Proiectarea evolutivă a rețelelor neuronale

- Motivație
- Determinarea evolutivă a ponderilor (antrenare evolutivă)
- Determinarea evolutivă a arhitecturii
- Determinarea evolutivă a regulilor de învățare

Proiectarea evolutivă a rețelelor neuronale

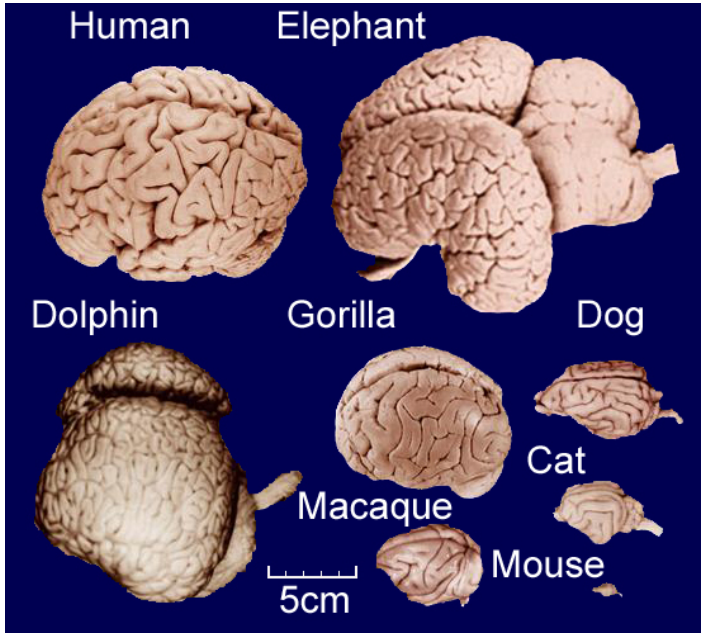
Motivație. Proiectarea unei rețele neuronale presupune:

- Stabilirea arhitecturii rețelei (număr unități funcționale + mod de interconectare+funcții de activare)
  - Influențează abilitatea rețelei de a rezolva problema
  - Presupune încercarea mai multor variante (abordare de tip trial and error)

- Determinarea valorilor parametrilor ajustabili (antrenare)
  - Este o problemă de optimizare = determinarea parametrilor care minimizează eroarea pe setul de antrenare/validare
  - Metodele clasice (ex: BackPropagation) întâmpină dificultăți:
    - Dacă funcțiile de activare nu sunt diferențiabile
    - Se pot bloca în optime locale

Proiectarea evolutivă a rețelelor neuronale

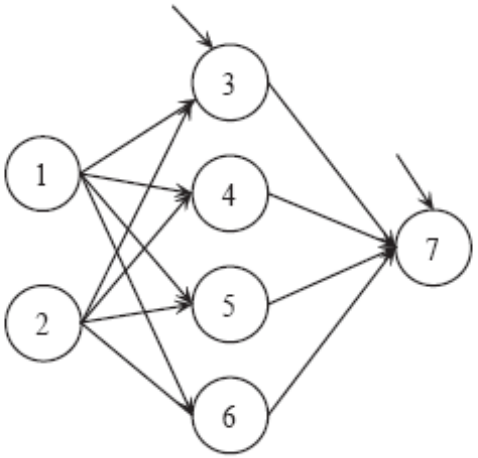
Idee: utilizarea unui proces de evoluție



- inspirată de dezvoltarea biologică a creierului
- sistemul nu este proiectat in mod explicit ci structura sa este determinată printr-un proces de evoluție la care participă o populație de rețele neuronale descrise codificat
  - Genotip = codificarea rețelei (descriere structurală)
  - Fenotip = rețeaua propriu-zisă, care poate fi simulată (descriere funcțională)

Antrenare evolutivă

- Se referă la determinarea ponderilor unei rețele neuronale cu arhitectură fixată rezolvând problema minimizării unei funcții de eroare pe setul de antrenare folosind un algoritm evolutiv



- Set de parametri: ponderi sinaptice și praguri

$$\text{Set antrenare}: \{(x^1, d^1), ..., (x^L, d^L)\}$$

$$\text{Functie eroare}: E(W) = \frac{1}{L}\sum_{l=1}^{L}(d^l - y^l)^2$$

$$W = \{w_{31}, w_{32}, w_{30}, w_{41}, w_{42}, w_{40}, ..., w_{70}, w_{73}, ..., w_{76}\}$$

Antrenare evolutivă

Algoritm evolutiv:

- Codificare: vector de parametri cu valori reale ce conține toate ponderile conexiunilor din rețea (similar cu strategiile evolutive)
- Operatori evolutivi: specifici strategiilor evolutive (ex: recombinare convexă și mutație prin perturbație aleatoare)
- Evaluare elemente: calcul funcție de eroare pe setul de antrenare; un element e cu atât mai bun cu cât valoarea erorii pe setul de antrenare este mai mică

Antrenare evolutivă

Aplicabilitate:

- In cazul rețelelor cu funcții de transfer nederivabile (algoritmul BackPropagation nu poate fi aplicat)
- In cazul rețelelor recurente antrenate supervizat (algoritmul Backpropagation nu poate fi aplicat întrucât nu există o dependență explicită între semnalul de ieșire și cel de intrare, astfel că nu pot fi calculate derivatele parțiale necesare aplicării algoritmului)

Dezavantaje:

- antrenarea evolutivă este mai costisitoare din punct de vedere computațional decât cea neevolutivă
- Este adecvată pentru explorarea spațiului ponderilor, fiind mai puțin adecvată pentru căutarea locală

Antrenare evolutivă

Varianta hibride:

- Antrenarea inițială cu o strategie evolutivă
- Rafinarea valorilor parametrilor folosind un algoritm de optimizare locala (e.g. BackPropagation – dacă natura problemei permite)

Pregatirea (pre-procesarea) setului de antrenare (filtrarea în manieră evolutivă a setului de antrenare):

- Selecția atributelor
- Selecția exemplelor

Antrenare evolutivă

Selecția atributelor (în cazul problemelor de clasificare):

- Motivație: dacă numărul de atribute este mare rețeaua conține multe unități de intrare iar antrenarea poate deveni mai dificilă
- este importantă în cazul datelor ce conțin multe atribute dintre care unele sunt irelevante pentru procesul de clasificare
- se selectează atributele relevante pentru procesul de clasificare
- în cazul unor date inițiale conținând N atribute se folosește o codificare binară în care un element este setat pe 0 dacă atributul corespunzător nu trebuie selectat și pe 1 dacă atributul trebuie selectat
- Elementele populației se evaluează prin atrenarea rețelei pentru setul de antrenare astfel filtrat (antrenarea se face cu un algoritm adecvat problemei – nu neapărat evolutiv)

