# CAPITOLUL VII

# ÎNVĂŢARE

## 7.1 Problematica învăţării în Inteligenţa Artificială

Problemele de învăţare reprezintă o direcţie importantă de cercetare în Inteligenţa Artificială, cea a *maşinilor auto-instruibile* (*învăţării automate*). Învăţarea automată reprezintă studiul modelelor de sisteme, care pe baza unui set de date (de instruire), îşi îmbunătăţesc performanţele prin experienţe şi prin învăţarea unor cunoştinţe specifice domeniului, prin experienţe.

Încercarea de modelare a raţionamentului uman conduce la apariţia noţiunii de *raţionament inteligent*. *Raţionamentul* este un proces de a deduce concluzii; raţionamentul *inteligent* vrem să simuleze raţionamentul efectuat de oameni.

Majoritatea sistemelor cu Inteligenţă Artificială sunt sisteme deductive, capabile să tragă concluzii(să facă inferenţe) pe baza cunoaşterii lor iniţiale sau furnizate, nefiind însă capabile să achiziţioneze şi să genereze singure cunoştinţe.

Capacitatea de a învăţa fiind strâns legată de comportamentul inteligent, o direcţie de cercetare în Inteligenţa Artficială este implementarea în maşini a capacităţii de învăţare. Realizarea unor astfel de maşini auto-instruibile devine deosebit de importantă în realizarea sistemelor complexe bazate pe cunoaştere, cum ar fi de exemplu sistemele expert.

După cum am arătat până acum, *inteligenţa* unui agent este încorporată în partea sa de program. La un moment dat, agentul va alege cel mai bun mod de acţiune, în sensul în care a fost programat să o facă.

În situaţii în care programul are informaţii (cunoştinţe) incomplete despre mediul în care agentul acţionează, *învăţarea* este singura modalitate prin care agentul poate să achiziţioneze cunoştinţele de care are nevoie. Astfel, *învăţarea* conferă *autonomia* agentului, în sensul în care a fost descrisă în capitolul I. Deasemenea, conferă o modalitate de a construi sisteme performante – furnizând o experienţă dobândită prin învăţare în domeniul aplicaţiei.

Ştiinţa maşinilor cu învăţare automată (maşini auto-instruibile) se ocupă cu dezvoltarea de algoritmi care:

- capătă experienţă când operează într-un anumit mediu;
- recunosc propria stare internă (poate chiar şi datele şi programul) pentru a deveni mai eficienţi în mediul pentru care au fost pregătiţi;
- au abilitatea de a-şi evalua propriile performanţe fără a fi necesară intervenţia unor stimuli externi.

Studiul sistemelor inteligente cuprinde atât maşinile cu învăţare automată, cât şi tehnologiile de genul sistemelor expert şi reţelelor neuronale, care necesită uriaşe eforturi umane pentru dezvoltarea seturilor de reguli euristice, respectiv a datelor necesare pentru antrenare.

### 7.1.1 Modelul general al unui agent care învaţă

Un agent care învaţă poate fi împărţit în patru componente conceptuale. Distincţia cea mai importantă care trebuie făcută este între *elementul de învăţare,* care este responsabil de a face îmbunătăţiri, şi *elementul de performanţă*, care este responsabil de a selecta acţiuni externe. Elementul de performanţă este ceea ce până acum am considerat a fi întregul agent: primeşte percepţiile şi alege acţiunile. Elementul de învăţare vede cum se "descurcă" agentul şi stabileşte cum ar trebui modificat elementul de performanţă pentru a acţiona mai bine în viitor.

Modelul general al unui agent care învaţă este descris în Figura 7.1.

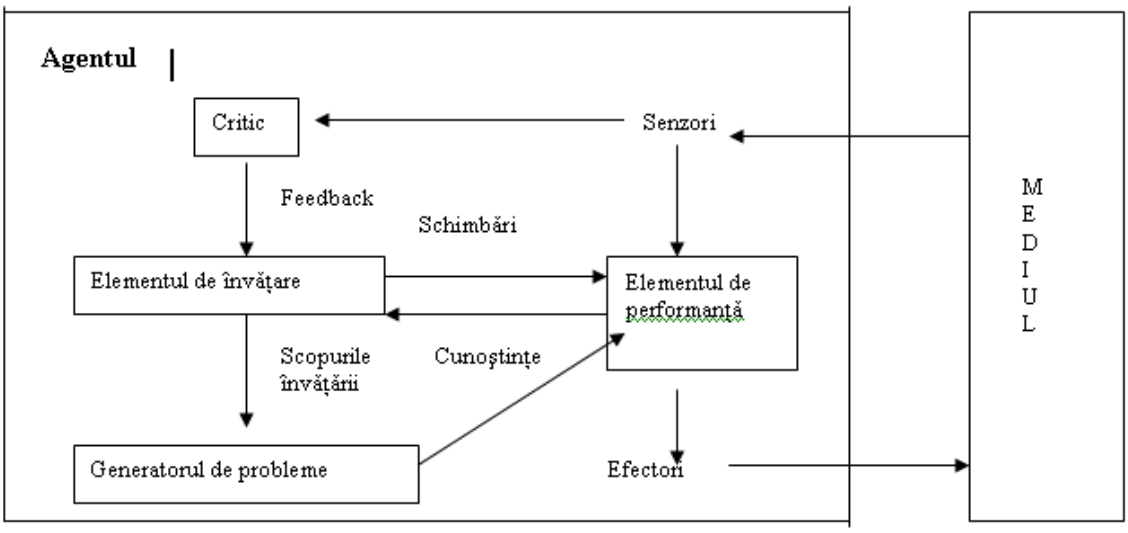


**Figura 7.1. Modelul general al unui agent care învaţă**

Într-un astfel de model, *criticul* îi spune elementului de învăţare cât de bine se descurcă agentul. Criticul se foloseşte de un standard fix de performanţă. Acest lucru este necesar deoarece percepţiile nu dau nici o indicaţie despre succesul agentului. De exemplu, un program de şah poate primi o percepţie care indică faptul că i-a dat mat adversarului său, dar este nevoie de un standard al performanţei pentru a şti dacă acesta este un lucru bun deoarece percepţia nu i-o spune. Este important ca standardul de performanţă să fie o măsură fixă aflată în exteriorul agentului, altfel agentul ar putea ajusta standardul de performanţă pentru ca acesta să arate că este bun comportamentul său.

*Generatorul de probleme* este responsabil de a sugera acţiuni care ar putea conduce la noi informaţii. Daca elementul de performanţă ar face totul, fără a fi întrebat generatorul de probleme, acesta ar alege acţiunile cele mai bune având în vedere ceea ce cunoaşte agentul la un moment dat. Dar dacă agentul ar explora puţin şi ar alege acţiuni suboptimale pe termen scurt, ar putea descoperi acţiuni mult mai bune pe termen lung. Sarcina generatorului de probleme este de a sugera aceste acţiuni de explorare.

Să considerăm exemplul unui agent care este şofer de taxi. Elementul de performanţă este alcătuit din colecţia de cunoştinţe şi proceduri pe care le are agentul pentru a putea selecta acţiunile necesare conducerii maşinii (întoarcerea, accelerarea, frânarea, etc.). Agentul conduce maşina folosind acest element de performanţă. Elementul de învăţare formulează ţelurile, de exemplu, să înveţe cum se comportă maşina pe un drum umed. Criticul observă lumea şi trimite informaţiile la elementul de învăţare. De exemplu, dacă şoferul trece rapid de pe o bandă pe alta fără a semnaliza, criticul observă iritarea celorlalţi participanţi la trafic şi îi transmite aceste observaţii elementului de învăţare. Acesta formulează o regulă care spune că este greşită schimbarea benzii fără semnalizare şi elementul de performanţă se va modifica prin adăugarea acestei noi reguli. Din când în când generatorul de probleme dă agentului o sugestie. De exemplu: dacă este aglomerată strada, generatorul de probleme i-ar putea sugera agentului să încerce să meargă pe o stradă paralelă şi să vadă dacă nu cumva aceasta este mai rapidă decât calea aleasă iniţial.

*Elementul de învăţare* mai este responsabil şi de îmbunătăţirea eficienţei elementului de performanţă. De exemplu, dacă şoferului de taxi i se va cere să meargă pe o rută nouă, acesta îşi va consulta harta pentru a găsi drumul cel mai scurt. Când i se va solicita din nou acest traseu, şoferul nu va mai căuta pe hartă căci deja va cunoaşte acest traseu.

**Componentele elementului de performanţă**

Elementul de performanţă al unui agent care învaţă are următoarele componente:

1. o mapare directă între condiţii din stări şi acţiuni;
2. o metodă de a deduce proprietăţi relevante ale mediului din secvenţa perceptuală;
3. informaţii despre modul în care lumea (mediul) evoluează;
4. informaţii despre rezultatele posibilelor acţiuni pe care agentul le poate efectua;
5. informaţii de utilitate indicând utilitatea stărilor mediului;
6. informaţii acţiuni-valoare indicând utilitatea unor acţiuni particulare în stări particulare;
7. scopuri indicând clase de stări a căror realizare va maximiza utilitatea agentului.

Fiecare dintre cele şapte componente ale elementului de performanţă pot fi descrise matematic sub forma unor **funcţii**. Spre exemplu componenta 3 poate fi interpretată ca o funcţie care asignează unei stări din mediu (starea curentă) o altă stare a mediului (starea următoare); componenta 7 poate fi intrepretată ca o funcţie care asignează unei stări a mediului o valoare booleană (adevărat sau fals) după cum starea este sau nu o stare finală (scop).

Ca urmare, orice sarcină de învăţare poate fi văzută de fapt ca fiind învăţarea reprezentării unei anumite funcţii.

**Feed-back-ul disponibil**

Pentru anumite componente ale elementului de performanţă, cum ar fi componenta pentru precizerea rezultatelor unei acţiuni (compenta 4), feed-back-ul disponibil îi spune, în general, agentului, care este rezultatul corect (ieşirea corectă).

Orice situaţie în care pot fi observate atât semnalele de intrare cât şi semnalele de ieşire ale unei componente se numeşte ***învăţare supervizată*** (***Supervised Learning***) (în general, ieşirile sunt furnizate de către un profesor). Aceasta constă în faptul că agentul crede că o anumită acţiune are un anumit rezultat iar mediul emite un stimul prin care îi spune agentului care este rezultatul corect al acţiunii sale.

Pe de altă parte, în învăţarea componentei *condiţie-acţiune* (componenta 1), agentul primeşte o evaluare a acţiunii sale, dar nu i se spune care este ieşirea (rezultatul corect). Acest tip de învăţare se numeşte ***învăţare prin întărire*** (***Reinforcement Learning)***, iar evaluarea primită de agent se numeşte ***recompensă*** (***reinforcement***).

Învăţarea în care nu se dă agentului nici o indicaţie referitoare la rezultatele corecte se numeşte ***învăţare nesupervizată*** (***Unsupervised Learning***).

### 7.1.2 Învăţarea unui domeniu

În învăţarea unui domeniu specific, sistemul conţine numeroase concepte predefinite, structuri de reprezentare a cunoştinţelor, restricţiile domeniului şi reguli euristice. Se aşteaptă ca sistemul să deducă noi atribute şi concepte în procesul de învăţare (inducţie constructivă). Un sistem bazat pe această abordare este dezvoltat pentru un domeniu particular şi nu poate fi folosit direct în alt domeniu. Tipurile de învăţare folosite sunt: *învăţarea din exemple, învăţarea prin analogie* şi *învăţarea prin observaţie şi descoperire*.

S-au construit sisteme ce combină această abordare şi calculul neuronal. Separând mecanismele de inferenţă generale de cunoştinţele specifice domeniului s-au realizat sisteme ce pot fi aplicate la o largă varietate de domenii diferite.

### 7.1.3 Tipuri de învăţare

În procesul de învăţare, informaţia furnizată de un profesor sau de mediu este transformată într-o formă nouă în care ea este memorată pentru uzul ulterior. Natura acestei transformări determină tipul de strategie. Astfel putem distinge *învăţarea mecanică*, *învăţarea prin instruire*, *învăţarea prin analogie* şi *învăţarea prin deducţie*. Ultima se subdivide în *învăţarea din exemple* şi *învăţarea din observaţie* (*învăţarea nesupervizată*) şi prin *descoperire*.

În învăţarea mecanică informaţia furnizată de profesor nu suferă nici o transformare esenţială, fiind memorată direct. Problema de bază este cum să fie indexată şi memorată cunoaşterea în vederea regăsirii ulterioare. În învăţarea prin instruire, transformările de bază sunt *selecţia* şi *reformularea* (mai ales la nivel sintactic) a informaţiei furnizate de profesori. În învăţarea prin deducţie, pe baza cunoaşterii date, sunt formulate inferenţe deductive care au proprietatea de a conserva adevărul, concluziile utile fiind memorate. Dacă procesele de transformare presupun *generalizarea* informaţiei primite şi *selectarea* celui mai plauzibil sau dezirabil rezultat, avem de-a face

cu un proces de *inferenţă inductivă*. În învăţarea *prin analogie* se combină învăţarea deductivă şi cea inductivă. Descrierile din diferite puncte de vedere sunt combinate pentru a determina o substructură comună, care să servească drept bază pentru analogii. Găsirea substructurii comune presupune inferenţă inductivă, în timp ce realizarea proiecţiei analogice este o formă de deducţie.

## 7.2 Învăţarea în Inteligenţa Artificială Tradiţională

### 7.2.1 Învăţarea Supervizată

#### 7.2.1.1 Învăţarea inductivă

La învăţarea supervizată elementul de învăţare primeşte valorile corecte (sau aproape corecte) ale unei funcţii pentru anumite semnale de intrare particulare. Elementul de învăţare modifică reprezentarea funcţiei astfel încât să se potrivească cu informaţiile pe care le primeşte. De exemplu, se dă o pereche de forma ($x$, $f(x)$) unde $x$ este semnalul de intrare, iar $f(x)$ este valoarea funcţiei $f$ aplicată lui $x$. Sarcina inducţiei este următoarea: fiind dată o colecţie de valori ale funcţiei $f$, să se găsească funcţia $h$ care aproximează funcţia $f$. Funcţia $h$ se numeşte *ipoteză*. Funcţia $f$ nu se cunoaşte, dar sunt mai multe variante de alegere a funcţiei $h$.

Să presupunem că avem un agent reflex care învaţă de la un profesor. Prin funcţia *Reflex_Learning_Element*, descrisă în Figura 7.2.a), agentul, prin elementul de învăţare, îşi reactualizează variabila globală, *exemple,* care conţine o listă de perechi (*acţiune*, *stare*). Percepţia ar putea fi o poziţie de pe tabla de şah şi acţiunea ar putea fi cea mai bună mutare care îl poate conduce pe agent la câştigarea partidei. Când elementul de performanţă (descris în Figura 7.2.b) prin funcţia *Reflex_Performance_Element*) va primi o percepţie despre care a învăţat, va alege acţiunea corespunzătoare. În celelalte cazuri va apela algoritmul de învăţare *Inducţie*, care va formula o ipoteză *h* pe care agentul o va folosi pentru a alege acţiunea următoare.

După cum se vede din Figura 7.2, elementul de învăţare, memorează fiecare pereche (percepţie, acţiune). Elementul de performanţă repetă acţiunea care a mai făcut-o când a primit acelaşi stimul sau caută o acţiune. Setul *exemple* e o variabilă globală care apare atât în elementul de învăţare cât şi în cel de performanţă.

