o formalizare a problemei, aceasta putând fi utilizată în faza de funcţionare.

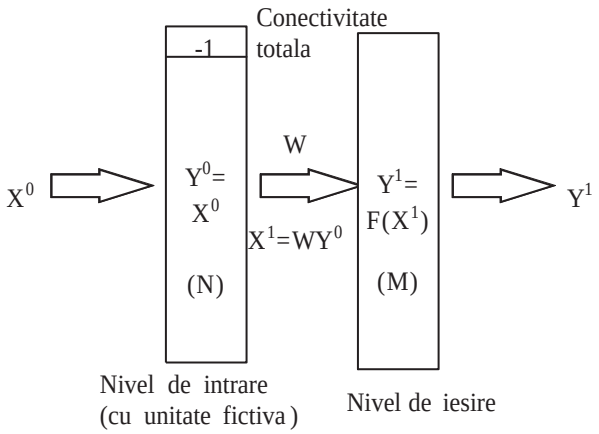

Figura 7: Reprezentare schematică a unei reţele cu un nivel de intrare (N+1 unităţi) şi un nivel de ieşire (M unităţi) şi conectivitate totală între nivelul de intrare şi cel de ieşire.

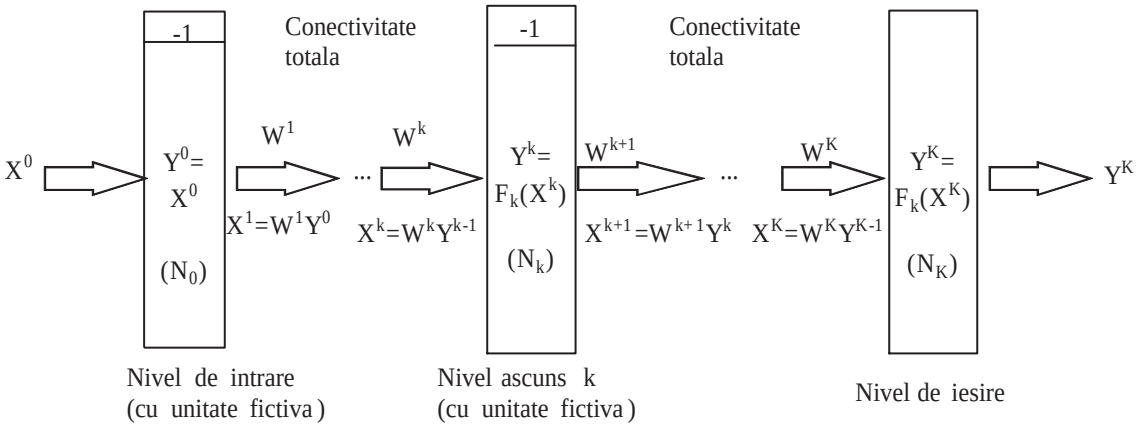

Figura 8: Reprezentare schematică a unei reţele cu un K nivele de unităţi funcţionale şi conectivitate totala între nivelele consecutive.

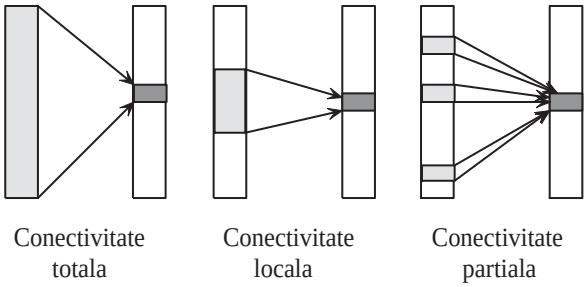

Figura 9: Variante de conectare a unităţilor de pe două nivele

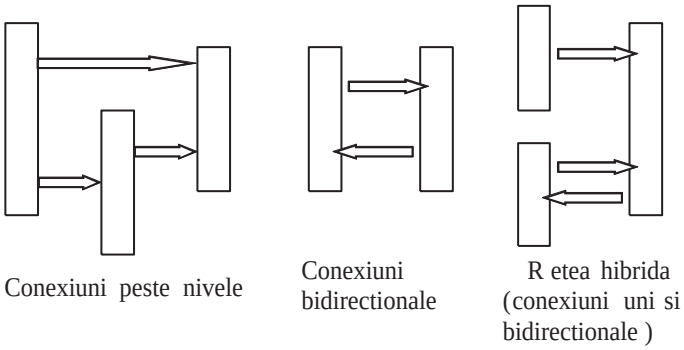

Figura 10: Variante de conectare între două sau mai multe nivele

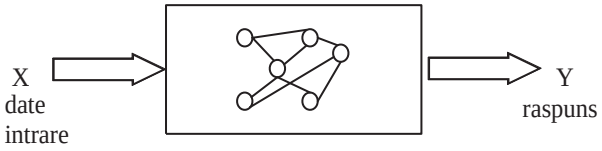

Figura 11: Reţeaua neuronală ca sistem de transformare a semnalelor de intrare în semnale de iesîre.