Antrenare evolutivă

Exemplu: identificarea persoanelor cu risc de boli cardiovasculare

Set total de atribute:

(vârsta, greutate, înălțime, indice masă corporală, tensiune arterială, colesterol, glicemie)

Element al populației: (1,0,0,1,1,1,0)

Subset corespunzator de atribute:

(vârsta, indice masa corporală, tensiune arteriala, colesterol)

Evaluare: se antrenează rețeaua folosind subsetul de atribute selectat și se calculează scorul proporțional cu acuratețea clasificării

Observație:

- tehnica se poate utiliza și la antrenarea altor clasificatoare (ex: Nearest-Neighbor)
- Este cunoscută sub denumirea de “wrapper based attribute selection”

Antrenare evolutivă

Selecția exemplelor din setul de antrenare

- Motivație: dacă setul de antrenare este mare procesul de învățare devine mai costisitor și se poate ajunge la supraantrenare
- Este un proces de selecție similar celui anterior dar de această dată se efectuează la nivelul elementelor din setul de antrenare
- Codificarea adecvată este cea binară (similar cu cea de la selecția atributelor)
- Evaluarea elementelor constă în antrenarea rețelei (folosind o metodă adecvată de antrenare) pentru subsetul de exemple corespunzător elementului

Evoluția arhitecturii

Metode neevolutive de adaptare a arhitecturii:

- Constructive (“growing networks”)
  - Se pornește de la o rețea de dimensiune mică
  - Daca procesul de antrenare stagnează se adaugă o nouă unitate
  - Asimilarea noii unități se realizează prin antrenarea în primă fază doar a ponderilor asociate ei (celelalte rămânând fixate)
- Distructive (“pruning networks”)
  - Se pornește de la o rețea de dimensiune mare
  - Se elimină conexiuni/unități care nu influențează procesul de antrenare

Evoluția arhitecturii

Evoluția arhitecturii se poate realiza la unul sau mai multe dintre nivelele :

- Stabilirea numărului de unități
- Stabilirea modului de interconectare
- Alegerea tipului de funcție de transfer

Modalități de codificare:

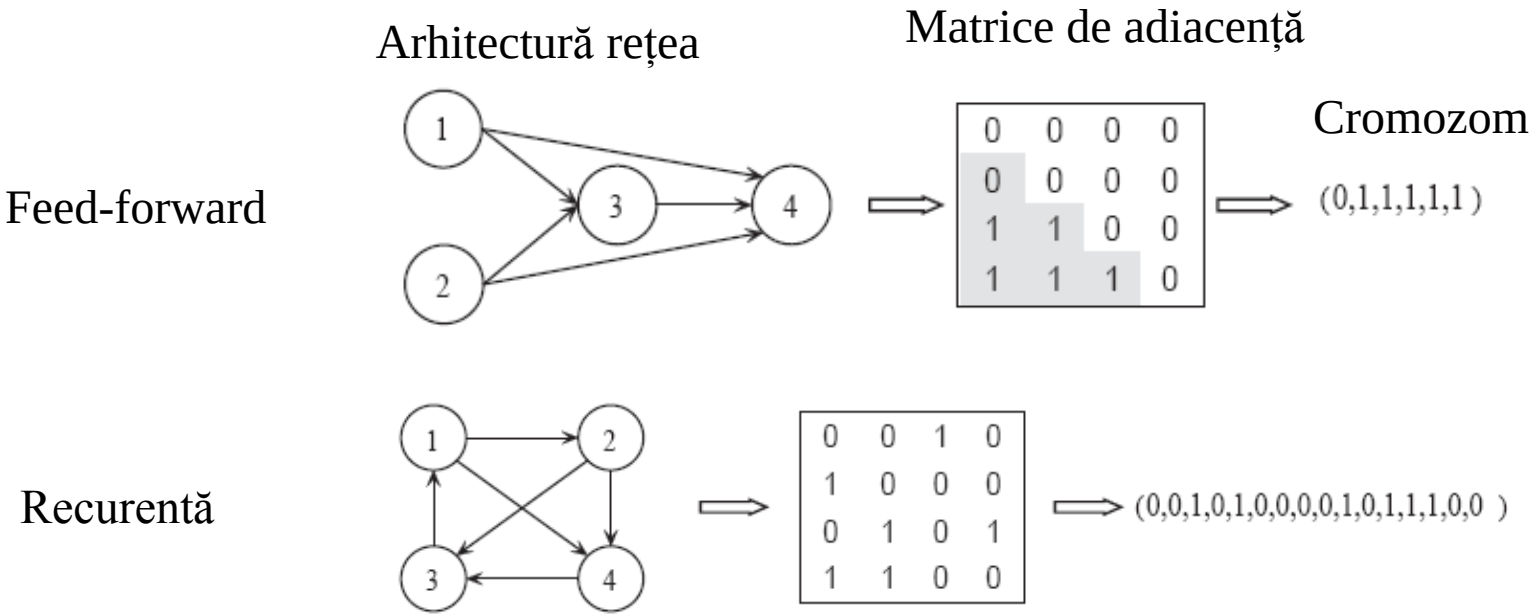
- Codificare directă
- Codificare indirectă

Evoluția arhitecturii

Codificare directă: fiecare element al rețelei se regăsește direct în codificare

- Arhitectura rețelei = graf orientat
- Rețeaua se codifica prin matricea de adiacență a grafului