**funcţia *Reflex_Learning_Element*** (*percepţie, acţiune*)
**intrări**: *percepţie, acţiune*
*exemple* ← *exemple* ∪ {(*percepţie, acţiune*)}
**sf- *Reflex_Learning_Element*** **a**

**funcţia *Reflex_Performance_Element*** (*percepţie*) **returnează** *acţiune*
**dacă** (*percepţie, a*) **în** *exemple* **atunci returnează** *a*
**altfel** h ← **Inducţie**(*exemple*)
**sf-dacă**
**returnează h**(*percepţie*)
**sf- *Reflex_Performance_Element*** **b**

**Figura 7.2. Modelul unui agent reflex care învaţă de la un profesor**

### 7.2.1.2. Învăţarea prin reţele neuronale

Calculul neuronal implică două aspecte fundamentale: învăţarea şi reprezentarea cunoaşterii. Abordarea neuronală a acestor aspecte este diferită de abordarea standard întâlnită în modelele bazate pe reprezentarea simbolică a cunoştinţelor. Calculul neuronal este caracterizat prin posibilitatea învăţării bazată pe informaţii parţiale sau pe date ce conţin erori. Acest lucru este posibil datorită caracterului robust (în raport cu informaţiile ce reprezintă erori) al învăţării neuronale.

Există un consens aproape unanim în a accepta faptul că tehnicile de învăţare reprezintă calea prin care maşinile pot deveni capabile să realizeze sarcinile dificile ale inteligenţei artificiale. În mod evident un program inteligent, care învaţă prin instruire şi din experienţă, poate fi capabil să realizeze sarcini mai dificile decât un program bazat pe o listă de posibilităţi care a fost stabilită şi analizată de programatorul uman.

Există o analogie între neuronii umani şi cei artificiali. Această analogie este sintetizată mai jos:

| **Neuron uman** | **Neuron Artificial** |
|---|---|
| Neuron | Element de procesare |
| Dendrite | Conexiuni |
| Corp celular | Funcţie de activare |
| Axon | Element de ieşire |
| Sinapse | Ponderi |

Totuşi, analogia nu este perfectă. Neuronii umani şi activitatea neuronală sunt mult mai complecşi decât poate sugera studierea neuronilor artificiali. Neuronii umani nu efectuează o simplă sumă ponderată, iar mecanismul de funcţionare al dendritelor în sistemul biologic este mult mai elaborat.

Învăţarea în reţele neuronale poate fi văzută în două moduri. Din punct de vedere computaţional este vorba despre o metodă de reprezentare a funcţiilor folosind reţele de elemente computaţionale aritmetice simple şi despre metode de învăţare a acestor reprezentări din exemple. Din punct de vedere biologic, învăţarea în reţele neuronale şi

reţele “de încredere” poate fi văzută ca un model matematic pentru reprezentarea operaţiilor creierului. Elementele computaţionale aritmetice simple corespund *neuronilor* (celulele care execută procesul de informare în creier), iar reţeaua corespunde unei colecţii de neuroni interconectaţi. Din acest motiv reţelele se numesc *reţele neuronale*. Reţelele neuronale oferă una dintre acele şanse pentru înţelegerea multor fenomene psihologice, fenomene care apar din structura specifică şi din operaţiile creierului.

O *reţea neuronală* este compusă dintr-un număr de noduri sau *unităţi* conectate prin *legături*. Fiecare legătură are asociată o anumită *pondere*. Învăţarea are loc prin reactualizarea ponderilor. Unele elemente sunt conectate la mediul extern şi pot fi proiectate ca elemente de intrare sau ieşire. Ponderile sunt modificate în încercarea de a determina comportamentul reţelei să fie cât mai aproape de semnalele de intrare date de mediu.

**Funcţia de activare neuronală**

Neuronii unei reţele neuronale îşi transformă intrarea în ieşire folosind o funcţie numită funcţie de activare. Această ieşire constituie parte din intrarea tuturor neuronilor conectaţi la neuronul respectiv. Învăţarea unei reţele neuronale depinde de valorile ponderilor conexiunilor interneuronale dar şi de funcţia de activare (numită şi funcţie de transfer a informaţiei) specifică neuronilor.

În cazul neuronilor ascunşi, funcţia de activare are scopul de a introduce non-liniaritatea în reţeaua neuronală. Reţelele neuronale cu mai multe straturi dau rezultate foarte bune şi datorită non-liniarităţii.

Câteva din cele mai importante funcţii de activare neuronală sunt prezentate mai jos:

- **Funcţia identitate**

Această funcţie lasă intrarea nemodificată, ieşirea neuronului fiind astfel identică cu intrarea sa. Uneori însă, avem de-a face cu extinderi ale acestei funcţii, intrarea fiind multiplicată cu o constantă pentru a determina valoarea de ieşire.

$g(x)=x$

- **Funcţia prag**

Este cunoscută şi sub denumirea de funcţie binară de activare, deoarece poate lua doar două valori: 0 şi 1 şi este folosită de obicei în reţelele neuronale cu un singur strat.

$$g(x)=\begin{cases}1 & \text{if}(x \geq \theta\ ) \\ 0 & \text{if}(x < \theta\ )\end{cases}$$

- **Funcţia sigmoidală**

Este indicată folosirea acestui tip de funcţie de activare la reţelele care folosesc un algoritm de învăţare cu propagare înapoi a erorii şi în aplicaţiile în care ieşirile sunt numere reale între 0 şi 1.

$$g(x) = \frac{1}{1+e^{-x}}$$

- **Funcţia sigmoidală bipolară**

După cum sugerează şi numele, această funcţie este asemănătoare funcţiei sigmoidale, fiind o extensie a acesteia. Este aplicabilă problemelor pentru care ieşirea ia valori reale între -1 şi 1.

$$g(x) = \frac{1-e^{-x}}{1+e^{-x}}$$

Se observă că funcţiile de activare liniare, dau posibiliatatea neuronului să aibă mai mult de două ieşiri. Ieşirea neuronului este proporţională cu activarea sa. Pentru neuronii a căror funcţie de transfer este sigmoidală, valorile de ieşire variază continuu, dar nu liniar.

**Reţele neuronale cu transmitere înainte (feedforward)**

Reţelele neuronale cu transmitere înainte sunt cele mai populare şi cele mai folosite modele în aplicaţiile practice. Mai sunt cunoscute şi sub denumirea de perceptroni multi-strat. O reţea neuronală cu transmitere înainte este o reţea de elemente similare neuronilor, organizate în straturi, cu o funcţie de activare diferenţiabilă, de obicei funcţia sigmoidală. Algoritmul cu propagare înapoi a erorilor ajustează ponderile pe baza ideii de minimizare a erorii pătratice. Funcţia de activare diferenţiabilă permite algoritmului cu propagare înapoi să ajusteze ponderile pentru toate straturile de neuroni. Având mai multe noduri in fiecare strat, problemele n-separabile ca şi sau-exclusiv (sau XOR), care nu se puteau rezolva doar cu perceptronul, pot fi rezolvate cu reţele cu transmitere înainte. În figura de mai jos se poate observa o reţea neuronală cu transmitere înainte în care, de la intrare până la ieşire, fiecare nod este conectat cu toate nodurile din straturile adiacente:

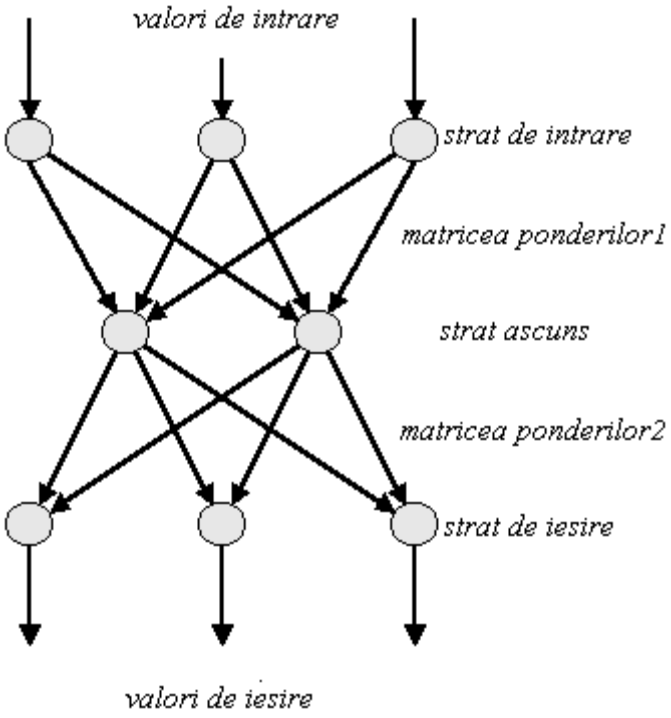


**Figura 7.3. Reţea cu transmitere înainte**

Fiecare neuron din stratul neuronal de intrare are o valoare de activare care reprezintă un stimul extern. Un neuron de intrare trimite valoarea sa de activare fiecărui neuron din stratul ascuns cu care este conectat. Fiecare neuron al stratului ascuns calculează propria sa valoare de activare depinzând de valorile de activare pe care le primeşte de la neuronii de intrare. Acest semnal este apoi trimis neuronilor din stratul de ieşire sau următorului strat ascuns de neuroni, în cazul în care există mai multe straturi ascunse. Aceşti neuroni din straturile ascunse calculează valorile de activare în acelaşi fel şi le trimit mai departe vecinilor. În cele din urmă semnalul provenind de la neuronii de intrare se propagă prin reţea pentru a determina valorile de activare pentru toţi neuronii de ieşire.

Activările neuronilor sunt determinate de ponderile conexiunilor dintre neuroni. Ponderile pot fi pozitive sau negative. O pondere negativă reprezintă inhibarea neuronului care primeşte informaţa, de către cel care o trimite. Valoarea de activare pentru fiecare neuron este calculată conform unei funcţii de activare simple. Funcţiile de activare variază în detaliu, dar sunt toate conforme aceluiaşi plan de bază. Funcţia însumează contribuţiile tuturor neuronilor din stratul anterior, contribuţia unui neuron fiind ponderea conexiunii dintre neuronul din stratul anterior şi cel care primeşte înmulţită cu valoarea de activare a neuronului care trimite. Această sumă este de obicei modificată mai departe, de exemplu prin scalarea la o valoare între 0 şi 1, sau prin setarea valorii de activare la 0 dacă nu este atinsă o anumită valoare prag. Deoarece toţi neuronii calculează aceeaşi funcţie de activare, realizările intelectuale depind în primul rând de valorile ponderilor dintre neuroni.

Aşadar în reţelele neuronale cu transmitere înainte, activarea este transmisă direct de la stratul de intrare la cel ascuns şi apoi la cel de ieşire. În modelele mai realiste ale

creierului, însă, există mai multe straturi ascunse de neuroni şi de conexiuni recurente care transmit semnalele înapoi de la nivelele mai înalte, la cele mai de jos. Această recurenţă este necesară pentru aputea explica anumite procese cognitive, cum ar fi memoria pe termen scurt. În reţelele neuronale cu transmitere înainte, prezentarea repetată a aceloraşi intrări produce de fiecare dată aceeaşi ieşire, dar chiar şi cele mai simple organisme învaţă sa ignore prezentarea repetată a aceluiaşi stimul.

## Algoritmul de Propagare Înapoi (Backpropagation)

Algoritmul backpropagation este cel mai folosit algoritm de antrenare pentru reţelele neuronale multistrat.

Reţelele neuronale multistrat sunt formate în mod normal din trei sau patru straturi, având un strat de intrare, unul de ieşire şi de obicei un strat ascuns, deşi în cazuri mai rare sunt necesare două straturi ascunse. Termenul de strat de intrare este impropriu, deoarece acestui strat nu-i este aplicată funcţia de activare neuronală., ci valorile de intrare sunt trimise stratului ascuns fără a fi modificate în vreun fel. În calcularea valorii fiecărui neuron din stratul ascuns şi din cel de ieşire trebuie mai întâi să calculăm produsul vectorial dintre intrările neuronului şi ponderile asociate acestora, iar apoi să aplicăm rezultatului o funcţie (de exmplu sigmoidală) pentru a calcula activarea neuronului. Ieşirea neuronului este tocmai această activare.

Reţeaua învaţă apoi prin modificarea tuturor ponderilor. Calculând derivatele parţiale ale erorii în raport cu fiecare pondere, vom observa direcţia de îndreptare a erorii reţelei. Dacă adunăm apoi la veche pondere derivatele corespunzătoare cu semn schimbat, eroarea va descreşte. Acest lucru are sens pentru că dacă derivata este pozitivă, înseamnă că eroarea creşte odată cu ponderea şi de aceea firesc este să adunăm la ponderea respectivă o valoare negativă. Vice versa dacă derivata este negativă. Deoarece calculul acestor derivate parţiale şi aplicarea lor fiecărei ponderi se face începând de la startul de ieşire la cel ascuns şi apoi de la stratul ascuns la cel de intrare (acest lucru este necesar deoarece pentru modificarea acestui set de ponderi avem nevoie de derivatele parţiale calculate pentru stratul anterior) algoritmul poartă numele de algoritmul cu propagare înapoi (sau backpropagation algorithm).

## Antrenarea cu backpropagation

Aşadar antrenarea folosind backpropagation presupune transmiterea exemplelor de antrenare ca vectori de intrare prin reţea, calcularea erorii la ieşire şi ajustarea ponderilor reţelei pentru a minimiza eroarea.

Antrenarea poate fi oprită dacă eroarea scade sub un anumit prag (de exmplu e bun un prag de 0.001 pentru eroarea pătratică. Acesta variază de la o problemă la alta şi în unele cazuri s-ar putea să nici nu se obţină o eroare pătratică mai mică sau egală cu 0.001). Antrenarea excesivă poate avea rezultate dăunătoare deoarece reţeaua poate deveni prea adaptată la exemplele din setul de antrenare şi va fi incapabilă să clasifice corect exmple din afara acestuia.

Exemplele din afara setului de antrenare formează setul de validare. Acestea sunt de fapt exemplele pe care putem măsura performanţa reţelei. Nu putem aprecia performanţa doar bazându-ne pe succesul reţelei în învăţarea unui anumit set de

antrenare. Trebuie facute teste pentru a confirma faptul că reţeaua este capabilă să clasifice şi exemple din afara setului de antrenare.

**Descrierea algoritmului**

Primul pas constă în iniţializarea ponderilor cu valori aleatoare, de obicei între -0.5 şi 0.5, sau între-1 şi 1, iar apoi transmiterea vectorului de intrare prin reţea şi calcularea valorii corespunzătoare fiecărui neuron din reţea. Aceasta se face, aşa cum am mai amintit, prin calcularea produsului vectorial dintre intrările neuronului şi ponderile corespunzătoare acestora şi apoi aplicarea funcţiei sigmoidale rezultatului. Fie *x* vectorul activărilor (ieşirilor) stratului anterior (sau vectorul intrărilor stratului curent). Atunci are loc:

$$o = \sigma(\vec{w}\vec{x})$$
$$\sigma(y) = \frac{1}{1+e^{-y}}$$

Vectorul *w* este vectorul ponderilor conexiunilor dintre stratul curent şi stratul anterior.