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

Procesul de învăţare se bazează pe două elemente:

- o mulţime de informaţii;
- un algoritm de adaptare la informaţiile primite.

În funcţie de natura informaţiilor primite, învăţarea poate fi de una dintre categoriile:

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

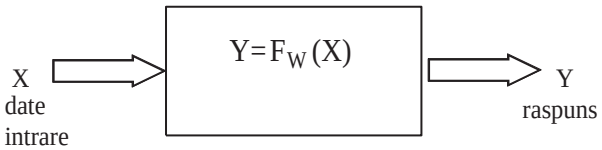

Figura 12: Funcţionarea unei reţele cu flux de tip feedforward

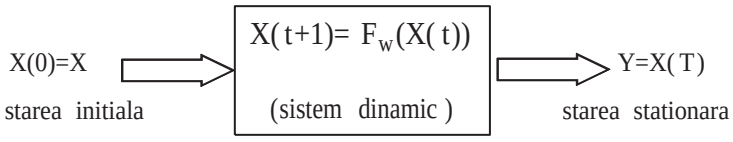

Figura 13: Funcţionarea unei reţele recurente

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

2. *Supervizată.* Se dispune de un *set de antrenare* (mulţime de exemple) care conţine perechi de forma (i, d) cu i reprezentând semnalul de intrare, iar d răspunsul corect (în cazul învăţării 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 performanţă sau minimizarea unei funcţii de eroare. Relativ la această metodă de adaptare apar şi următoarele probleme:

   (a) Testarea reţelei "antrenate" : se face de regulă prin reţinerea din setul de antrenare a unui subset de testare (care nu este utilizat în determinarea ponderilor).

   (b) Asigurarea unei bune capacităţi de generalizare: se menţine un nivel *acceptabil* de eroare pe setul de antrenare în scopul evitării *supraînvăţării* (învăţarea detaliilor nesemnificative din cadrul exemplelor).

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

1. Stabilirea unei arhitecturi iniţiale care să fie compatibilă cu problema (de exemplu structura nivelului de intrare în reţea trebuie să fie compatibilă cu volumul datelor de intrare ale problemei) şi alegerea tipului de unităţi funcţionale. Stabilirea gradului de maleabilitate al reţelei prin specificarea parametrilor ajustabili (o reţea va fi cu atât mai generală cu cât va avea mai mulţi parametri ajustabili). Pentru fiecare instanţiere a parametrilor se obţine o anumită funcţie asociată reţelei (două reţele având aceeaşi arhitectură dar valori diferite ale parametrilor pot rezolva două probleme diferite). În anumite cazuri chiar şi arhitectura (de exemplu numărul de unităţi) este maleabilă ea fiind stabilită prin procesul de învăţare.

2. Alegerea unui algoritm de învăţare potrivit cu arhitectura reţelei şi cu cantitatea de informaţie de care se dispune despre problemă. În alegerea algoritmului de învăţare trebuie să se ţină cont de:

   (a) funcţia pe care o poate realiza reţeaua (deci de arhitectură);

(b) specificul "mediului informaţional" al problemei (de volumul şi natura datelor despre problemă).

3. Antrenarea reţelei pentru a rezolva o anumită problemă. Antrenarea se realizează prin "amplasarea" reţelei în "mediul informaţional" specific problemei şi activarea algoritmului de învăţare.

4. Testarea (validarea) reţelei presupune verificarea corectitudinii răspunsurilor pe care le dă reţeaua când primeşte date de intrare care nu aparţin setului de antrenare dar pentru care se cunoaşte răspunsul corect.

5. Utilizarea propriu-zisă a reţelei.

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

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

- Probleme de clasificare şi recunoaştere (gruparea şi clasificarea datelor, recunoaşterea scrisului şi a vorbirii etc.).
- Probleme de aproximare şi estimare (extragerea dependenţei funcţionale dintre două mărimi măsurate experimental, estimarea parametrilor unor modele asociate seriilor temporale etc);
- Probleme de modelare şi control (modelarea sistemelor neliniare, controlul dispozitivelor autonome de tipul roboţilor etc.).
- Probleme de optimizare (proiectarea de circuite, probleme de rutare etc.).
- Prelucrarea şi analiza semnalelor (filtrare, extragere de informaţii din imagini etc.)