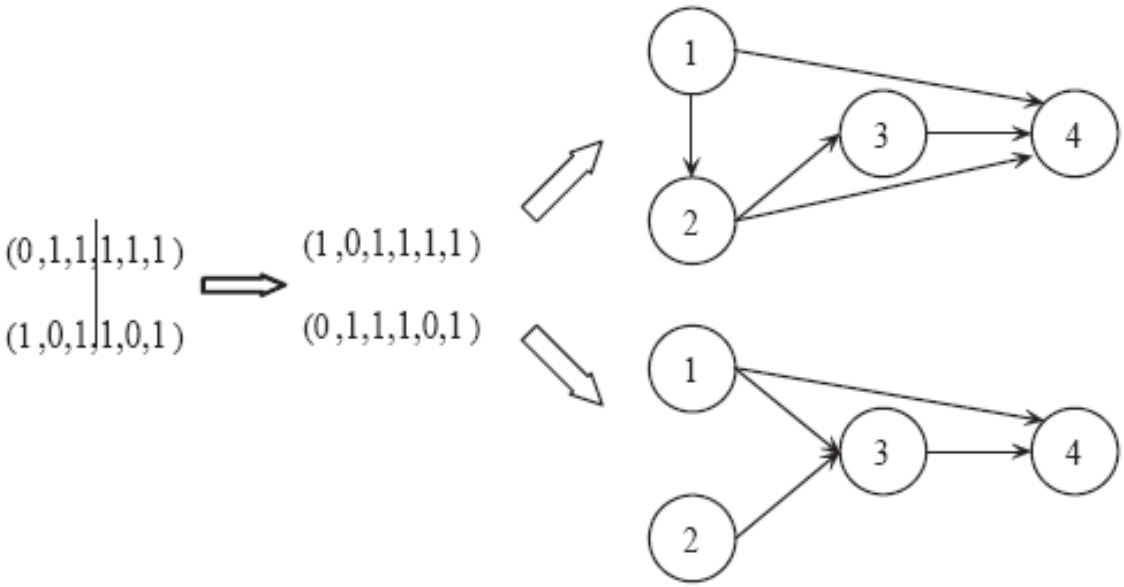
Obs. In cazul rețelelor feedforward unitățile pot fi numerotate astfel încât unitatea i primește semnale doar de la unități j cu j<i => matrice inferior triunghiulară



Evoluția arhitecturii

Operatori de variație pentru modificarea conexiunilor (număr de unități fixat):

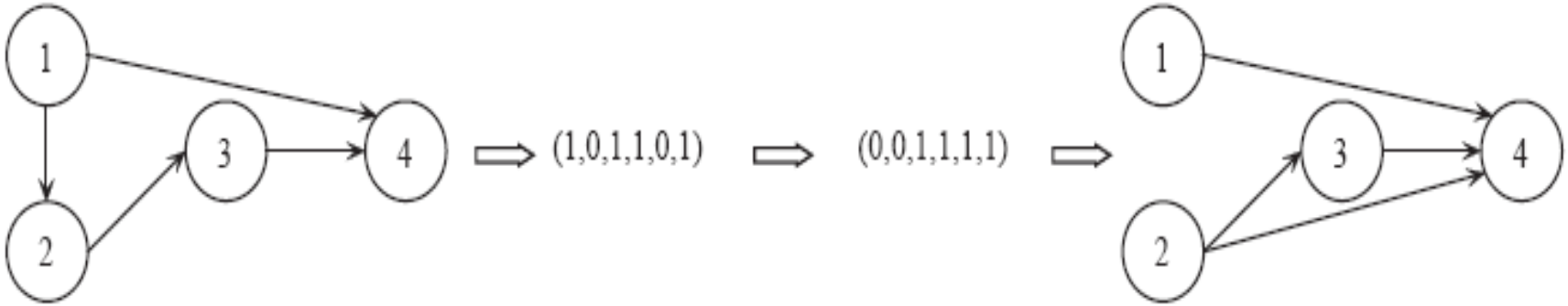
- Incrucișare specifică algoritmilor genetici



Evoluția arhitecturii

Operatori de variație pentru modificarea conexiunilor (număr de unități fixat):

- Mutație specifică algoritmilor genetici



Evoluția arhitecturii

Evoluția numărului de unități și a conexiunilor

Ipoteza: N – număr maxim de unități

Codificare:

- Vector binar cu N elemente
  - 0: unitate inactivă
  - 1: unitate activă
- Matrice de adiacență NxN asociată conexiunilor dintre unități
  - Pentru un element nul din vectorul cu unități linia și coloana corespunzătoare sunt ignorate

Evolutia arhitecturii

Evoluția tipului de functie de activare:

Codificare:

- Vector cu N elemente
  - 0: unitate inactivă
  - 1: unitate activă cu functie de activare de tip 1 (ex: tanh)
  - 2: unitate activă cu functie de activare de tip 2 (ex: logistică)
  - 3: unitate activă cu functie de activare de tip 3 (ex: liniară)

Evoluția ponderilor

- Matricea de adiacență se înlocuiește cu matricea de ponderi
  - 0: conexiune inexistentă
  - <>0: valoarea ponderii

Evoluția arhitecturii

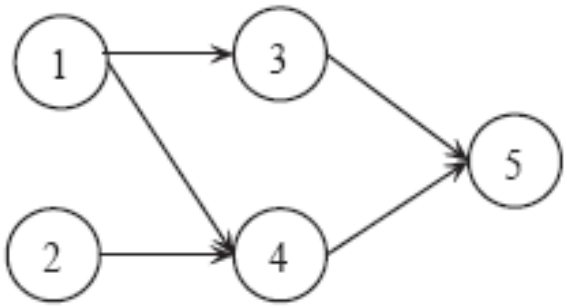
Evaluarea elementelor:

- Se antreneaza rețeaua (folosind un algoritm evolutiv sau unul clasic)
- Se estimează eroarea pe setul de antrenare (E_a)
- Se estimează eroarea pe setul de validare (E_v)
- Valoarea scorului (funcției de adecvare) este invers proporțională cu:
  - Eroarea pe setul de antrenare
  - Eroarea pe setul de validare
  - Dimensiunea rețelei (numărul de parametri ce intervin în rețea) – în felul acesta sunt favorizate rețelele de dimensiune mică

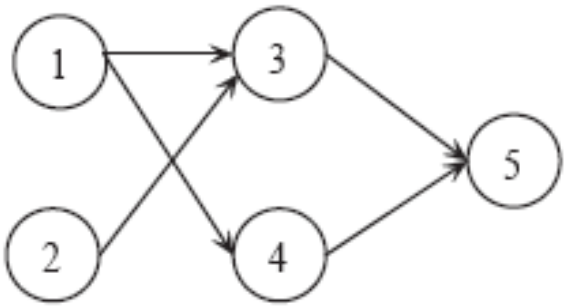
Evoluția arhitecturii

Dezavantaje ale codificării directe:

- Nu este scalabilă (pentru rețele de dimensiuni mari conduce la elemente de dimensiuni mari)
- Poate conduce la codificări diferite care corespund aceleiași arhitecturi (problema permutării) – redundanță fenotipică care limitează puterea de explorare a algoritmului (vezi exemplul de mai jos)
- Nu este adecvată pentru descrierea rețelelor neuronale modulare (în care porțiuni din rețea se repetă)



(0,1,0,1,1,0,0,0,1,1)



(0,1,1,1,0,0,0,0,1,1)

Evoluția arhitecturii

Codificare indirectă:

- mai plauzibilă dpdv biologic – ADN-ul nu codifică explicit fiecare celula din organism ci doar modul în care se sintetizează proteinele

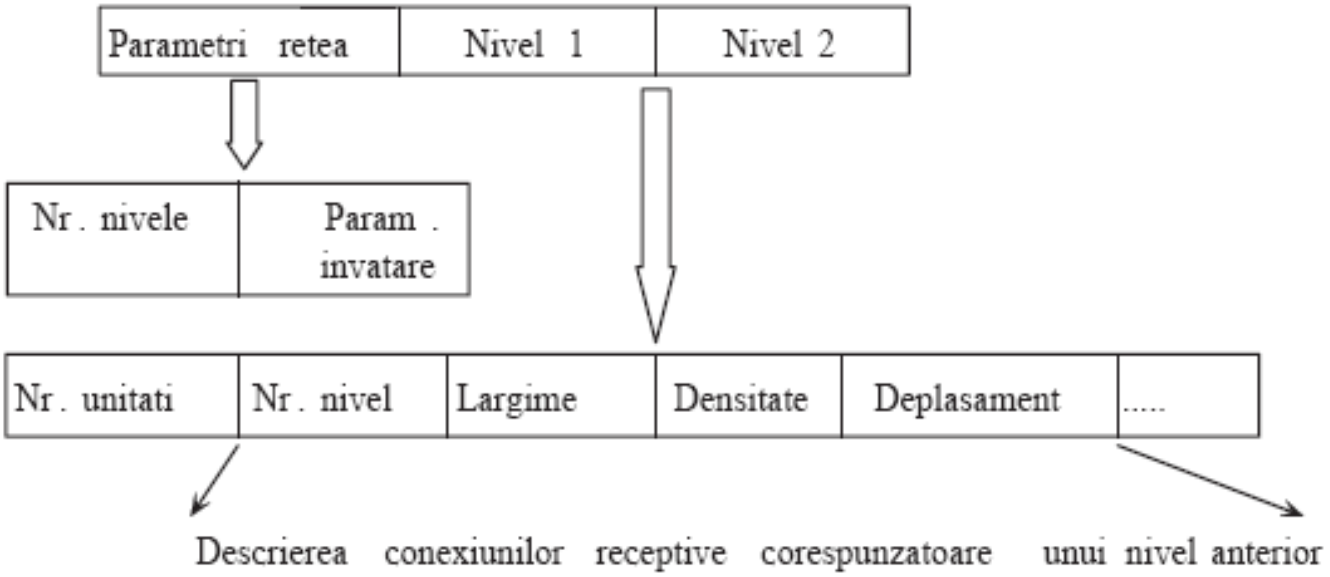
- Codificare parametrică
  - Rețeaua e descrisă de un set de caracteristici asociate arhitecturii (un fel de amprentă)
  - Caz particular (arhitectură parțial stabilită): rețele cu un singur nivel de unități ascunse și conectivitate standard. In acest caz singura caracteristică variabilă este numărul de unități de pe nivelul ascuns
  - La evaluarea unui element al populației trebuie instanțiată o rețea și antrenată (folosind un algoritm clasic sau unul evolutiv)

- Codificare prin reguli de dezvoltare

Evolutia arhitecturii

Codificare indirectă:

- Codificare parametrică



Instanțierea unei rețele: stabilirea aleatoare a conexiunilor în conformitate cu parametrii specificați în descriere

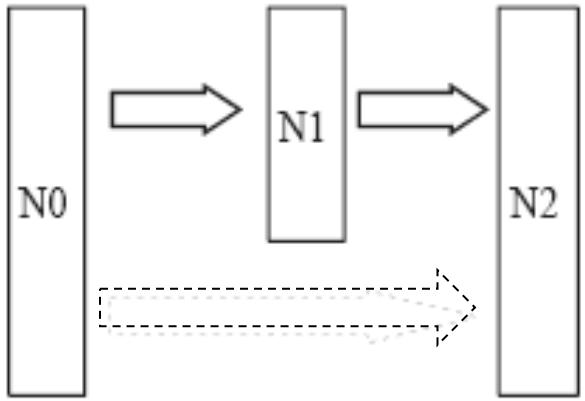
Evoluția arhitecturii

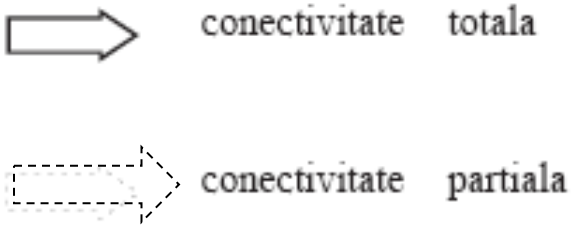
Exemplu:

Operatori:

Mutație: modificarea caracteristicilor rețelei

Recombinare: combinarea caracteristicilor nivelelor





Param. BP

Info. nivel 2

2 0.1 0.9 N1 0 100% 100% 0 N2 0 15% 80% 50% 1 100% 100% 0

Info. nivel 1

Numar nivele

Evoluția arhitecturii

Reguli de dezvoltare a arhitecturii (similar cu Grammar Evolution) :

Structura generala a unei reguli:

$$sn \rightarrow \begin{pmatrix} s_{11} & s_{12} \\ s_{21} & s_{22} \end{pmatrix}$$

Exemple de reguli:

$$S \rightarrow \begin{pmatrix} A & B \\ C & D \end{pmatrix}, \quad A \rightarrow \begin{pmatrix} a & a \\ a & a \end{pmatrix}, \quad B \rightarrow \begin{pmatrix} b & b \\ b & a \end{pmatrix}, \quad C \rightarrow \begin{pmatrix} b & a \\ a & c \end{pmatrix}, \quad D \rightarrow \begin{pmatrix} a & d \\ a & d \end{pmatrix},$$

$$a \rightarrow \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \quad b \rightarrow \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \quad c \rightarrow \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, \quad d \rightarrow \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}.$$

Structura unui element al populației:

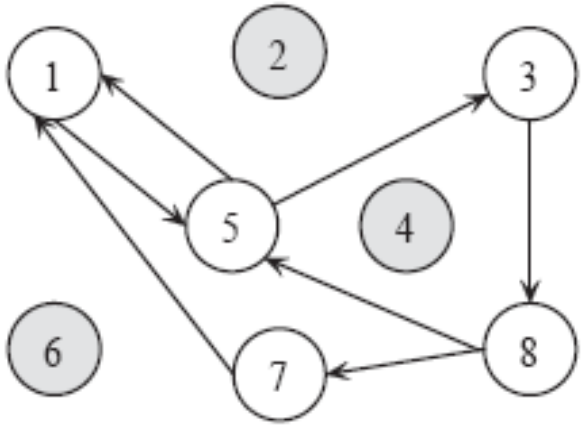
$$(A, B, C, D, a, a, a, a, b, b, b, a, b, a, a, c, a, d, a, d, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0)$$