Cel de-al doilea pas este calcularea erorii pătratice a reţelei. Aceasta se calculează prin însumarea pătratelor erorilor pentru fiecare neuron din stratul de ieşire. Vectorul ţintă care apare în calculul erorii, este asociat exemplului de antrenare (vectorului de intrare).

$$E(\vec{w}) = \frac{1}{2}\sum_{k \in outputs}(t_k - o_k)^2$$

Valorile *t* denotă valorile din vectorul ţintă, iar *o* reprezintă activările neuronilor din stratul de ieşire.

Cel de-al treilea pas constă în cacularea erorii pentru fiecare neuron de ieşire, indicată mai jos prin δ.

$$\delta_k \leftarrow o_k(1-o_k)(t_k - o_k)$$

Această eroare este de fapt derivata parţială a erorii în raport cu fiecare pondere.

Pasul al patrulea este calcularea erorii fiecărui neuron din stratul ascuns.

$$\delta_h \leftarrow o_h(1-o_h)\sum_{k \in outputs} w_{kh}\delta_k$$

Eroarea unui neuron din stratul ascuns depinde de erorile calculate pentru neuronii de ieşire.

Pasul al cincilea constă în calcularea valorilor cu care se vor ajusta ponderile. η reprezintă rata de învăţare. O rată de învăţare mică poate asigura o convergenţă mai stabilă, pe când o valoare mare a acesteia poate grăbi convergenţa.

$$\Delta w_{ji} = \eta \delta_j x_{ji}$$

Valoarea x reprezintă neuronul conectat cu neuronul curent prin conexiunea având ponderea w.

Ultimul pas reprezintă adunarea la fiecare pondere a valorilor calculate anterior.

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

După terminare, se revine la pasul întâi. Algoritmul se opreşte când eroarea devine mai mică sau egală cu pragul stabilit, sau când se atinge un număr maxim de iteraţii.

**Inconveniente ale algoritmului backpropagation**

- Timpul de antrenare mare datorat numărului mare de iteraţii.

- Modificarea lentă a ponderilor.
- Posibilitatea blocării într-un minim local.
- Posibilitate ca reţeaua să nu conveargă din cauza numărului mare de perechi de vectori de antrenare.

**Aplicaţii ale reţelelor neuronale**

Reţelele neuronale sunt utilizate cu succes în acele domenii în care programarea clasică, algoritmică, sau chiar inteligenţa artificială tradiţională cu metode simbolice, au eşuat, sau s-au dovedit ineficiente. Astfel, reţelele neuronale sunt aplicate problemelor al căror algoritm sau model nu este cunoscut, sau a căror rezolvare, folosind algoritmi cunoscuţi, durează foarte mult.

Aplicaţiile pentru care reţelele neuronale dau rezultatele cele mai bune, cuprind:

- Aproximări de funcţii.
- Predicţii.
- Clasificări.
- Asocieri de date.
- Optimizări.
- Filtrări de date.
- Luări de decizii.
- Recunoaştere de voce.

şi multe alte probleme care intră în categoria recunoaşterii unor modele sau tipare.

Cea mai comună utilizare a reţelelor neuronale este pentru proiecţia a ceea ce urmează să se întâmple. În multe aplicaţii predicţiile pot ajuta la stabilirea priorităţilor. De exemplu în camera de urgenţă a unui spital este foarte important de stabilit cine are nevoie mai rapid de ajutor medical pentru a permite o operaţie cu succes. Practic, fiecare organizaţie trebuie să îşi stabilească anumite proritǎţi care să guverneze alocarea resurselor acesteia. Această nevoie de proiecţie a viitorului a dus la crearea reţelelor pentru predicţii.

O altă utilizare importantă şi foarte frecventă a reţelelor neuronale este pentru clasificări. O reţea care poate face clasificări, poate fi folosită în industria medicală pentru a prelucra atât rezultate de laborator, cât şi înregistrări ale simptomelor pacienţilor pentru a determina cea mai probabilă boală.

Recent, au fost descrise în presă câteve aplicaţii de recunoaşteri de modele. Unul din acestea, era un sistem care poate detecta bombe în bagaje în aeroporturi, identificând modelele din nişte senzori specializaţi. Un alt articol a raportat cum un medic a antrenat o reţea neuronală feedforward cu date colectate în camere de urgenţă de la oameni care simţeau că au un atac de cord şi dădea ca rezultat probabilitatea ca atacurile de cord să fie reale şi nu alarme false.

În continuare vom prezenta câteva exemple de aplicaţii ale reţelelor neuronale:

- **Aplicarea reţelelor neuronale în recunoaşteri de caractere**

  Ideea recunoaşterii de caractere a devenit foarte importantă, deoarece dispozitivele pentru scrisul de mână au devenit tot mai populare. Utilizarea reţelelor feedforward în recunoşterea caracterelor scrise de mână este mai veche. Ca şi în majoritatea cazurilor de învăţare supervizată, este prezentat

reţelei un model de intrare, în acest caz matricea caracterului scris de mână, împreună cu ieşirea dorită, aici litera sau cifra corectă. Antrenarea se face cu modele de scris de mână ale utilizatorului.

- **Aplicarea reţelelor neuronale în comprimarea imaginilor**

Pentru că pot accepta un şir mare de date de intrare şi le pot procesa rapid, reţelele neuronale sunt utile şi în comprimarea imaginilor. O arhitectură a unei reţele neuronale pentru comprimare de imagini are dimensiunea stratului de intrare egală cu cea a stratului de ieşire şi un strat intermediar cu o dimensiune mai mică. Rata de comprimare este tocmai raportul dintre dimensiunea stratului de intrare şi dimensiunea stratului intermediar. Scopul acestor reţele pentru comprimarea datelor este de a recrea intrarea. Aşadar, în antrenare, imaginea folosită ca intrare, va fi şi ieşirea dorită. Imaginea va fi o matrice de dimensiunea 256x256, iar intrările reţelei vor fi matrici de dimensiuni 8x8 din imaginea iniţială, alese aleator. Acestea sunt prezentate reţelei repetat, iar ponderile sunt ajustate continuu, până când reţeaua reproduce cât mai fidel imaginea.

Odată terminată antrenarea, reconstrucţia imaginii este demonstrată în faza de testare. În acest caz, reţelei îi sunt prezentate tot părţi de dimensiuni 8x8 din imagine, dar selectate secvenţial, de la stânga la dreapta şi de sus în jos. Pentru fiecare astfel de matrice de 8x8, este calculat şi afişat pe ecran rezultatul pentru a se putea observa vizual performanţa reţelei.

- **Reţelele neuronale în predicţii financiare**

Reţelele neuronale s-au dovedit a fi instrumente puternice în predicţiile de pe piaţa de capital. Multe companii au obţinut profituri foarte mari folosind reţele neuronale şi le-au folosit cu multă uşurinţă pentru predicţii ale preţurilor acţiunilor.

Ideea predicţiilor pe piaţa financiară nu este nouă. Oamenii de afaceri încearcă deseori să anticipeze preţul pieţei, interpretând parametri externi cum ar fi indicatorii economici, opinia publică şi climatul politic curent. Întrebarea este dacă reţelele neuronale pot descoperi anumite tendinţe în datele de intrare, pe care oamenii s-ar putea să nu le observe şi să folosească aceste tendinţe cu succes în predicţiile lor. Au fost obţinute rezultate bune folosind o reţea relativ simplă, cu doar 6 indicatori financiari ca intrări şi cu trei straturi neuronale. Reţeaua a fost antrenată cu algoiritmul backpropagation pe un volum mare de date anterioare. Rezultate şi mai bune au fost obţinute cu o reţea neuronală având 2 straturi ascunse şi mai mult de 6 variabile de intrare.

- **Aplicarea reţelelor neuronale în medicină**

Reţelele neuronale sunt un domeniu de cercetare important şi în medicină şi se crede că vor avea o mare aplicabilitate în sistemele biomedicale în următorii ani. Pentru moment, cercetătorii se ocupă cu modelarea anumitor

părţi ale corpului uman şi recunoaşterea unor boli. Reţelele neuronale sunt ideale în stabilirea de diagnostice, deoarece nu au nevoie de un algoritm precis pentru a identifica o anumită boală. Ele învaţă din exemple, aşa că nu sunt necesare detalii legate de modul de recunoaştere a bolii. Este nevoie doar de un set de exemple reprezentativ pentru toate variaţiile bolii. Acestea trebuie alese cu multă grijă pentru ca sistemul să fie cât mai eficient şi de încredere.

Unul dintre domeniile care a intrat în atenţie în ultima vreme, este diagnosticarea cardiopulmonară. Modul în care reţelele neuronale funcţionează în acest domeniu sau în alte domenii ale diagnosticării medicale, este prin compararea mai multor modele diferite. Datele de intrare pot include pulsul, presiunea arterială, respiraţia, vârsta, sexul, activitatea fizică, etc. pentru diferiţi pacienţi. Reţeaua neuronală poate învăţa prin studieraea diferitelor condiţii şi modele, determinându-le să formeze un tablou conceptual complet, putând apoi să diagnosticheze pacientul bazându-se pe aceste modele.

Reţelele neuronale mai sunt frecvent folosite şi în diagnosticarea cancerului la sân. Celulele canceroase sunt examinate în mod tradiţional la microscop, de către om, iar acesta decide gradul cancerului prezent. Însă oamenii sunt inconsistenţi în astfel de judecăţi. Reţelele neuronale au obţinut rezultate corecte între 52% si 89% din cazurile noi, neprezentate anterior reţelei.

- **Reţele neuronale folosite în împrumuturi**

Acordarea de împrumuturi este un domeniu în care reţelele neuronale pot ajuta oamenii, deoarece nu este un domeniu bazat pe criterii bine definite. Băncile vor să aibă cât mai mulţi bani, iar o modalitate de a-i avea este de a diminua rata eşecurilor folosind reţele neuronale pentru a decide dacă să acorde sau nu un împrumut. Reţelele neuronale sunt cu atât mai utile în acest domeniu cu cât nici un proces nu va garanta o precizie de 100%. Chiar o precizie de 85-90% ar fi o îmbunătăţire faţă de metodele folosite de om. În unele bănci rata de eşec în cazul împrumuturilor aprobate folosind reţele neuronale este mai mică decât în cazul celor acordate folosind cele mai bune metode tradiţionale.

Procesul funcţionează prin analizarea eşecurilor anterioare, iar deciziile curente se bazează pe experienţa trecută. Singura problemă este justificarea deciziei luate faţă de cel care solicită împrumutul. Însă explicarea modului în care a învăţat reţeaua şi pe ce caracteristici se bazează deciziile luate, este dificilă.

- **Folosirea reţelelor neuronale în marketing**

Reţelele neuronale au fost folosite şi în aplicaţii de marketing. Una din aceste aplicaţii este cea de împărţire a locurilor pentru o companie aeriană. Aplicaţia este un sistem format din mai multe tehnologii inteligente. În acest sistem este integrată o reţea neuronală feedforward care a fost antrenată

folosind algoritmul backpropagation. Mediul s-a schimbat rapid şi constant şi de aceea abordarea neuronală adaptivă a fost o soluţie bună. Sistemul a fost folosit pentru a monotoriza şi a recomanda rezervările pentru fiecare zbor. Astfel de informaţii au un impact direct asupra profitului unei companii aeriene şi dau un avantaj tehnologic pentru utilizatorii sistemului.

### 7.2.1.3 Învăţarea prin arbori de decizie

Învăţarea prin arbori de decizie este una dintre cele mai simple şi totuşi una dintre cele mai de succes metode de învăţare. Este o bună introducere pentru aria învăţării inductive şi este uşor de implementat.

Arborii de decizie primesc la intrare o situaţie, descrisă printr-un set de proprietăţi, şi au ca ieşire o decizie de forma "*Da / Nu*". De aceea arborii de decizie reprezintă funcţii booleene. Fiecare nod intern din arbore corespunde unui test al valorii unei proprietăţi şi fiecare ramură a nodului este etichetată cu o valoare posibilă a testului. Fiecare nod frunză din arbore specifică o valoare booleană ce va fi returnată dacă se va merge pe un drum în arbore ce conduce la acel nod.

Ca un exemplu, să considerăm următoarea problemă: să se aştepte sau nu pentru o masă în restaurant. Predicatul scop este *VomAştepta* iar definiţia acestui predicat este exprimată printr-un arbore de decizie. Pentru a putea defini această problemă de învăţare mai întâi trebuie să vedem care dintre proprietăţi sau *atribute* sunt valabile pentru a putea descrie exemplele din domeniu. Să presupunem că avem următoarea listă de atribute:

1. *Alternativă*: dacă este un restaurant acceptabil în apropiere.
2. *Vineri / Sâmbătă*: dacă este vineri sau sâmbătă.
3. *Bar*: dacă restaurantul are bar unde se poate aştepta.
4. *Apetitul*: dacă clienţilor le este foame.
5. *Clienţi*: câte persoane sunt în restaurant (valorile sunt: *Nimeni, Câţiva, Plin*).
6. *Preţ*: preţurile din restaurant. (valorile sunt: $, $$, $$$)
7. *Ploaie*: dacă plouă afară.
8. *Rezervare*: dacă s-a făcut rezervare sau nu.
9. *Tipul*: tipul restaurantului (valorile sunt Francez, Chinezesc, Italian).
10. *PerioadaDeAşteptare*: perioada de aşteptare pentru o masă. (valorile sunt: 0-10 minute, 10-30 minute, 30-60 minute, mai mult de o oră).