3 Specificul calculului evolutiv

Calculul evolutiv oferă mecanisme de căutare în spaţiul soluţiilor bazate pe principiile evoluţiei naturale. Pentru găsirea soluţiei se utilizează o *populaţie* de "agenţi" de rezolvare a problemei. Această populaţie este supusă unui proces de evoluţie caracterizat prin: *selecţie*, *încrucişare*, *mutaţie*.

În funcţie de modul în care este construită populaţia şi de modul în care este implementată evoluţia, sistemele de calcul evolutiv se încadrează în una dintre următoarele categorii:

- *Algoritmi genetici:* populaţia este reprezentată de stări din spaţiul problemei care reprezintă soluţii potenţiale. De regulă, elementele populaţiei sunt codificate în formă binară. Operatorul principal este cel de încrucişare, mutaţia având un rol secundar. Algoritmii genetici au fost introduşi de Holland (1960) iniţial ca modele ale evoluţiei şi adaptării la mediu a sistemelor naturale. Ulterior s-a dovedit ca pot fi şi modele eficiente de calcul, in special în rezolvarea problemelor de optimizare combinatorială.
- *Programare genetică:* populaţia 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 şi nu ca linii de cod. "Programe" extrem de simple asociate unui calcul sunt expresiile. De exemplu, expresia "a+b*c" poate fi descrisă în notaţie prefixată prin (+ a (* b c)). O astfel de structură este descrisă uşor în Lisp, astfel

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

- *Strategii evolutive:* au fost concepute iniţial (Rechenberg, 1970) pentru a rezolva probleme de optimizare în tehnică (proiectarea construcţiilor, proiectarea avioanelor etc.). Populaţia este reprezentată de vectori cu valori reale (reprezentate în virgulă flotantă) iar principalul operator este cel de mutaţie (văzut ca o perturbaţie aleatoare). Încrucişarea este o operaţie utilă însă nu este obligatorie. Pentru aceşti algoritmi au fost dezvoltate pentru prima dată metode de stabilire a unor parametri de control prin auto-adaptare.

- *Programare evolutivă:* populaţia este reprezentată de automate cu stări finite fiecare dintre ele candidând la rezolvarea problemei abordate. Operatorul principal este cel de mutaţie, constând în perturbarea diagramei de tranziţie a stărilor. Această direcţie a fost iniţiată 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 spaţiului soluţiilor potenţiale doar un operator de mutaţie (fără a folosi recombinare).

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

În fiecare dintre variantele enumerate evoluţia este controlată prin intermediul unei funcţii de *performanţă* ("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 soluţia problemei.

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

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

Iniţializarea populaţiei $P(0) = \{x_1(0), \ldots, x_m(0)\}$
Iniţializarea contorului de generaţie: $t = 0$
REPETĂ
  EVALUARE $P(t)$
  Selecţie părinţi $P(t) \rightarrow P'(t)$
  Recombinare părinţi $P'(t) \rightarrow P''(t)$
  Mutaţie $P''(t) \rightarrow P'''(t)$
  Evaluare $P'''(t)$
  Selecţie noua populaţie $\{P'''(t), P(t)\} \rightarrow P(t)$
PÂNĂ CÂND < este satisfăcută o condiţie 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 populaţiei.
2. Stabilirea modului de evaluare a elementelor populaţiei.
3. Alegerea operatorilor evolutivi şi a parametrilor asociaţi.

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

- Optimizare neliniară multidimensională (funcţii 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 aplicaţii în tehnică (de exemplu proiectarea automată a circuitelor electronice).
- Probleme de căutare în volume mari de informaţii (de exemplu în ingineria genetică se pune problema găsirii unei proteine cu proprietăţi specifice în cadrul unui lanţ de aminoacizi)
- Probleme de predicţie (de exemplu predicţia evoluţiei unui activ financiar sau a cursului valutar).
- Probleme de planificare a activităţilor (generarea automată a orarelor sau planificarea sarcinilor ce vor fi desfăşurate de către un dispozitiv industrial).
- Programare automată (generarea unor programe care să rezolve sarcini specifice şi generarea unor structuri computaţionale cum sunt automatele celulare şi reţelele de sortare).
- Prelucrarea imaginilor (proiectarea filtrelor pentru imagini şi analiza imaginilor).
- Proiectarea reţelelor neuronale (stabilirea arhitecturii şi/sau a ponderilor).
- Modelarea unor sisteme sau procese din biologie şi ecologie (modelarea sistemului imunitar şi a fenomenelor ecologice de tipul: simbioză şi coevoluţie în sisteme de tip gazdă-parazit, migraţii etc.).
- Modelarea unor sisteme sociale şi extrapolarea rezultatelor la sisteme multi-agent (comportamentul social al unor colonii de insecte, probleme de cooperare şi comunicare).
- Simularea unor activităţi creative (generare de imagini, muzică etc.)