Evoluția arhitecturii

Exemplu de derivare a unei arhitecturi:

$$S \rightarrow \begin{pmatrix} A & B \\ C & D \end{pmatrix} \rightarrow \begin{pmatrix} a & a & b & b \\ a & a & b & a \\ b & a & a & d \\ a & c & a & d \end{pmatrix} \rightarrow$$

$$\rightarrow \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$



Evoluția arhitecturii

Dezavantaj al evoluției separate a arhitecturii:

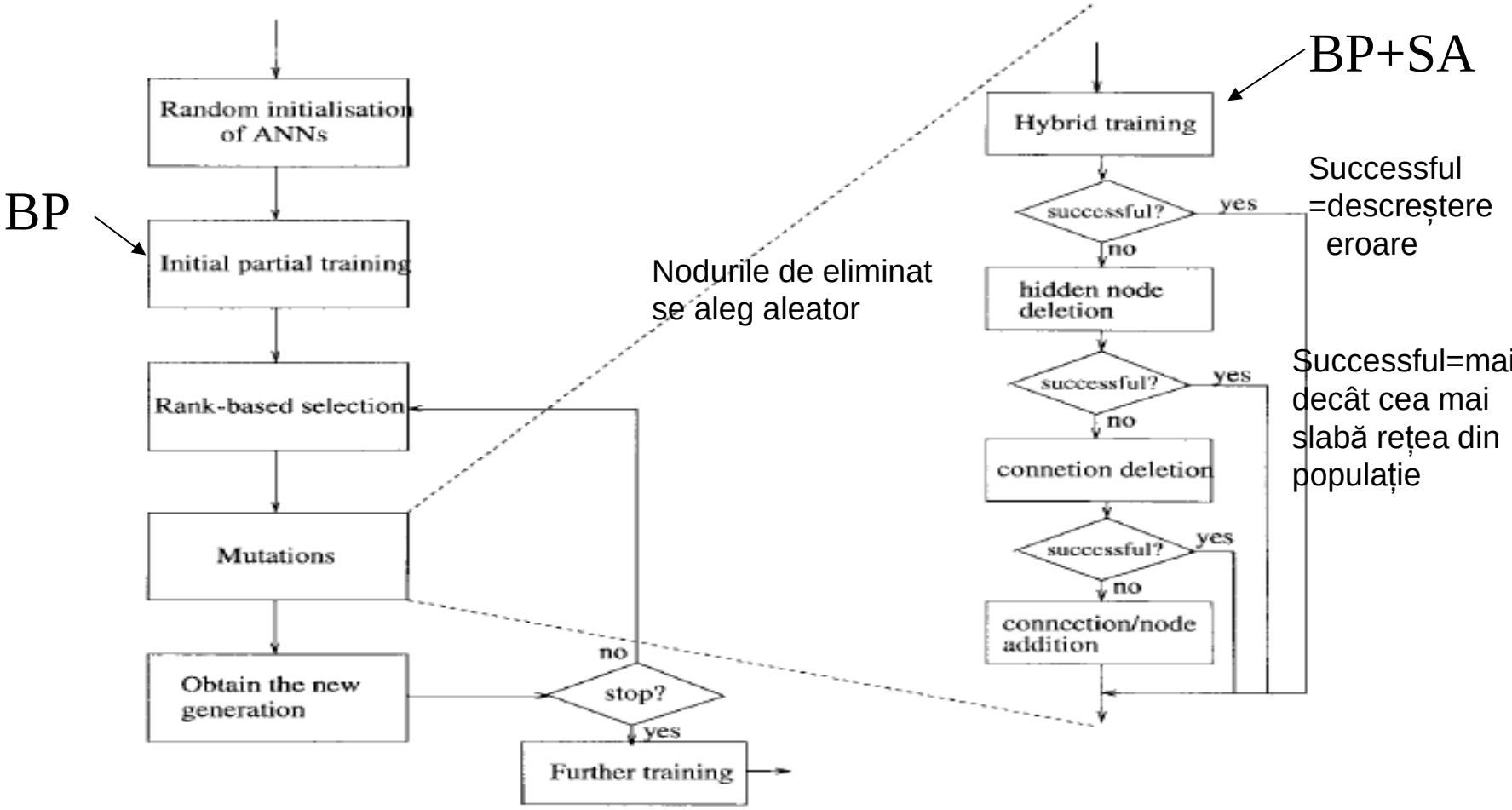
- Ca urmare a antrenării pornind de la valori inițiale aleatoare se obțin estimări afectate de zgomot al fitness-ului corespunzător unei arhitecturi

Soluții:

- Antrenarea de mai multe ori a aceleiași arhitecturi și calculul fitness-ului mediu => costuri mari
- Evoluția simultană a arhitecturii și ponderilor (asigură o corespondență 1 la 1 a genotipului (codificarea arhitecturii) și a fenotipului (rețeaua antrenată))

EPNet

Exemplu: EPNet = proiectare evolutivă rețele neuronale feedforward bazată pe principiile programării evolutive [Xin Yao, 1996]