Arborele poate fi exprimat ca o conjuncţie de implicaţii individuale care corespund drumurilor prin arbore până la nodurile frunză care au valori "*Da*". De exemplu, drumul pentru un restaurant plin de clienţi, cu un timp de aşteptare de 10-30 de minute, când agentului nu îi este foame, este exprimat prin următoarea propoziţie logică:

$$\forall \textit{Clienţi}(r, \textit{Plin}) \wedge \textit{PerioadaDeAşteptare}(r, 0\text{-}10) \wedge \textit{Apetit}(r, N) \Rightarrow \textit{VomAştepta}(r)$$

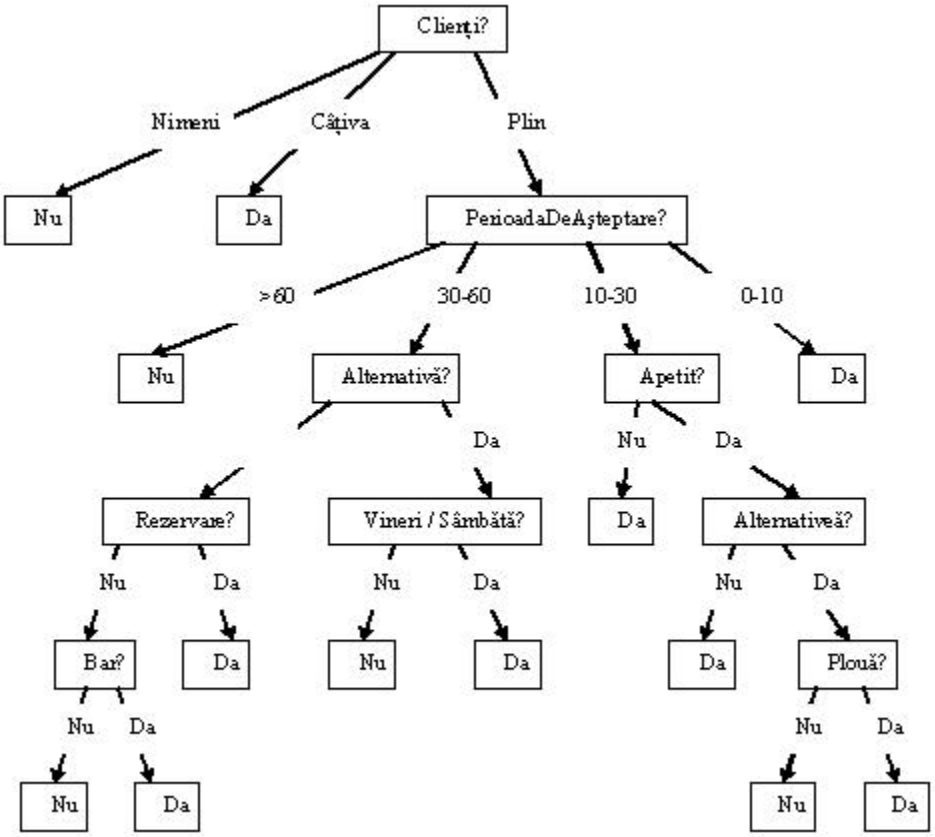


**Figura 7.4 Un arbore de decizie pentru a decide dacă se aşteaptă sau nu pentru o masă**

Arborii de decizie sunt buni pentru anumite tipuri de funcţii şi nepotriviţi pentru alte tipuri de funcţii. Nu există nici o reprezentare eficientă pentru toate tipurile de funcţii. În cazul în care considerăm un set de funcţii booleene cu *n* atribute sunt tot atâtea tabele de adevăr pentru aceste funcţii. Un tabel de adevăr are $2^n$ coloane deoarece fiecare intrare este descrisă de *n* atribute, ceea ce înseamnă că sunt ${}_2 2^n$ funcţii diferite cu *n* atribute.

De exemplu, dacă avem numai şase atribute booleene, sunt aproximativ $2 * 10^{19}$ funcţii diferite. Va fi nevoie de nişte algoritmi ingenioşi pentru a găsi ipoteze bune într-un spaţiu atât de mare.

Ideea de bază pentru algoritmul *Învăţarea prin arbori de decizie* este ca prima oară să fie testat cel mai important atribut. Prin cel mai important atribut se înţelege acel atribut care realizează cea mai semnificativă diferenţă în clasificarea exemplelor. În acest mod, se speră găsirea unei clasificări corecte cu un număr mic de teste, aceasta însemnând că drumurile în arbore vor fi scurte şi întregul arbore va fi mic.

**Definiţie:** Se numeşte *clasificare* a unui exemplu valoarea predicatului scop. Dacă predicatul scop este adevărat pentru un exemplu, acesta se numeşte *exemplu pozitiv*, altfel se numeşte *exemplu negativ*.

În Figura 7.5 sunt descrise cinci exemple pentru problema restaurantului. Exemplele pozitive sunt cele pentru care valoarea predicatului *VomAştepta* este adevărat ($X_1$, $X_3$, $X_4$ ) iar cele negative sunt cele pentru care această valoare este fals ($X_2$, $X_5$, $X_6$ ). Setul complet de exemple se numeşte *setul de instruire*.

| Exemple | Atribute | | | | | | | | | | Scop |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Alternativă | Bar | Vineri | Apetit | Clienţi | Preţ | Plouă | Rez. | Tip | Timp | VomAştepta |
| $X_1$ | da | nu | nu | da | câţiva | $$$ | nu | da | Francez | 0-10 | da |
| $X_2$ | da | nu | nu | da | plin | $ | nu | nu | Chinezesc | 30-60 | nu |
| $X_3$ | nu | da | nu | nu | câţiva | $ | nu | da | Italian | 0-10 | da |
| $X_4$ | da | nu | da | da | lin | $$$ | nu | da | Chinezesc | 10-30 | da |
| $X_5$ | da | nu | da | nu | plin | $$ | nu | da | Francez | >60 | nu |
| $X_6$ | nu | nu | nu | nu | nimeni | $ | nu | nu | Chinezesc | 0-10 | nu |

**Figura 7.5 Exemple pentru problema restaurantului**

Avem cele cinci exemple de instruire pe care le-am clasificat în exemple pozitive şi exemple negative. Apoi trebuie să decidem care este atributul cel mai semnificativ. Figura 7.5 arată că atributul *Clienţi* este un atribut important, deoarece dacă valoarea lui este *câţiva* sau *nimeni* atunci ne rămâne un set pentru care putem da un răspuns; pentru atributul *plin* mai avem nevoie de câteva teste. Se observă că atributul *Tip* este un atribut irelevant, deoarece nu reduce domeniul căutărilor. Se cercetează astfel toate atributele şi se alege ca rădăcină a arborelui acel atribut care este cel mai important. Am ales atributul *Clienţi*.

După ce a fost ales primul atribut fiecare ieşire este la rândul ei un arbore de decizie, dar cu mai puţine exemple şi mai puţine atribute. Sunt patru cazuri:

- Dacă mai sunt exemple negative şi pozitive trebuie ales cel mai bun atribut pentru a le împărţi. Se alege *Apetit*.
- Dacă toate exemplele rămase sunt fie pozitive, fie negative, am terminat: se poate răspunde prin *da* sau *nu*. Exemple de felul acesta sunt cazurile *nimeni* şi *câţiva*.
- Dacă nu au mai rămas exemple în setul de instruire, înseamnă că nu a mai fost observat nici un exemplu şi se va returna o valoare implicită calculată din majoritatea clasificărilor nodurilor părinţi.
- Dacă nu au mai rămas atribute, dar nu se poate lua o decizie atunci înseamnă că avem exemple cu aceeaşi descriere dar clasificări diferite. Acest lucru se întâmplă când unele date sunt incorecte sau când atributele nu oferă destule informaţii pentru a putea descrie complet situaţia.

S-ar putea trage concluzia că algoritmul de învăţare nu se comportă conform aşteptărilor în învăţarea funcţiei corecte. Această concluzie este greşită deoarece algoritmul de învăţare ia în considerare exemplele, nu cunoaşte funcţia corectă.

Algoritmul de învăţare în arbori de decizie este descris Figura 7.6.

```
funcţia Învăţare_Arb(exemple, atribute, implicit) returnează un arbore de decizie
intrări: exemple        //setul de exemple
         atribute       //setul de atribute
         implicit       //valoarea implicită pentru predicatul scop
 dacă nu mai există exemple atunci Învăţare_Arb ← implicit altfel
  dacă toate exemplele au aceeaşi clasificare atunci Învăţare_Arb ← clasificarea altfel
    dacă nu mai există atribute
       atunci
          Învăţare_Arb ← Majoritatea_Valorilor(exemple)
       altfel
          cel_mai_bun ← Alege_Atribut(atribute, exemple)
          arbore ← un nou arbore de decizie cu rădăcina cel_mai_bun
          pentru fiecare valoare v_i din cel_mai_bun execută
                exemple_i ← {elementele din exemple cu cel_mai_bun = v_i}
                subarbore ← Învăţare_Arb(exemple_i, atribute – cel_mai_bun,
                                         Majoritatea_Valorilor(exemple))
                @adaugă o nouă ramură la arbore cu eticheta v_i şi subarborele subarbore
          sf-pentru
    sf-dacă
  sf-dacă
 sf-dacă
 Învăţare_Arb ← arbore
sf-Învăţare_Arb
```

**Figura 7.6. Algoritmul de învăţare în arbori de decizie**

### 7.2.2 Învăţarea Prin Întărire

În această secţiune vom aborda problematica generală a învăţării prin întărire, fără a intra în amănunte legate de algoritmi specifici.

Problema învăţării prin întărire este problema generală de îmbunătăţire a comportamentului unui agent artificial pe baza unui feed-back al performanţei sale.

Vom studia în acest capitol modul în care agenţii pot învăţa în condiţii mai puţin favorabile, când ei nu primesc exemple şi pornesc fără un model al mediului înconjurător şi fără a avea o funcţie de performanţă. *Învăţarea prin întărire* (*reinforcement learning*) este modalitatea de a îmbunătăţi comportarea unui agent pe baza unui anumit feed-back primit în legătura cu performanţa sa.

Învăţarea prin întărire este o abordare a învăţării care combină două discipline în scopul rezolvării unor probleme pe care nici una nu o poate rezolva separat.

- *Programarea dinamică* este un domeniu al matematicii utilizat îndeosebi în probleme de optimizare şi control.
- *Învăţarea supervizată* este o metodă generală de instruire a unor reţele (care aproximează funcţii – cum ar fi reţelele neuronale) pentru a reprezenta funcţii.

În esenţă, învăţarea supervizată necesită perechi (intrare-ieşire) pentru funcţia ce urmează a fi învăţată (un set de întrebări cu răspunsurile corespunzătoare).

Într-un sistem de învăţare prin întărire calculatorului i se dă pur şi simplu sarcina pe care trebuie să o realizeze. Sistemul va învăţa cum să-şi realizeze sarcina (să-şi atingă scopul) prin încercări şi eventuale feedback-uri (răspunsuri) pe care le primeşte de la mediu.

Un exemplu este un agent care învaţă să joace şah fără a fi supervizat, adică fără a avea nişte exemple de situaţii de joc împreună cu cele mai bune mutări alese în situaţiile respective. În situaţia în care nu există un supervizor care să-i furnizeze exemplele, ce va face agentul? Încercarea aleatoare a mutărilor, ar fi total ineficientă, atâta timp cât numărul de mutări la un moment dat este mare, procesul devenind exponenţial. Fără un feedback despre ce e bine şi ce e rău, agentul nu ar putea să decidă la un moment dat cea mai bună mutare care ar trebui aleasă. Există totuşi un fel de feedback pe care-l obţine agentul la sfârşitul jocului – chiar în absenţa unui supervizor –, când percepe dacă a câştigat sau a pierdut. Acest feedback poartă numele de *recompensă (reward)* sau *întărire*, iar agentul îl primeşte fie la sfârşit, într-o stare finală, fie pe parcurs când are informaţii exacte despre ce a făcut bine sau nu.

Rolul *învăţării prin întărire* este de a folosi recompense pentru învăţarea cu succes a unui agent funcţional.

Învăţarea prin întărire poate fi văzută ca un microcosmos pentru toate problemele de Inteligenţă Artificială. Un agent primeşte nişte percepţii de la mediul în care se află, le marchează ca fiind utilităţi pozitive sau negative si apoi se decide ce acţiune să execute (Figura 7.7).

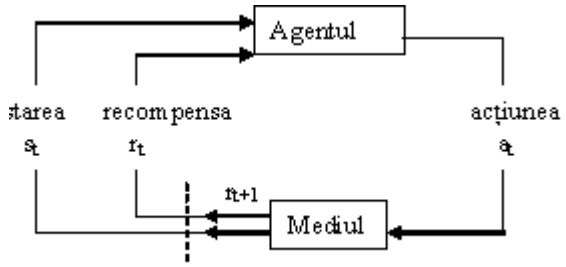


**Figura 7.7. Interacţiunea dintre agent şi mediu în învăţarea prin întărire**

La fiecare pas de timp *t*, agentul primeşte informaţii despre starea în care se află, $s_t \in S$, unde *S* este mulţimea stărilor posibile, şi pe baza acestei informaţii selectează acţiunea $a_t \in A(s_t)$ unde $A(s_t)$ este mulţimea de acţiuni disponibile din starea $s_t$. La următorul pas, în parte ca o consecinţă a acţiunii efectuate, agentul primeşte o recompensă numerică $r_{t+1} \in R$, şi se află într-o nouă stare $s_{t+1}$. La fiecare pas de timp agentul efectuează o mapare de la stări la probabilităţile de a alege o acţiune posibilă. Această mapare se numeşte *politica* agentului notată cu $\pi_t$, unde $\pi_t(s, a)$ este probabilitatea ca $a = a_t$ şi $s = s_t$. Metodele de învăţare prin întărire specifică cum ar trebui agentul să-şi modifice politica ţinând cont de experienţa acumulată.

Învăţarea prin întărire ne arată cum poate învăţa un agent şi cum poate deveni expert într-un mediu necunoscut, când i se dau numai recompense ocazionale. Scopul Învăţării prin Întărire este de a folosi recompensa primită pentru ca agentul să înveţe să facă acţiuni corecte. Recompensa e un număr şi valoarea acesteia variază de la pas la

pas. Scopul agentului este de a maximiza totalul recompenselor primite. Agentul trece printr-o succesiune de stări si primeşte o anumita recompensă. Aceasta poate fi recompensa de câştig (+1 de exemplu), de eşec (-1) sau nulă când nu s-a ajuns la o stare terminală (0).

Agentul învaţă să-şi maximizeze recompensele primite. Dacă dorim ca agentul să facă un anumit lucru trebuie să stabilim recompensele astfel încât agentul sa ajungă acolo unde vrem maximizând aceste recompense. Dar recompensa care îi va fi atribuită agentului pentru o acţiune trebuie sa-i spună acestuia ce dorim ca el sa obţină nu cum să obţină. Învăţarea prin Întărire e sinonima cu învăţarea prin interacţiune. În timpul învăţării sistemele adaptive încearcă aplicarea de acţiuni asupra mediului în care se află, iar apoi sunt recompensate in funcţie de acţiunea pe care o executa. Algoritmii de învăţare prin întărire reţin selectiv ieşirile care maximizează recompensa primită. În multe medii complexe învăţarea prin întărire este singura metodă de a antrena programe să îndeplinească sarcini cu un grad mare de dificultate. De exemplu, la jocuri este extrem de greu pentru un antrenor uman să dea evaluări exacte şi consistente ale unui număr mare de poziţii, care ar fi necesare pentru a antrena o funcţie de evaluare direct din exemple. În loc de acest lucru, programului i se poate spune când a câştigat sau a pierdut, şi acesta poate folosi aceste informaţii pentru a-şi construi o funcţie de evaluare care estimează aproape exact probabilitatea de reuşită sau eşec dintr-o anumită poziţie. Un agent primeşte percepţii de la mediul în care se află, le clasifică în utilităţi pozitive şi negative, iar apoi se decide ce acţiune să aleagă.

O problemă de învăţare prin întărire are trei părţi fundamentale:

- *mediul* – reprezentat sub formă de stări.
- *funcţia de întărire* – funcţia exactă a recompenselor pe care agentul caută să le maximizeze. Agentul va învăţa să execute acele acţiuni care să maximizeze suma recompenselor primite pe drumul de la o stare iniţială spre o stare finală. Există trei clase utilizate pentru construirea unor funcţii de întărire corespunzătoare scopului urmărit.
  - **Probleme de evitare** sau de **recompensă amânată** – recompensele sunt 0, cu excepţia stărilor terminale. Semnul recompensei într-o stare terminală va indica dacă acea stare e finală (succes) sau trebuie evitată (eşec) – probleme de teoria jocurilor.
  - **Timp minim de obţinere a soluţiei** – scopul este ca agentul să aleagă acele acţiuni care să genereze cel mai scurt drum către o stare finală. În acest caz funcţia de întărire va fi – 1 pentru toate tranziţiile dintre stări, cu excepţia tranziţiei spre o stare finală, caz în care recompensa va fi 0. Maximizarea recompenselor se va reduce în acest caz la minimizarea numărului de tranziţii (a lungimii drumului spre o stare finală).
  - **Jocuri** – în cazul jocurilor cu doi sau mai mulţi adversari, se va folosi o funcţie de întărire alternantă. Sistemul va învăţa să genereze comportări optimale ale jucătorilor, în sensul maximizării sau minimizării funcţiei de întărire. Problemele de joc se reduc de fapt la căutări în arbori (de joc) în care alternează două nivele: cel maximizant corespunzător primului jucător, respectiv cel minimizant corespunzător celui de-al doilea jucător.
- *funcţia de utilitate (valoare)* – **Strategia de control** va desemna ce acţiuni vor trebui efectuate în fiecare stare. **Valoarea** (utilitatea) unei stări se va defini ca fiind suma

recompenselor primite pe drumul de la starea curentă către o stare finală. Scopul va fi maximizarea sumei recompenselor obţinute pe drumul de la o stare iniţială arbitrară către o stare finală, în stările accesibile pe acest drum. *Funcţia de utilitate* va fi deci o asignare de valori stărilor şi va putea fi aproximată folosind orice tip de aproximator de funcţii (perceptron, etc.).

Marea problemă ce se pune în general în cercetările de învăţare prin întărire este: Care este şablonul unui algoritm care să găsească valorile optime ale funcţiei de utilitate? Dintr-un anumit punct de vedere *învăţarea prin întărire* este o nouă exprimare a problemelor de Inteligenţă Artificială. Un agent într-un mediu primeşte semnale, le ataşează unora utilităţi pozitive sau negative şi apoi va trebui să decidă ce acţiune să aleagă. Sarcina de *învăţare* poate varia în funcţie de anumite situaţii:

- Mediul poate fi accesibil sau inaccesibil. Într-un mediu accesibil, stările pot fi identificate cu percepţii, pe când într-un mediu inaccesibil agentul va trebui să încerce să reţină un model al mediului;
- Agentul poate începe cu o anumită cunoaştere despre mediu şi despre efectele acţiunilor sale; sau va trebui să înveţe acest model la fel ca şi informaţiile de utilitate;
- Recompensele pot fi recepţionate doar în stările finale sau în orice altă stare;
- Recompensele pot fi componente ale funcţiei de utilitate (puncte pentru un agent care joacă ping-pong sau dolari pentru un agent de pariuri) pe care agentul încearcă să o maximizeze, sau pot fi sugestii despre funcţia de utilitate (“mutare bună” sau “mutare rea”);
- Agentul poate învăţa **pasiv** sau **activ**. Un agent pasiv se rezumă doar la a observa mediul şi a învăţa utilitatea de a fi în diverse stări; un agent activ va trebui să şi acţioneze conform cu informaţia ce a învăţat-o şi poate să-şi utilizeze generatorul de probleme pentru a sugera explorarea unor părţi necunoscute ale mediului.

Pe de altă parte, deoarece agentul va primi recompense referitoare la utilităţi, există două configuraţii de bază:

- Agentul învaţă o funcţie de utilitate (asociată stărilor) şi o foloseşte pentru a selecta acţiunea care să maximizeze performanţa preconizată a rezultatelor.
- Agentul învaţă **valorile acţiunilor (action-value function)**, furnizând performanţa preconizată în urma alegerii unei anumite acţiuni într-o anumită stare. Acest mod de învăţare se numeşte **Q-învăţare**.

Un agent care învaţă funcţii de utilitate va trebui să aibă un model al mediului, pentru a putea aplica deciziile luate, va trebui să cunoască stările în care se ajunge în urma aplicării acţiunilor. În schimb, un agent care învaţă valorile acţiunilor, nu are nevoie de un model al mediului; atâta timp cât îşi cunoaşte mutările permise, poate compara direct valorile acestora fără a trebui să cunoască stările rezultate în urma aplicării mutărilor. Din acest punct de vedere agenţii din a doua categorie sunt mai uşor de realizat decât cei din prima categorie. Pe de altă parte însă, pentru că nu cunosc unde conduc acţiunile lor, agenţii care învaţă valori de acţiune, nu pot reveni asupra acţiunilor lor, ceea ce este o serioasă restricţie asupra abilităţii lor de a învăţa.

Un exemplu clasic de învăţare prin întărire este **problema banditului cu n braţe**, problemă ce va fi descrisă în continuare.

Să considerăm următoarea problemă de învăţare: suntem puşi adesea în situaţia de a alege între *n* variante sau acţiuni. După fiecare acţiune primim o recompensă numerică aleasă dintr-o distribuţie de probabilitate staţionară care depinde de acţiunea selectată. Obiectivul este de a maximiza recompensa totală obţinută după o anumită perioadă de timp (de exemplu după 100 de acţiuni selectate). Fiecare selectare a acţiunii se numeşte *joc*.

Fiecare acţiune are o recompensă acordată dacă acţiunea este selectată. Aceasta se numeşte *valoarea* acţiunii respective. Dacă s-ar cunoaşte valoarea fiecărei acţiuni problema banditului cu *n* braţe ar fi foarte uşor de rezolvat. Dacă fiecare acţiune are asociată o anumită recompensă, atunci există o acţiune a cărei valoare a recompensei este cea mai mare. Selectarea acţiunii a cărei recompensă estimată este cea mai mare se numeşte *selecţia greedy* (şi se realizează prin metoda *greedy*). Dacă se selectează acţiunea care maximizează recompensa atunci se *exploatează* cunoştinţele obţinute despre valorile acţiunilor. Dacă nu se selectează acest tip de acţiune spunem că se *explorează*. Exploatarea are avantajele ei, dar explorarea ar putea produce un rezultat mai bun pe termen lung. Recompensa este mai mică pe termen scurt, în timpul explorării, dar pe termen lung, după ce au fost descoperite acţiunile mai bune acestea pot fi exploatate.

Deoarece nu este posibil să exploatăm şi să explorăm cu un singur selector de acţiuni, apare un conflict între explorare şi exploatare.

În cazuri practice, decizia de a explora sau exploata depinde de valorile precise ale estimărilor, de incertitudini şi de numărul de jocuri rămase de jucat. Sunt mai multe metode de combinare a exploatării cu explorarea pentru formulări particulare ale problemei banditului cu n braţe şi ale problemelor înrudite cu aceasta.

În problemele învăţării prin întărire, agenţilor li se face o descriere a stării curente şi trebuie să aleagă următoarea acţiune dintr-un set de acţiuni posibile pentru a maximiza feedback-ul primit după fiecare acţiune.

Înafară de agent şi mediu, pot fi identificate patru principale elemente ale sistemului de învăţare prin întărire: o politică, o funcţie recompensă, o funcţie de evaluare şi un model al mediului.

- **Politica** ($\boldsymbol{\pi}$) defineşte comportamentul agentului la un moment dat, este o funcţie de la stările percepute ale mediului la acţiunile ce trebuie efectuate din aceste stări. Ea corespunde la ceea ce în psihologie se numeşte set de reguli stimul-răspuns. În unele cazuri politica poate fi o funcţie simplă, pe când în altele poate implica calcule extrem de complicate (de exemplu într-un proces de căutare). Politica este nucleul unui agent de învăţare prin întărire în sensul că este suficientă pentru a determina comportamentul acestuia. O politică $\boldsymbol{\pi}$ este o mapare de la starea $s \in S$ şi acţiunea $a \in A(s)$ la probabilitatea $\boldsymbol{\pi}(s,a)$ de a efectua acţiunea a din starea s.
- **Funcţia recompensă** defineşte scopul, atribuie fiecărei stări percepute (sau pereche stare-acţiune) un singur număr, o recompensă, semnificând dezirabilitatea stării (perechii) respective. Obiectivul fundamental al unui agent de învăţare prin întărire este maximizarea recompensei primite pe termen lung. Funcţia recompensă semnalează agentului care sunt acţiunile bune şi care sunt cele greşite. Recompensele sunt caracteristicile fundamentale

ale problemei cu care se confruntă agentul şi de aceea funcţia de recompensă trebuie să fie fixă. Ea poate fi însă folosită pentru a modifica politica. De exemplu, dacă o acţiune selectată de politică este urmată de o recompensă mică, atunci politica se poate schimba astfel încât să fie aleasă o altă acţiune în momentul în care apare din nou aceeaşi situaţie.

Funcţia recompensă este reprezentată de funcţia Q: S×A →R şi estimează recompensa pentru fiecare pereche stare-acţiune. Deci, valoarea efectuării acţiunii ***a*** din starea ***s*** sub politica $\boldsymbol{\pi}$, notată $\mathbf{Q^{\pi}(s,a)}$, este recompensa aşteptată când se pleacă din starea **s** şi se urmează politica $\boldsymbol{\pi}$ **:**

$$Q^{\pi}(s,a) = E_{\pi}\{R_t \mid s_t = s, a_t = a\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\}$$

Q se mai numeşte funcţie acţiune-valoare pentru politica $\boldsymbol{\pi}$.

- În timp ce funcţia recompensă semnalează ce e bun într-un moment imediat, **funcţia de evaluare** semnalează ce este bun pe termen lung. Valoarea unei stări este recompensa totală pe care agentul se poate aştepta să o primească de-a lungul timpului, plecând din acea stare. O stare poate produce o recompensă imediată mică, dar să aibă totuşi o valoare mare deoarece este de obicei urmată de stări ce produc recompense mari (sau reciproc). Între recompense şi valori există o strânsă legătură în sensul că fără recompense nu ar exista valori şi unicul motiv pentru care valorile sunt estimate este obţinerea de recompense mai mari. Cu toate acestea, valorile sunt cele importante atunci când se iau decizii. Se aleg acţiunile care duc în stări cu valoare mare, nu cu recompensă mare, deoarece aceste acţiuni vor duce, pe termen lung, la o recompensa totală maximă. Din păcate, este mult mai greu a determina valorile decât recompensele. Recompensele sunt date direct de mediu, pe când valorile trebuie estimate şi reestimate din secvenţa de observaţii pe care agentul le face de-a lungul întregii sale vieţi. De fapt, cea mai importantă componentă a majorităţii algoritmilor de învăţare prin întărire este o metodă de estimare eficientă a valorilor.

  Valoarea unei stări s sub o politică $\boldsymbol{\pi}$**,** notată $\mathbf{V^{\pi}(s)}$ este recompensa aşteptată plecând din s, având in vedere faptul că agentul urmează politica $\boldsymbol{\pi}$**,** iar t este un pas oarecare de timp:

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\}$$

- Cel de-al patrulea element al unui sistem de învăţare prin întărire este un **model al mediului**, o reprezentare internă, de obicei simplificată, a mediului utilizată de agent pentru a prezice ce s-ar putea întâmpla ca rezultat al acţiunilor viitoare. Un agent mai complicat ar putea utiliza acest model pentru a-şi planifica viitorul curs al acţiunilor.

Mediul celui care învaţă poate fi modelat de un timp discret, de o stare finită şi de un proces de decizie Markov care poate fi reprezentat prin cvadruplul (S, A, P, r) unde S este un set de stări, A este un set de acţiuni, iar P: S×S×A→ [0,1] indică probabilitatea de a trece din starea $s_1$ în starea $s_2$ întreprinzând acţiunea a şi r: S×A→ R este o funcţie cu rezultat scalar, funcţia recompensă. Fiecare agent păstrează o politică $\boldsymbol{\pi}$ care translatează starea curentă în acţiunea dorită a fi realizată în acea stare.

**Q-Learning**

Au fost propuse mai multe strategii pentru învăţarea prin întărire care pot fi folosite de agenţi în dezvoltarea unei politici de maximizare a rezultatelor pozitive acumulate de-a lungul timpului. Pentru a evalua sistemul de clasificare paradigmatic pentru învăţarea prin întărire în sistemele multiagent, acesta este comparat cu algoritmul Q-Learning, care este conceput pentru a găsi o politică $\boldsymbol{\pi}$ care maximizează $\mathbf{V^{\pi}(s)}$ pentru toate stările **s** din **S**. Valorile Q sunt definite ca $Q^{\pi}(\mathbf{s}, \mathbf{a}) = V^{\mathbf{a}:\boldsymbol{\pi}}$, unde cu $\mathbf{a}:\boldsymbol{\pi}$ se notează secvenţa de alegere a acţiunii **a** în starea curentă, urmată de alegerea acţiunilor bazate pe politica $\boldsymbol{\pi}$. Acţiunea **a** în starea **s** este aleasă în aşa fel încât să se maximizeze recompensa:

$$V^{\pi^*}(s) = \max_{a \in A} Q^{\pi^*}(s,a), \quad \forall\, s \in S.$$

Politica $\boldsymbol{\pi}^*$ se numeşte politică optimală şi este mai mare sau egală decât toate celelalte politici. Deşi pot fi mai multe astfel de politici toate au aceeaşi funcţie de evaluare, numită funcţie valoare optimă, notată $\boldsymbol{V}^*$, definită prin:

$$V^*(s) = \max_{\pi} V^{\pi}(s), \quad \forall\, s \in S.$$

Politicile optime au şi aceeaşi funcţie acţiune-valoare, notată $Q^*$, definită prin:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a), \quad \forall\, s \in S,\ a \in A(s).$$

Dacă o acţiune **a** într-o stare **s** produce o întărire a lui R şi o tranziţie la starea **s′**, atunci valoarea Q corespunzătoare se modifică astfel:

$$Q(s,a) \leftarrow (1-\beta)Q(s,a) + \beta(R + \gamma * \max_{a \in A} Q(s', a')),$$

unde $\beta$ este o constantă mică, numită rată de învăţare.

### 7.2.3 Învăţarea nesupervizată

Învăţarea care are loc fără ca agentul să primească vreo o sugestie despre semnalele de ieşire corecte se numeşte ***învăţare nesupervizată***.

În ciuda numeroaselor sale aplicaţii, învăţarea supervizată a fost criticată ca fiind neplauzibilă din punct de vedere biologic. Aceste critici pornesc de la observaţia că este greu de conceput existenţa în creier a unui mecanism de instruire care ar compara ieşirile dorite cu cele reale, propagând înapoi în toată reţeaua de neuroni corecţiile efectuate. Dacă acesta ar fi mecanismul de instruire se pune problema provenienţei formelor de ieşire dorite.

Pe de altă parte s-a pus în evidenţă o autoorganizare a creierului încă din stadiile timpurii ale dezvoltării sale. Această autoorganizare nu poate fi explicată satisfăcător ca rezultatul unui mecanism de învăţare supervizată. Învăţarea nesupervizată este un model de învăţare mult mai plauzibil pentru sistemele biologice. În general, un model de învăţare nesupervizată este unul în care ajustarea ponderilor nu se bazează pe compararea cu răspunsuri ideale predeterminate. Mulţimea de instruire constă doar în vectorii de intrare.

În general algoritmii de clasificare nesupervizată modifică ponderile reţelei pentru a produce vectori de ieşire care sunt consistenţi. Prin aceasta înţelegem faptul că aplicarea a doi vectori de intrare care sunt suficient de asemănători va genera aceeaşi formă de ieşire (sau două forme foarte apropiate). Procesul de învăţare grupează vectorii similari în clase. Prezentând reţelei un vector dintr-o clasă aceasta va genera un vector de ieşire specific. Răspunsul pe care îl va produce o clasă de vectori de intrare nu poate fi însă determinat înainte de încheierea procesului de învăţare. Prin urmare ieşirile unei astfel de reţele trebuie, în general, să fie interpretate, adică să primească o formă inteligibilă după procesul de învăţare.