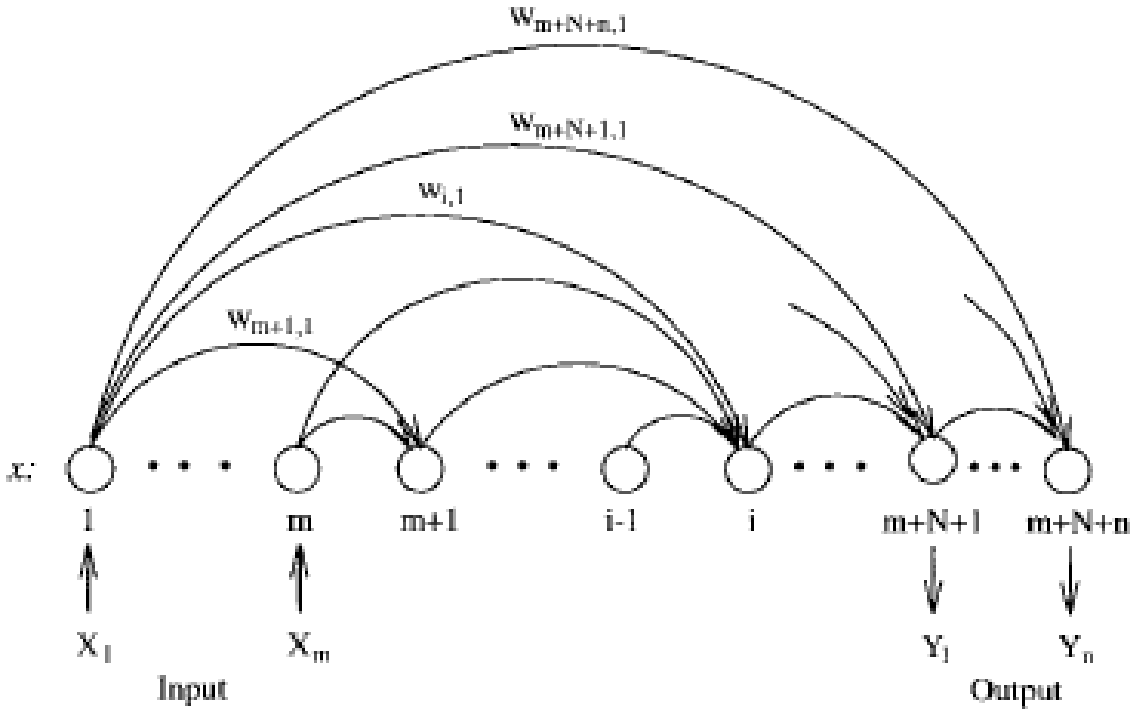


EPNet

Codificarea rețelei:

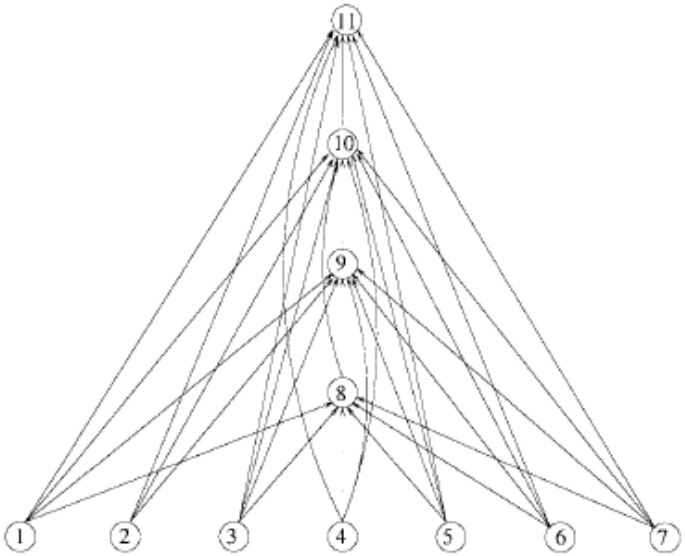
lista de noduri ascunse (tabel binar cu lg = nr. maxim de noduri ascunse)+
matrice de conectivitate +
matrice de ponderi

Interpretare: fiecare nod (neuron), cu exceptia primilor m (neuronii de intrare) este conectat cu toate nodurile anterioare

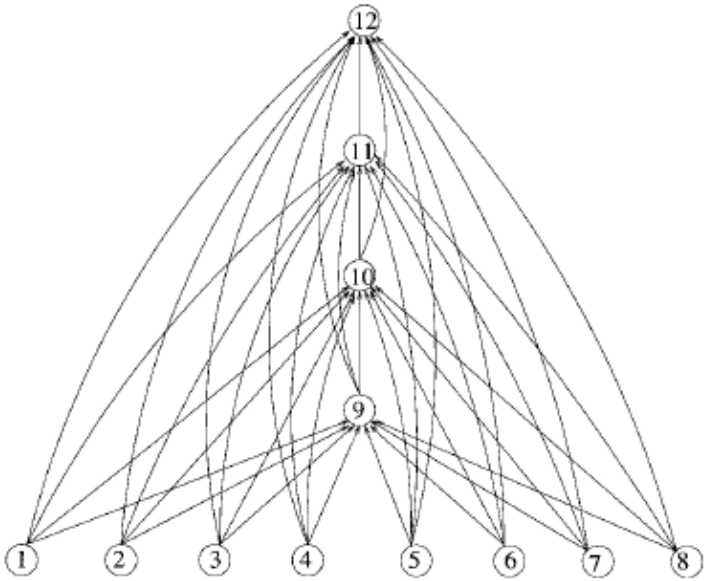


EPNet

Exemple de arhitecturi generate folosind EPNet pentru problema parității (generalizare XOR)



n=7



n=8

NEAT

NEAT = NeuroEvolution of Augmenting Topologies
(http://nn.cs.utexas.edu/?neat)

- Codificare directă:
  - Lista cu noduri (unități funcționale)
    - Tip nod: intrare, ascuns, ieșire, nod fictiv (pt specificarea pragului)
  - Lista cu conexiuni
    - Nod intrare (identificator nod)
    - Nod ieșire
    - Valoare pondere asociată conexiunii
    - Bit activare (0 – conexiune inactivă, 1- conexiune activă)
    - Indicator de inovare