### 7.2.3.1 Clusterizare

**Clusterizarea**, cunoscută şi sub numele de *clasificare nesupervizată* este procesul împărţirii unui grup (set) de date în grupuri de obiecte similare în raport cu o mulţime de atribute relevante ale obiectelor analizate. Aceste grupuri vor fi numite *clusteri*.

Clusterizarea, din perspectiva modelării datelor, este în strânsă legătură cu matematica, statistica şi analiza numerică. Din perspectiva învăţării automate, clusterii corespund unor *şabloane ascunse*, iar procesul de căutare (identificare) a clusterilor este învăţare nesupervizată.

Dintr-o perspectivă practică procesul de clusterizare joacă un rol deosebit de important în aplicaţii de *Data Mining* (procesul de descoperire de şabloane în date) cum ar fi: explorări de date ştiinţifice, regăsirea informaţiei (*Information Retrieval*), identificarea de şabloane în texte (*Text Mining*), aplicaţii cu baze de date spaţiale, analiză a informaţiilor Web, marketing, diagnostic medical şi multe alte aplicaţii.

În continuare vom prezenta pe scurt problematica generală a clusterizării.

Se consideră un set de date $O=\{O_1,O_2,...,O_n\}$ care constă într-o mulţime de puncte (obiecte, instanţe, modele, tuple, tranzacţii, etc). Fiecare obiect $O_i$ este caracterizat în raport cu o mulţime $A=\{A_1,A_2,...,A_m\}$ de atribute, $O_i=\{O_{i1},O_{i2},...,O_{im}\}$, unde $O_{il}$ este valoarea atribuitului $A_l$ pentru obiectul $O_i$.

Scopul clusterizării este asignarea obiectelor unei mulţimi finite de *k* submulţimi, numite *clusteri*, astfel încât:

$$O=C_1\cup C_2\cup...\cup C_k,\quad C_i\cap C_j=\phi,\forall i\neq j$$

Scopul partiţionării obiectelor este ca obiectele din cadrul unui cluster să fie mai similare între ele decât cu obiectele din alt cluster. Similaritatea şi disimilaritatea între obiecte este calculată folosind *metrici* sau *semi-metrici* aplicate valorilor atributelor care caracterizează obiectele. Spre exemplu, distanţa între două obiecte poate fi exprimată ca fiind *distanţa euclidiană* între acestea (una din metricile des folosite în clusterizare):

$$d(O_i,O_j)=d_E(O_i,O_j)=\sqrt{\sum_{i=1}^{m}(O_{il}-O_{jl})^2}$$

Similaritatea între două obiecte va fi calculată ca inversa distanţei între cele două obiecte:

$$sim(O_i, O_j) = \frac{1}{d(O_i, O_j)}$$

Fără a intra în amănunte legate de algoritmii de clusterizare, amintim principalele metodele de clusterizare existemte:

- metode ierarhice;
- metode de partiţionare (algoritmul *k-means* este unul dintre algoritmii de clusterizare din această grupă);
- clusterizare bazată pe constrângeri;
- algoritmi de clusterizare folosiţi în învăţare automată:
  - reţele neuronale
  - metode evolutive.

## 7.3 Învăţarea în Inteligenţa Artificială Distribuită

### 7.3.1 Introducere

Învăţarea şi inteligenţa sunt strâns legate una de alta. Se spune de obicei că un sistem capabil să înveţe merită să fie numit inteligent, şi reciproc, de la un sistem considerat inteligent se aşteaptă, printre altele, să fie capabil să înveţe. Învăţarea presupune întotdeauna auto-îmbunătăţirea comportamentului viitor pe baza experienţelor precedente. Mai precis, din punct de vedere al inteligenţei artificiale (AI), învăţarea poate fi definită astfel:

*Achiziţionarea de noi cunoştinţe şi aptitudini cognitive şi încorporarea acestora în activităţile viitoare ale sistemului cu condiţia ca achiziţionarea şi încorporarea să fie făcute de către sistem şi să ducă la o îmbunătăţire a performanţelor acestuia.*

Învăţarea în sistemele multiagent reprezintă un subdomeniu important atât al inteligenţei artificiale distribuite (IAD), cât şi al maşinilor auto-instruibile (machine learning - ML). Maşinile auto-instruibile reprezintă o direcţie distinctă de cercetare în IA şi se ocupă de aspectele computaţionale ale învăţării atât în sistemele naturale cât şi în sistemele tehnice.

Intersecţia dintre IAD şi ML o constituie o tânără, dar importantă arie de cercetare şi aplicaţii. Comunităţile IAD şi ML au ignorat în mare măsură această arie o perioadă îndelungată (există excepţii de ambele părţi care doar confirmă regula). Pe de o parte, munca în IAD era în principal preocupată de sistemele multiagent a căror organizare structurală şi comportamental funcţională era determinată în detaliu şi de aceea era mai mult sau mai puţin fixată. Pe de altă parte, munca în ML trata la început învăţarea ca un proces centralizat şi izolat care are loc în sistemele inteligente stand-alone. Această ignorare reciprocă a IAD şi ML a dispărut, şi astăzi aria învăţării în sistemele multiagent se bucură de o largă atenţie într-o constantă creştere. Există două motive principale care justifică această atenţie, ambele ilustrând importanţa unificării IAD şi ML:

- există o mare nevoie să echipăm sistemele multiagent cu abilităţi de învăţare;

- o privire de ansamblu asupra ML care cuprinde nu doar învăţarea monoagent, ci şi cea multiagent poate duce la o înţelegere îmbunătăţită a principiilor generale, fundamentale ale învăţării atât în sistemele naturale cât şi în cele computaţionale.

Primul motiv se bazează pe faptul că sistemele multiagent trebuie să reacţioneze în medii complexe, mari, deschise, dinamice şi imprevizibile. Pentru astfel de medii este deosebit de dificil şi uneori chiar imposibil să se specifice sistemelor corect şi complet o prioritate la momentul creerii lor şi înainte de a fi folosite. Pentru asta este nevoie, de exemplu, să se cunoască prioritatea pentru condiţiile ambientale care se vor ivi, agenţii care vor fi disponibili în acel moment şi cum vor reacţiona şi interacţiona agenţii disponibili ca răspuns la aceste condiţii. Singurul mod posibil de a face faţă acestei dificultăţi este înzestrarea agenţilor individuali cu abilitatea de a-şi îmbunătăţi propriile performanţe şi pe cele ale sistemului în general. Al doilea motiv reflectă faptul că învăţarea în sistemele multiagent nu este doar o sporire a învăţării în sistemele stand-alone şi nici doar suma activităţilor de învăţare izolate a câtorva agenţi. Învăţarea în sistemele multiagent cuprinde învăţarea în sisteme stand-alone pentru că un agent poate învăţa individual şi complet independent de alţi agenţi. Mai mult, învăţarea în sistemele multiagent extinde învăţarea în sisteme stand-alone. Asta pentru că activităţile de învăţare ale unui agent individual pot fi influenţate considerabil (întârziate, accelerate, redirectate sau făcute posibile) de alţi agenţi şi pentru că mai mulţi agenţi pot lucra într-un mod distributiv şi interactiv ca un întreg coerent. Această viziune extinsă a învăţării este diferită calitativ de viziunea tradiţională asupra ML şi are capacitatea de a provoca impulsuri în cadrul cercetării care să conducă la noi tehnici şi algoritmi în domeniul maşinilor auto-instruibile.

Învăţarea în sistemele multiagent este un fenomen cu mai multe laturi şi de aceea nu ne surprinde faptul că mulţi termeni ce se găsesc în literatură se referă la acest tip de învăţare accentuând diferite laturi. Exemple de astfel de termeni sunt: învăţarea reciprocă, învăţarea prin cooperare, învăţarea prin colaborare, învăţarea în echipă, învăţarea socială, învăţarea împărţită, învăţarea pluralistă, învăţarea organizatorică. Scopul acestei secţiuni este de a explicita mai multe astfel de laturi, oferind o caracterizare generală a învăţării în sistemele multiagent. Aceasta se face prin descrierea, din punct de vedere al sistemelor multiagent, a categoriilor principale de învăţare şi a trăsăturilor de bază prin care abordările în învăţare pot diferi.

### 7.3.2 Învăţarea Cooperativă în SMA

În ultimii ani a fost manifestat un interes deosebit pentru abordările descentralizate în rezolvarea problemelor complexe din lumea reală.

Învăţarea în sistemele multiagent este aplicarea învăţării automate în probleme care implică mai mulţi agenţi. Sunt două aspecte de bază ale învăţării în sisteme multiagent, şi anume:

- datorită faptului că învăţarea în SMA abordează domenii de probleme ce implică mai mulţi agenţi, spaţiul de căutare implicat poate fi deosebit de mare;
- învăţarea în SMA *poate* implica *mai multe entităţi care învaţă,* fiecare dintre acestea învăţând şi adaptându-se în contextul celorlalţi.

Anumite abordări din literatura de specialitate se referă la o singură entitate care învaţă pentru a îmbunătăţi performanţa întregii echipe de agenţi. Această categorie de învăţare se numeşte ***învăţare în echipă*** deoarece procesul de învăţare ajustează comportamentul echipei privită ca un întreg.

Alte abordări existente în literatură se referă la procese individuale de învăţare pentru agenţii individuali, urmărind în acelaşi timp şi calitatea echipei de agenţi. Această abordare se numeşte ***învăţare concurentă***.

Ca urmare, se disting două categorii principale de învăţare în sistemele multiagent:

- învăţarea centralizată (izolată);
- învăţarea descentralizată (interactivă).

Pentru a clarifica ce tipuri de învăţare sunt cuprinse în cele două categorii, introducem noţiunea de proces de învăţare:

*Procesul de învăţare se referă la toate activităţile (ex.: planificare, deducţie, paşi de decizie) care sunt executate cu intenţia de a atinge un anume scop.*

Învăţarea este centralizată dacă procesul de învăţare este executat sub toate aspectele lui de un singur agent şi nu are nevoie de interacţiunea cu alţi agenţi; cel care învaţă se comportă ca şi cum ar fi singur. Învăţarea este descentralizată dacă mai mulţi agenţi sunt implicaţi în acelaşi proces de învăţare. Asta înseamnă că activităţile ce constituie procesul de învăţare sunt executate de agenţi diferiţi. În contrast cu învăţarea centralizată, învăţarea descentralizată se bazează sau chiar are nevoie de prezenţa mai multor agenţi capabili să îndeplinească anumite activităţi.

În sistemele multiagent mai mulţi indivizi care învaţă şi încearcă să atingă diferite sau chiar aceleaşi scopuri pot fi activi în acelaşi timp. Similar, pot fi mai multe grupuri de agenţi implicaţi în diferite procese de învăţare descentralizată. Mai mult, scopurile urmărite de aceşti agenţi pot fi diferite sau aceleaşi. Este important să vedem dacă un singur agent poate fi implicat în câteva procese de învăţare centralizate şi/sau descentralizate în acelaşi timp.

### 7.3.2.1 Învăţarea în echipă

În *învăţarea în echipă* este implicată o singură entitate care învaţă: dar această entitate descoperă o mulţime de comportamente mai degrabă pentru o echipă de agenţi, decât pentru un singur agent.

Învăţarea în echipă este o abordare simplă a învăţării în SMA, deoarece se pot folosi technici standard de învăţare (pentru un singur agent): există o singură entitate care efectuează procesul de învăţare.

Un alt avantaj al a unei singure entităţi care învaţă este faptul că agenţii tind să acţioneze mai degrabă în scopul maximizării recompensei echipei, decât a recompenselor lor individuale. Acest lucru face ca agenţii să se comporte mai degrabă egoist decât lacom.

Învăţarea în echipă are, însă, şi anumite dezavanjate: O problemă majoră este spaţiul stărilor care creşte foarte mult în timpul procesului de învăţare. Spre exemplu, dacă agentul ***A*** poate fi într-una din 100 de stări şi agentul ***B*** poate fi într-un a din alte 100 de stări, echipa formată din cei doi agenţi poate fi în 10000 de stări posbile. Această explozie a spaţiului de stări poate fi copleşitoare pentru metode de învăţare care

explorează spaţiul utilităţilor stărilor (cum ar fi învăţarea prin întărire), dar poate să nu afecteze drastic tehnici care exporează spaţiul comportamentelor (cum ar fi programarea evolutivă).

Un al doilea dezavantaj major al învăţării în echipă este centralizarea algoritmului de învăţare: toate resursele trebuie să fie disponibile într-un singur loc în care toate caculele sunt efectuate. Acest lucru poate fi un dezanvantaj în domenii în care datele sunt distribuite.

Învăţarea în echipă poate fi împărţită în două categorii: învăţare *omogenă* şi *eterogenă*. Entităţile omogene care învaţă dezvoltă un singur comportament care este folosit de fiecare membru din echipă. Entităţile eterogene pot dezvolta comportamente unice pentru fiecare agent. Există, deasemenea, o abordare hibridă între cele două.

#### 7.3.2.2 Învăţarea concurentă

Cea mai cunoscută alternativă la învăţarea în echipă în cadrul sistemelor cooperative multiagent este învăţarea concurentă*, în care fiecare agent* din echipă învaţă independent cum să îşi îmbunătăţească performanţele şi performanţa echipei ca un întreg.

Cel mai mare avantaj al învăţării concurente este faptul că împarte spaţiul de căutare reunit al echipei în spaţii individuale de căutare, mai mici. Dacă problema poate fi descompusă astfel încât comportamentele individuale ale agenţilor sunt relativ disjuncte, atunci acest lucru poate duce la o reducere substanţială a spaţiului de căutare şi a complexităţii computaţionale.

### 7.3.3 Trăsături diferenţiale

Cele două categorii descrise anterior sunt generale şi acoperă o varietate de forme de învăţare care pot apărea în sistemele multiagent. Pentru a structura această varietate vom descrie câteva astfel de trăsături. Ultimele două trăsături, binecunoscute în ML, sunt la fel de potrivite pentru caracterizarea abordărilor centralizate şi descentralizate. Celelalte sunt utile caracterizării învăţării descentralizate.

(1) **Gradul de descentralizare**

Descentralizarea unui proces de învăţare îi afectează:

- distribuţia;
- paralelismul.

Extrema este atunci când un singur agent îndeplineşte toate activităţile de învăţare secvenţial. Cealaltă extremă este când activităţile de învăţare sunt distribuite şi paralelizate la toţi agenţii din sistemul multiagent.

(2) **Trăsături specifice interacţiunii**

Există mai multe trăsături care pot fi folosite la clasificarea interacţiunilor necesare pentru a realiza un proces de învăţare descentralizat. Iată câteva exemple:

- nivelul de interacţiune (variază de la observarea trecerii unui semnal la schimbul sofisticat de informaţii prin dialog şi negociere);
- persistenţa interacţiunii (variază de la termen scurt la termen lung);
- frecvenţa interacţiunii (de la joasă la înaltă);
- tiparul interacţiunii (de la complet nestructurat la strict ierarhic);

- variabilitatea interacţiunii (de la fixă la schimbătoare).

Există situaţii în care învăţarea are nevoie de o "interacţiune minimă" (ex.: observarea unui alt agent pentru o perioadă scurtă), în timp ce alte situaţii necesită "interacţiune maximă" (ex.: negocierea repetată pentru o perioadă îndelungată)

(3) **Trăsături specifice implicării**

Exemple de trăsături care pot fi folosite pentru a caracteriza implicarea unui agent în procesul de învăţare:

- relevanţa implicării;
- rolul jucat în timpul implicării.

În funcţie de relevanţă pot fi distinse două extreme: implicarea unui agent nu este o condiţie pentru atingerea scopului deoarece activităţile sale pot fi executate şi de un alt agent; şi contrariul, scopul nu poate fi atins fără implicarea acestui agent. În funcţie de rolul pe care un agent îl joacă în învăţare, acesta se poate comporta ca un „generalist" dacă îndeplineşte toate activităţile (în cazul învăţării centralizate) sau ca un „specialist" dacă se specializează pe o anumită activitate (în cazul învăţării descentralizate).

(4) **Trăsături specifice scopului**

Două exemple de trăsături ce caracterizează învăţarea în sistemele multi agent, în funcţie de scop sunt:

- tipul de îmbunătăţire care se încearcă a fi obţinut prin învăţare;
- compatibilitatea scopurilor urmărite de agenţi.

Prima trăsătură conduce la deosebirea importantă dintre învăţarea care urmăreşte îmbunătăţirea faţă de un singur agent (ex.: aptitudinile motorii sau abilităţile de deducţie) şi învăţarea care urmăreşte îmbunătăţirea faţă de mai mulţi agenţi lucrând în grup (de exemplu abilităţile de comunicare şi negociere şi gradul de coordonare şi coerenţă). A doua trăsătură conduce la deosebirea importantă între scopurile conflictuale şi complementare.

(5) **Metoda de învăţare**

Se deosebesc următoarele metode sau strategii folosite de un agent:

- învăţarea "pe de rost" (ex.: implantarea directă de cunoştinţe şi aptitudini fără a necesita deducţie sau transformare din partea celui care învaţă);
- învăţarea din instrucţiuni şi ascultarea sfaturilor (ex.: operaţionalizarea – transformarea într-o reprezentare internă şi integrarea de cunoştinţe şi aptitudini anterioare – a noii informaţii ca o instrucţiune sau sfat care nu este direct executabil de către cel care învaţă);
- învăţarea din exemple şi antrenament (ex.: extragerea şi rafinarea cunoştinţelor şi aptitudinilor ca un concept general sau un tipar standardizat de mişcare de la exemple pozitive şi negative sau din experienţă practică);
- învăţarea prin analogie (ex.: păstrarea soluţiei de la transformarea cunoştinţelor şi aptitudinilor printr-o problemă rezolvată la o problemă nerezolvată, dar similară);
- învăţarea prin descoperire (ex.: strângerea de cunoştinţe şi aptitudini noi făcând observaţii, experimentând, generând şi testând ipoteze sau teorii pe baza rezultatelor experimentale).

(6) **Feedback-ul învăţării**

Feedback-ul învăţării indică nivelul de performanţă obţinut. Această trăsătură duce la următoarele deosebiri:

- învăţarea supervizată (feedback-ul specifică activitatea dorită de cel care învaţă şi obiectivul învăţării este să potrivească această acţiune cât mai bine);
- învăţarea prin întărire (feedback-ul specifică doar utilitatea activităţii celui care învaţă şi obiectivul este să maximizeze această utilitate);
- învăţarea nesupervizată (nu se oferă nici un feedback şi obiectivul este să se găsească activităţi folositoare şi dorite pe baza proceselor încercare – eroare şi auto-organizare).

În toate aceste cazuri feedback-ul se presupune a fi oferit de mediul sistemului sau chiar de agenţi. Asta înseamnă că mediul sau agentul care oferă feedback se comportă ca un "învăţător" în cazul învăţării supervizate, ca un "critic" în cazul învăţării prin întărire şi "pasiv" în cazul învăţării nesupervizate.

Aceste trăsături caracterizează învăţarea în sistemele multiagent din diferite puncte de vedere şi pe diferite nivele. În particular, ele au un impact semnificativ asupra trebuinţelor şi abilităţilor agenţilor implicaţi în învăţare. Sunt posibile diferite combinaţii cu diferite valori pentru aceste trăsături.

### 7.3.4 Problema Credibilităţii (The Credit Assignment Problem)

Problema principală cu care se confruntă fiecare sistem care învaţă este problema credibilităţii (**Credit Assignment Problem - CAP**). Acesta este problema asignării exacte a feedback-ului de crezare sau învinuire, pentru o schimbare globală a performanţei (creştere sau descreştere), a fiecărei activităţi a sistemului care a contribuit la această schimbare.

Această problemă a fost, în mod tradiţional, considerată în contextul sistemelor de sine stătătoare, dar există, deasemenea, şi în contexul sistemelor multiagent. Considerând abordarea din Inteligenţa Artificială tradiţională, conform căreia activităţile unui sistem inteligent sunt date de acţiuni externe efectuate de acesta şi de inferenţele interne şi deciziile care implică aceste acţiuni, problema credibilităţii pentru sistemele multiagent poate fi descompusă în două probleme:

- problema *CAP inter-agenţi* - asignarea creditului sau învinuirii pentru o schimbare globală a performanţei datorate acţiunilor agenţilor;
- problema *CAP intra-agenţi* - asignarea creditului sau învinuirii pentru o acţiune particulară a unui agent relativ la inferenţele şi deciziile sale interne.

Problema credibilităţii *inter-agenţi* este o problemă dificilă în sistemele multiagent, deoarece o schimbare a performanţei globale poate fi cauzată de acţiuni externe sau diferiţi agenţi distribuiţi spaţial şi/sau logic.

Descrierea anterioară a problemei *CAP* este conceptuală, scopul fiind de a evidenţia deosebirea clară între subproblemele inter şi intra-agent. În practică, deosebirea nu este totdeauna evidentă. De altfel, abordările disponibile pentru învăţare în sistemele multiagent nu fac diferenţa, în general, în mod explicit, înte cele două subprobleme, sau se orientează doar pe una dintre subprobleme, simplificând-o mult pe cea de-a doua. Oricum, este util să fim conştienţi de ambele subprobleme atunci când abordăm o problemă de învăţare într-un sistem multiagent.

### 7.3.5 Învăţarea şi coordonarea activităţii

Această secţiune explică problema în care mai mulţi agenţi pot învăţa să-şi coordoneze activităţile (ex: pentru a-şi împărţi optim resursele sau pentru a maximiza profitul). Coordonarea activităţii se preocupă de dezvoltarea şi adaptarea fluxului de date şi de tiparele de control care îmbunătăţesc interacţiunea dintre agenţi. Anumiţi agenţi sunt nevoiţi să interacţioneze cu alţi agenţi care au diferite scopuri, abilităţi, structuri şi durată de viaţă. Pentru a utiliza eficient oportunităţile prezentate şi pentru a evita capcanele, agenţii trebuie să înveţe despre alţi agenţi şi să-şi adapteze comportamentul în funcţie de structura grupului şi de dinamică.

#### 7.3.5.1 Învăţarea izolată, concurentă şi consolidată

Tehnicile de învăţare prin întărire pot fi folosite de agenţi pentru a dezvolta politici de selectare a acţiunii optimizând astfel feedback-ul mediului prin formarea unei corespondenţe între percepţie şi acţiune. Un avantaj al acestor tehnici este că pot fi folosite în domenii în care agenţii au foarte puţină experienţă sau chiar deloc, şi au informaţii puţine despre aptitudinile şi ţelurile altor agenţi. Lipsa acestor informaţii îngreunează problema coordonării. Aproape toate mecanismele de coordonare se bazează pe cunoştinţele în domeniu şi colaborarea dintre agenţi. Ideea este că învăţarea prin întărire se poate folosi pe post de tehnică nouă de coordonare în domenii în care tehnicile cunoscute sunt ineficiente.

Se pune întrebarea: să aleagă agenţii să nu comunice în timp ce învaţă sa se coordoneze? Cu toate că activitatea de comunicare este indispensabilă în activităţile de grup, nu garantează un comportament coordonat, consumă timp, şi poate distrage de la alte activităţi dacă nu este controlată cu atenţie. De asemenea, agenţii care depind de comunicare vor fi grav afectaţi dacă calitatea acesteia este compromisă (canale de comunicaţie întrerupte, informaţii eronate etc.). Câteodată, comunicarea poate fi riscantă sau chiar fatală (ca şi în cazul luptelor când inamicul interceptează mesajele). Chiar şi atunci când comunicarea se desfăşoară normal, este bine să se facă doar dacă este absolut necesar. Acest design produce sisteme care nu inundă canalele de comunicare cu informaţie negarantată. Ca rezultat, agenţii nu trebuie să se rătăcească printr-un labirint de informaţie inutilă pentru a localiza informaţia necesară.

În această formă de învăţare izolată şi concurentă, fiecare agent învaţă să-şi îmbunătăţească recompensa din mediu. Alţi agenţi din mediu nu sunt modelaţi explicit. Astfel, o altă întrebare interesantă ar fi dacă merită ca un astfel de agent să folosească acelaşi mecanism de învăţare în ambele medii: cooperativ şi non-cooperativ.

O prezumţie de bază a majorităţii tehnicilor de învăţare prin întărire este că dinamica mediului nu este afectată de alţi agenţi. Această prezumţie este eronată în domenii cu mai mulţi agenţi care învaţă. Ne punem problema dacă tehnicile de învăţare prin întărire vor fi adecvate pentru învăţarea prin coordonare concurentă, izolată. Următoarele dimensiuni caracterizează învăţarea prin întărire, concurentă şi izolată (CIRL):

**Cuplarea agenţilor.** În unele domenii acţiunile unui agent afectează puternic planurile altor agenţi (sisteme cuplate strâns), în timp ce în alte domenii acţiunile unui agent afectează mai puţin planurile altor agenţi (sisteme cuplate slab).

**Relaţiile agenţilor.** Agenţii dintr-un sistem multiagent pot avea diferite relaţii:

- Acţionează într-un grup pentru a rezolva o problemă comună (agenţi cooperativi).
- Nu au o dispoziţie unul faţă de celălalt, dar interacţionează deoarece folosesc aceleaşi resurse (agenţi independenţi).
- Au interese opuse (agenţi adversari).

**Sincronizarea feedback-ului.** În unele domenii agenţii pot cunoaşte efectele acţiunii lor imediat, în timp ce în altele pot primii feedback-ul acţiunii lor după o întârziere.

**Combinaţii comportamentale optime.** Câte combinaţii comportamentale ale agenţilor participanţi vor rezolva problema în mod optim? Numărul variază de la unu la infinit pentru diferite domenii.

Pentru a evalua aceste întrebări a fost folosită tehnica Q-Learning în trei domenii:

**Împingerea cuburilor.** Doi agenţi individuali învaţă să împingă o cutie dintr-un loc iniţial într-un loc ţintă de-a lungul unei traiectorii date. Ambele situaţii, cooperativă (au aceeaşi ţintă) şi competitivă (au ţinte diferite) au fost studiate. Feedback-ul se bazează pe devierea locului cutiei de la traiectoria stabilită. Caracteristicile domeniului sunt: învăţarea concurentă de către doi agenţi cu feedback de mediu apropiat; sisteme cuplate strâns; comportamente optime multiple.

**Împărţirea resurselor.** Fiindu-le date sarcini individuale, doi agenţi trebuie să înveţe să împartă o resursă pentru o anumită perioadă. Caracteristicile domeniului sunt: feedback-ul mediului întârziat, sisteme cuplate strâns; comportament optim unic.

**Navigarea roboţilor** Doi roboti învaţă să navigheze intersectându-şi traiectoriile fără să se lovească. Caracteristicile domeniului sunt: feedback-ul mediului imediat, cuplare variabilă, comportamente optime multiple.

Concluzia a fost că CIRL conferă o nouă paradigmă sistemelor multiagent prin care atât prietenii cât şi duşmanii pot obţine o cunoaştere a coordonării foarte utilă. Nici cunoştinţele vechi despre domeniu şi nici un model explicit despre aptitudinile altor agenţi nu este necesar. Limitarea constă în incapacitatea CIRL de a dezvolta o coordonare eficientă când acţiunile agenţilor sunt strâns legate, feedback-ul este întârziat şi nu sunt decât una sau puţine combinaţii optime. O posibilă soluţie parţială la această problemă ar fi folosirea unei forme de învăţare alternativă. În acest caz, fiecare agent poate să înveţe o perioadă de timp, apoi să execute politica lui fără modificări, apoi să înveţe din nou etc. Doi agenţi pot să-şi sincronizeze comportamentul astfel încât unul să înveţe în timp ce altul acţionează şi invers. Chiar dacă sincronizarea perfectă nu este posibilă, învăţarea alternativă este mai eficientă decât cea concurentă.

Alte observaţii interesante:

- In situaţii cooperative, agenţii pot învăţa politici complementare de a rezolva o problemă. Asta duce la specializare şi nu la dezvoltarea comportamentului identic.
- Agenţii pot transfera învăţarea în situaţii similare (ex.: o dată ce învaţă să se coordoneze la o problemă, învaţă sa se coordoneze mai repede la alta similară).

### 7.3.5.2 Învăţând despre şi de la alţi agenţi

În această secţiune sunt considerate scenarii în care agenţii învaţă să-şi îmbunătăţească performanţele. Uneori această îmbunătăţire a performanţei ori creştere a recompensei mediului vine datorită altor agenţi din acel mediu. Aici se pune accentul pe agenţi care învaţă despre alţi agenţi pentru a profita de oportunităţi şi pe întrebarea despre cum poate fi influenţată învăţarea unui agent de către alţi agenţi. Acest subcapitol se concentrează asupra prezumţiei comportamentului altor agenţi (incluzând preferinţele lor, strategii, intenţii etc.), asupra îmbunătăţirii comportamentului unui agent interacţionând cu alţi agenţi, asupra dezvoltării unui punct de vedere comun asupra lumii.

**Roluri organizaţionale în învăţare**

Agenţii din grupuri trebuie să înveţe să-şi distribuie diverse roluri pentru a se completa reciproc. Adaptând structura grupului şi activităţile individuale ale membrilor la o manieră dependentă de situaţie, se permite unui grup să sporească performanţele sistemului şi să întâmpine dificultăţile neprevăzute. Nagendra Prasad, Lesser şi Lander prezintă un formalism care combină negocierea bazată pe memorie şi învăţarea prin întărire pentru a permite membrilor grupului alegerea rolurilor organizaţionale. Ei adresează problema agenţilor care învaţă să adopte roluri specifice într-un domeniu al rezolvării problemei în mod cooperativ. Fiecare agent se presupune capabil de a juca unul din rolurile situaţiei. Ţelul este ca un agent să fie capabil să aleagă cel mai potrivit rol.