NEAT

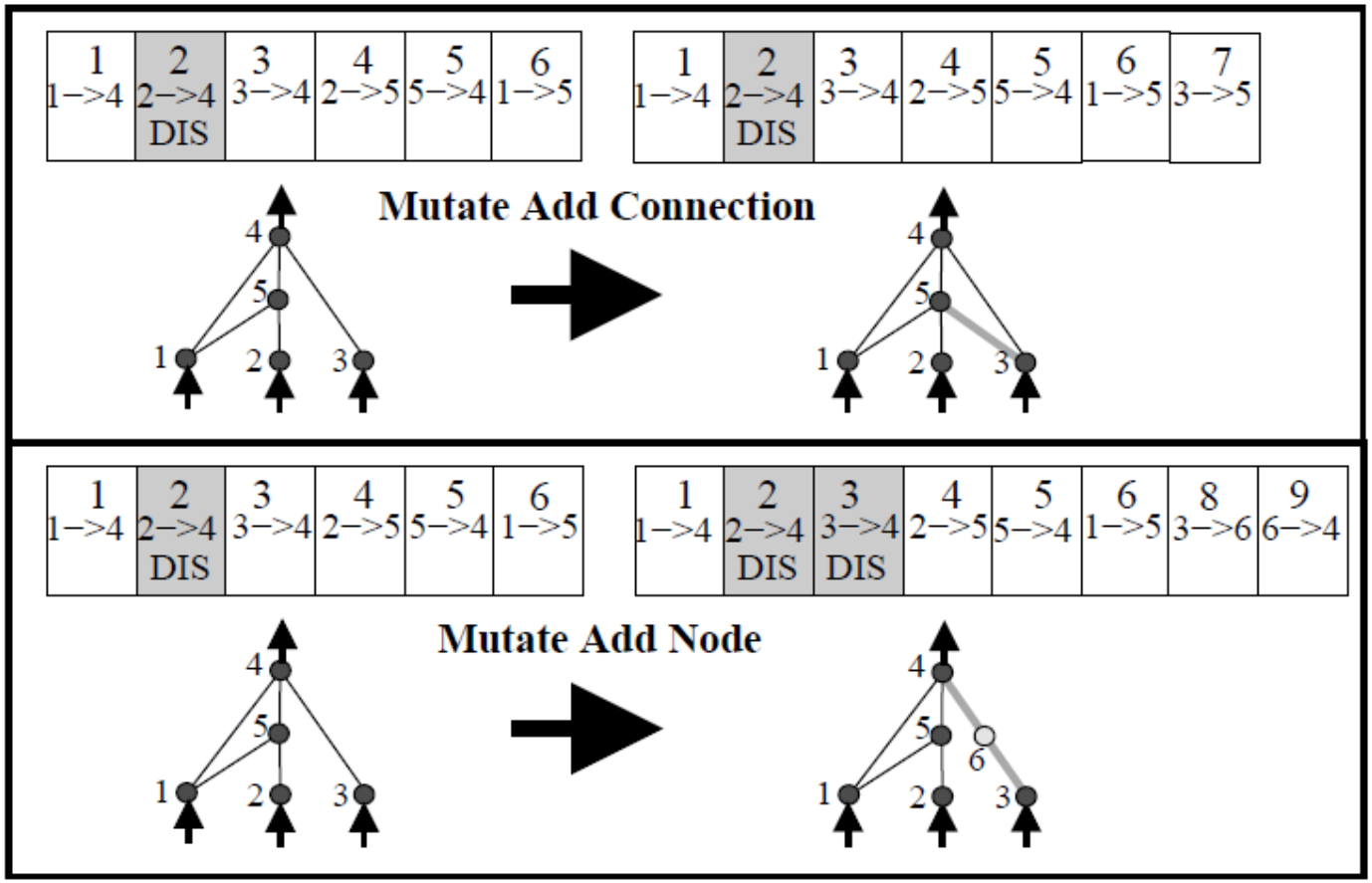
NEAT = NeuroEvolution of Augmenting Topologies
(http://nn.cs.utexas.edu/?neat)

- Populatia inițială este constituită din arhitecturi simple (doar nivel de intrare și nivel de ieșire)

- Variante de mutație:
  - Adăugare nod prin inserare între două noduri conectate: (prin divizarea unei conexiuni existente; conexiunea veche este eliminată iar două noi conexiuni sunt adăugate: cea care intră în noul nod este setată pe 1 iar cea care iese are valoarea care era atașată conexiunii care a fost eliminată)
  - Adăugare conexiune între noduri neconectate anterior (ponderea noii conexiuni se setează pe o valoare aleatoare)

NEAT

Exemplu mutație: (K.Stanley, R. Miikulainen – Evolving Neural Networks through Augmenting Topologies, Evol.Comput. 2002)



NEAT

Incrucișare:

doi parinți ---- un urmaș

similară încrucișării uniforme de la algoritmii genetici

Etapa 1: identificarea genelor corespondente din cei doi părinți

- Fiecare genă (corespunzătoare unei conexiuni) are un indicator de inovare (alocat în momentul creării genei respective); valoarea indicatorului de inovare se stabilește pe baza unei variabile globale (asociate rețelei) și care este incrementată de fiecare dată când este adăugată o genă
- Genele din părinți care au același indicator de inovare sunt considerate gene corespondente
- Dacă o genă dintr-un părinte nu are corespondent (genă cu același indicator de inovare) în celălalt părinte atunci este denumită genă disjunctă sau genă în exces

NEAT

Incrucișare:

doi părinți ---- un urmaș

similară încrucișării uniforme de la algoritmii genetici

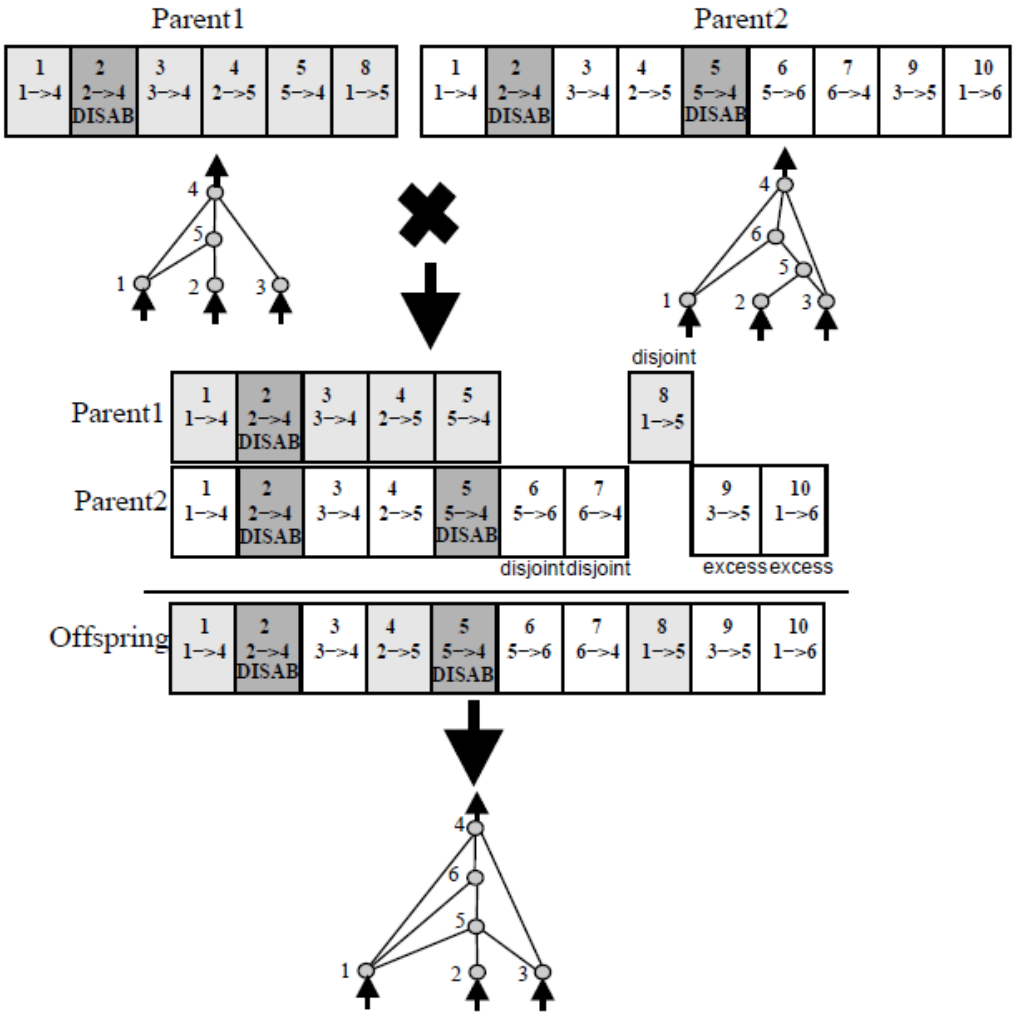
Etapa 2: construirea urmașului

- Pentru genele corespondente se alege aleator părintele de la care se selectează gena (indicatorul de inovare rămâne neschimbat)
- Pentru genele disjuncte sau în exces se decide dacă se includ în urmaș fie aleator fie pe baza calității părinților (dacă părintele care conține gena e mai bun decât cel care nu o conține atunci gena se include în urmaș, altfel nu se include)

NEAT

Exemplu încrucișare

(Stanley, Miikulainen, 2002)



NEAT

Observații:

- La un moment dat populația va conține rețele cu arhitecturi diferite
- Pentru a permite noilor arhitecturi să supraviețuiască în populație se utilizează o tehnică de evoluție la nivel de specii (speciere) prin care, în procesul de selecție sunt comparate între ele doar rețele având arhitecturi similare)
- Măsura de similaritate între două arhitecturi: combinație liniară în care intervin:
  - diferența medie dintre ponderile asociate genelor corespondente (W)
  - numărul de gene disjuncte (D)
  - numărul de gene în exces (E)

  S=D/N+E/N +W (N=max{N1,N2}, N1,N2=nr gene din cele două rețele)
- Selecția se bazează utilizarea unei funcții de partajare (curs viitor)

Evoluția regulii de învățare

Un algoritm de antrenare descrie un proces iterativ în care la fiecare iterație se ajustează valorile ponderilor

Forma generală a ajustării locale a unei ponderi este

$$w_{ij}(k+1) = \varphi(w_{ij}(k), x_i, y_i, x_j, y_j, \delta_i, \delta_j, \alpha)$$

x_i,x_j – semnale de intrare în unitățile i respectiv j

y_i,y_j – semnale de ieșire din unitățile i respectiv j

α – parametrii de control ai antrenării (ex: rata de invatare)

δ_i,δ_j – semnal de eroare corespunzător unitatii i, respectiv j

Exemplu: ajustare de tip BackPropagation

$$w_{ij}(k+1) = w_{ij}(k) + \eta \delta_i y_j$$

Evolutia regulii de învățare

Elemente care pot fi supuse procesului de evoluție:

- Parametrii algoritmului de antrenare (ex: rata de învățare, coeficientul termenului moment)
- Expresia corespunzătoare regulii de transformare (similar programării genetice)

Evaluarea fiecărui element al populației:

- Antrenarea rețelei pentru fiecare regulă

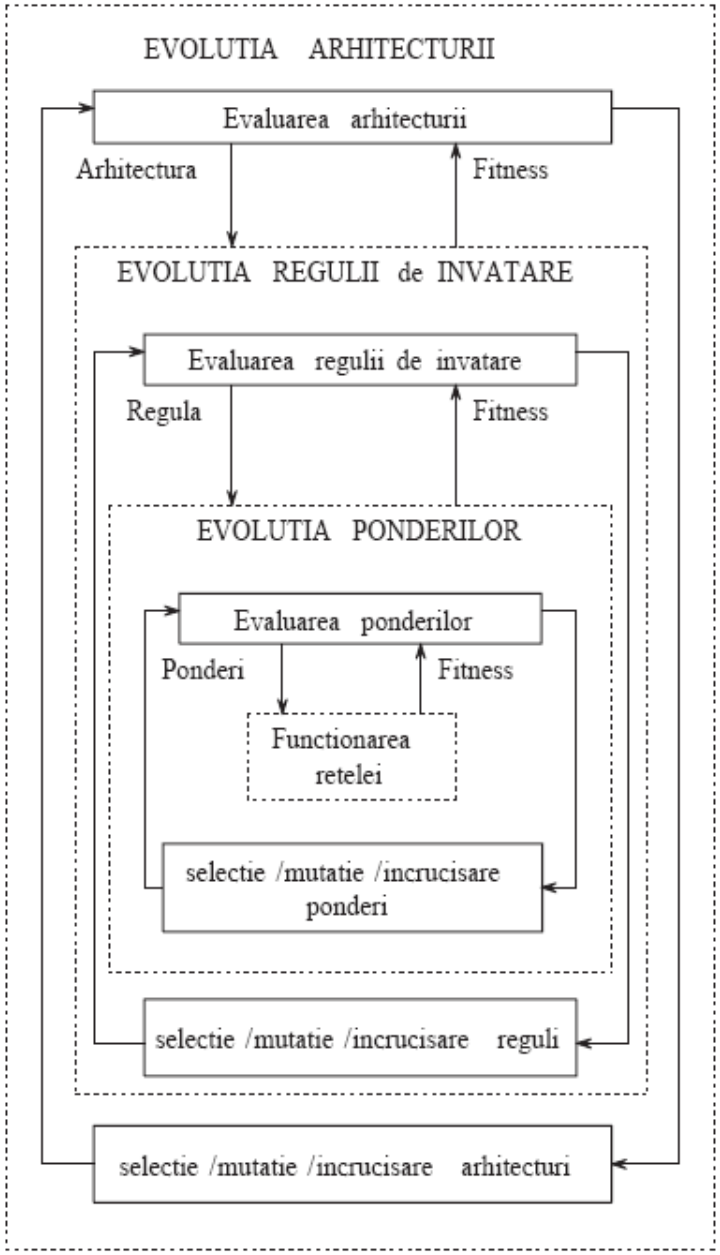
Dezavantaj: proces extrem de costisitor

Sumar

Structura generală a unui proces de proiectare evolutivă a unei rețele neuronale

Nivele:

- Ponderi
- Reguli de învățare
- Arhitectura



X. Yao, Evolving ANN, 1999