**Învăţarea de a profita de condiţiile de piaţă**

Agenţii informaţionali care vând şi cumpără informaţii într-o piaţă electronică trebuie să se poată adapta condiţiilor mediului. Vidal şi Durfee cercetează avantajele agenţilor care învaţă după modelele altor agenţi. Ei cercetează folosirea agenţilor pentru a vinde şi a cumpăra informaţii pe pieţe electronice, precum bibliotecile digitale. Se presupune că astfel de medii sunt deschise şi noi agenţi (fie cumpărători, fie vânzători de informaţii) pot intra sau părăsi piaţa după voie. O abordare practică a acestui sistem ar fi să considerăm că fiecare agent este o entitate "egoistă" cu scopul de a maximiza utilitatea locală. Un mecanism economic este folosit pentru a controla transferul informaţiilor între agenţii care furnizează informaţiile şi agenţii care au nevoie de ele. Calitatea informaţiei disponibilă la diferiţi vânzători poate varia şi aprecierea şi cumpărarea sunt lăsate în seama vânzătorilor şi cumpărătorilor.

Se presupune că informaţia poate fi reprodusă arbitrar la costuri neglijabile şi agenţii au acces uniform la toţi agenţii din piaţă. Într-un asemenea scenariu, un vânzător trebuie să ofere servicii cu valoare adăugată pentru a-şi diferenţia produsele de alţi vânzători. Într-o astfel de piaţă, un cumpărător anunţă bunul de care are nevoie. Vânzătorii etalează preţuri pentru oferirea acelor bunuri. Cumpărătorul alege între aceste oferte şi oferă preţul cuvenit. Vânzătorul oferă produsul cumpărătorului. Cumpărătorul poate evalua calitatea bunului numai după ce îl primeşte (nu poate înainte de a-l cumpăra). Profitul vânzătorului $s$ ce vinde un produs $g$ la un preţ $p$ este $p - c_s^g$, unde $c_s^g$ este costul producerii acelui bun. Dacă acest bun este de calitatea $q$, valoarea lui pentru

un cumpărător $b$ este $V_b^g(p,q)$. Într-o tranzacţie, scopul cumpărătorului şi al vânzătorului este de a maximiza valoarea şi profitul.

Într-o astfel de piaţă sunt cercetate trei tipuri de agenţi:

- Agenţi nivel-0: Aceşti agenţi nu modelează comportamentul altor agenţi. Ei fixează preţurile de vânzare şi cumpărare bazându-se pe experienţă.
- Agenţi nivel-1: Aceştia analizează comportamentul trecut al altor agenţi şi încearcă să prezică preţurile de vânzare-cumpărare ale acestora. Alţi agenţi sunt modelaţi ca agenţi nivel-0 sau agenţi care nu modelează alţi agenţi. Asta înseamnă ca dacă un agent nivel-1 A modelează un agent nivel-0 B, A nu consideră faptul că şi B îl modelează pe A. Observăm faptul că agenţii nivel-1 au informaţii despre agenţi individuali din mediu, în timp ce agenţii nivel-0 folosesc doar experienţa trecută.
- Agenţii nivel-2: Aceştia modelează alţi agenţi ca şi agenţi nivel-1. Acest lucru înseamnă că aceşti agenţi văd alţi agenţi ca agenţi care îi modelează pe alţii ca şi agenţi nivel-0 sau agenţi care nu au modelele altora.

## Învăţarea de a juca mai bine împotriva unui oponent

În domenii ca jocurile de masă, strategia clasică oferă o abordare conservatoare. Dacă strategia folosită de oponent se poate aproxima, atunci exploatarea slăbiciunilor strategiei poate fi folosită pentru a obţine rezultate mai bune împotriva acelui oponent. Într-o abordare similară în dezvoltarea jucătorilor care să exploateze punctele slabe ale adversarului, Sean şi Arora au folosit principiul Utilităţii Maxime Aşteptate (UMA) pentru exploatarea învăţării modelelor adversarilor. În abordarea lor, sunt folosite probabilităţi condiţionale pentru diferite mutări ale adversarului, corespunzătoare tuturor mutărilor din starea respectivă, pentru a calcula utilităţi aşteptate pentru fiecare mutare posibilă. Este jucată mutarea cu utilitatea maximă. Astfel se dezvoltă un model probabilistic al strategiei adversarului, observând mutări ale adversarului în diferite intervale de discrepanţă, măsurate de funcţia de evaluare a jucătorului.

Să presupunem că jucătorul şi adversarul aleg din mulţimile următoare de mutări, $\alpha = \{\alpha_1, \alpha_2, \ldots\}$, respectiv $\beta = \{\beta_1, \beta_2, \ldots\}$, iar utilitatea primită de A pentru perechea de mişcări $(\alpha_i, \beta_i)$ este $u(\alpha_i, \beta_i)$. Principiul UMA poate fi folosit pentru a alege mutări astfel: $\arg\max_{\alpha_i \in \alpha} \sum_{\beta_j \in \beta} p(\beta_j|\alpha_i) u(\alpha_i, \beta_j)$, unde $p(\beta_j|\alpha_i)$ este probabilitatea condiţionată ca adversarul să aleagă mutarea $\beta_j$ ştiind că agentul alege mutarea $\alpha_i$. Dacă strategia adversarului poate fi modelată perfect de către mecanismul de învăţare, jucătorul UMA va putea exploata slăbiciunile adversarului.

Toate domeniile discutate implică învăţarea izolată în sens distribuit. Unul sau mai mulţi agenţi pot învăţa concurent în acel mediu. Agenţii interacţionează des şi informaţia obţinută este folosită pentru a crea modele pentru alţi agenţi. Deoarece fiecare agent învaţă separat, toţi agenţii trebuie să execute toate activităţile de învăţare. Majoritatea mecanismelor de învăţare sunt variante ale învăţării prin întărire.

### 7.3.5.3 Învăţarea şi comunicarea

În acest subcapitol, accentul cade pe legătura dintre învăţare şi comunicare. Această legătură priveşte cerinţele asupra abilităţii agenţilor de a interschimba eficient informaţia folositoare. Se identifică două mari direcţii de cercetare:

- Învăţarea de a comunica: învăţarea este privită ca o metodă pentru a reduce sarcina comunicării între agenţi.
- Comunicarea privită ca învăţare: comunicarea este văzută ca o metodă de a schimba informaţii care permit agenţilor să continue sau să-şi perfecţioneze activităţile de învăţare.

Prima direcţie de cercetare porneşte de la faptul că activitatea de comunicare este de obicei înceată şi scumpă şi de aceea ar trebui evitată sau redusă ori de câte ori este posibil. Ultima direcţie se bazează pe premiza că învăţarea (cum ar fi planificarea şi luarea deciziilor) este limitată din start de informaţia disponibilă şi pe care o poate procesa un agent. Ambele direcţii doresc îmbunătăţirea comunicării şi a învăţării în sistemele multiagent, şi sunt legate de următoarele probleme:

- Ce să comunice (ce informaţii interesează pe ceilalţi).
- Când să comunice (cât efort trebuie să investească un agent în rezolvarea problemei înainte de a cere ajutor).
- Cu cine să comunice (pe care agent îl interesează această informaţie, cărui agent să-i ceară ajutor).
- Cum să comunice (la ce nivel să comunice agenţii, ce limbaj şi protocol să fie folosit, schimbul de informaţii să se facă direct punct cu punct şi difuzat, sau cu un mecanism "tablă".

Aceste probleme trebuie adresate de proiectantul sistemului sau derivate de sistemul însuşi. Următoarele două paragrafe ilustrează cele două direcţii, descriind abordări reprezentative în "învăţarea de a comunica" şi "comunicarea privită ca învăţare".

Mai există un aspect care merită dezvoltat când vorbim de învăţare şi comunicare în sistemele multiagent. O condiţie necesară pentru un schimb util de informaţie este existenţa unei ontologii comune. Desigur, comunicarea nu este posibilă dacă agenţii atribuie semnificaţii diferite aceloraşi simboluri, fără să realizeze diferenţele (sau fără să le poată detecta şi manevra). Dezvoltarea unei înţelegeri comune a simbolurilor poate fi considerată o sarcină esenţială a învăţării în sistemele multiagent .

#### 7.3.5.3.1 Reducerea comunicării prin învăţare

În mediile complexe, în special, este nevoie de mecanisme pentru reducerea comunicaţiilor transmise. Smith propune câteva mecanisme de acest gen, cum ar fi concentrarea pe adresare şi contractarea directă care au ca scop înlocuirea comunicării punct cu punct cu difuzarea. Un dezavantaj al acestor mecanisme este că trebuie ca designerul sistemului să ştie dinainte căile de comunicare directă, şi deci tiparele comunicării rezultate pot fi inflexibile în medii non – statice. În cele ce urmează este descris un mecanism mult mai flexibil numit "învăţarea adresată" (într-o formă simplificată).

Ideea de baza în învăţarea adresată este de a reduce eforturile comunicării pentru anunţarea sarcinilor permiţând agenţilor să acumuleze şi sintetizeze cunoştinţe despre

capacitatea celorlalţi agenţi de a rezolva sarcini. Cu ajutorul cunoştinţelor asimilate, sarcinile pot fi atribuite direct fără a fi necesară difuzarea anunţurilor către toţi agenţii. Negocierea "bazată pe caz" este aplicată ca un mecanism bazat pe experienţă pentru culegerea cunoştinţelor şi sintetizarea lor. Aceasta se bazează pe observaţia că oamenii rezolvă adesea o problemă pe baza unor soluţii care au funcţionat în trecut pentru probleme similare. Ea pune accentul pe construirea cazurilor, adică perechi problemă – soluţie (*case based reasoning*). Când se iveşte o problemă nouă se verifică dacă este complet necunoscută sau similară cu una cunoscută (refolosirea cazului). Dacă este necunoscută, o soluţie trebuie concepută. Dacă sunt asemănări cu o problemă cunoscută, soluţia acestei probleme poate fi folosită ca punct de plecare în rezolvarea celei noi (adaptarea cazului). Toate problemele întâlnite, împreună cu soluţiile lor, sunt memorate ca şi cazuri într-o bază de cazuri. Acest mecanism poate fi aplicat pentru reducerea comunicării într-o reţea contractuală după cum urmează: Fiecare agent îşi păstrează propria bază de cazuri. Un caz se consideră că conţine:

- detalii despre o sarcină;
- informaţii despre care agent a rezolvat această problemă şi cât de bună sau rea a fost soluţia.

Detaliile despre sarcina $T_i$ sunt de forma:

$$T_i = \{A_{i1}V_{i1}, \ldots, A_{im_i}V_{im_i}\},$$

unde $A_{ij}$ este un atribut al lui $T_i$ şi $V_{ij}$ este valoarea acestui atribut. Este nevoie, pentru a aplica negocierea bazată pe caz, de o măsură a similarităţii între sarcini. În cazul învăţării adresate, această măsură este redusă la similaritatea dintre atribute şi valorile lor. Mai precis, pentru fiecare două atribute $A_{ir}$ şi $A_{js}$, distanţa dintre ele este definită astfel:

$$DIST(A_{ir}, A_{js}) = SIMILAR\_ATT(A_{it}, A_{js}) * SIMILAR\_VAL(V_{ir}, V_{js}),$$

unde *SIMILAR_ATT* şi *SIMILAR_VAL* reprezintă similaritatea dintre atribute şi valori. Cum se definesc aceste două măsuri depinde de domeniul de aplicaţie şi de cunoştinţele despre atributele sarcinilor şi valorile lor. În cea mai simplă formă, se definesc astfel:

$$SIMILAR\_ATT(x, y) = SIMILAR\_VAL(x, y) = \begin{cases} 1, x = y \\ 0, altfel \end{cases}$$

ceea ce înseamnă că similaritatea este egală cu identitatea. Cu ajutorul distanţei DIST între atribute, similaritatea dintre două sarcini $T_i$ şi $T_j$ se va clacula astfel:

$$SIMILAR(T_i, T_j) = \sum_r \sum_s DIST(A_{ir}, A_{js})$$

Pentru fiecare sarcină $T_i$ poate fi definită o mulţime de sarcini similare $S(T_i)$ specificând cerinţele de similaritate între sarcini. Iată un astfel de exemplu:

$$S(T_i) = \{T_j : SIMILAR(T_i, T_j) \geq 0.85\},$$

unde sarcinile $T_j$ sunt incluse în baza cazurilor agentului care caută cazuri similare. Să considerăm cazul în care un agent N trebuie să decidă atribuirea unor sarcini $T_i$ altui agent. În loc să difuzeze anunţul $T_i$, N încearcă să aleagă unul sau mai mulţi agenţi pe care îi consideră potriviţi pentru rezolvarea $T_i$ calculând pentru fiecare agent M această potrivire:

$$SUIT(M, T_i) = \frac{1}{|S(T_i)|} \sum_{T_j \in S(T_i)} PERFORM(M, T_j)$$

unde $PERFORM(M, T_j)$ este o măsură bazată pe experienţă, ce indică cât de bun sau cât de rău a fost executată $T_j$ de către M în trecut. Astfel, agentul N trimite anunţul $T_i$ celui mai potrivit agent sau agenţi în loc să trimită tuturor.

### 7.3.5.3.2 Îmbunătăţirea Învăţării prin Comunicare

Deoarece un agent nu poate fi considerat omnipotent, în cele mai multe probleme nu poate fi considerat nici omniscient fără a încălca presupunerile realiste. Lipsa de informaţii pe care o poate avea un agent se poate datora:

- mediului în care este integrat (localizarea obstacolelor) şi problemei pe care trebuie să o rezolve (specificarea stării obiectivului ce trebuie atins);
- altor agenţi (aptitudinile, strategiile şi cunoştinţele lor);
- dependenţei dintre diferite activităţi şi efectele activităţii proprii faţă de ale altor agenţi asupra mediului şi asupra potenţialelor activităţi viitoare (o acţiune *a* întreprinsă de agentul A poate împiedica un agent B să întreprindă acţiunea *b* şi poate permite agentului C să facă acţiunea *c*).

Agenţii care au un acces limitat la informaţii importante, pot eşua în învăţarea unei sarcini date. Acest risc poate fi redus permiţând agenţilor să interschimbe informaţii, deci să comunice între ei. Pot fi delimitate două forme de îmbunătăţire a învăţării prin comunicare:

- învăţarea bazată pe comunicarea de nivel jos, adică, interacţiuni simple întrebare – răspuns în scopul schimbului de informaţii lipsă (cunoştinţe şi ipoteze);
- învăţarea bazată pe comunicarea de nivel înalt, adică, interacţiune mult mai complexă, cum ar fi negocierea şi explicarea în scopul combinării şi sintetizării informaţiei.

În timp ce prima formă de învăţare comunicativă are ca rezultat împărţirea informaţiei, a doua formă are ca rezultat împărţirea înţelegerii. În ambele forme, comunicarea este folosită ca mijloc de îmbunătăţire a învăţării. Lăsând deoparte acest punct de vedere, trebuie reţinut faptul că doar acest tip de comunicare poate fi văzut ca învăţare, pentru că este o realizare specifică a sistemului multiagent de acumulare a cunoştinţelor. Dacă învăţarea ar trebui să fie îmbogăţită prin comunicare rămâne o întrebare dificilă. În lumina evaluării criteriilor standard pentru algoritmii de învăţare – viteză, calitate, complexitate – această întrebare poate fi descompusă în trei întrebări:

- Cât de repede sunt dobândite rezultatele învăţării cu şi fără comunicare?
- Rezultatele învăţării cu/fără comunicare au o calitate suficientă?
- Cât de complex este procesul de învăţare cu sau fără comunicare?

Consideraţiile de mai sus ne arată că activitatea de comunicare oferă numeroase posibilităţi pentru îmbunătăţirea învăţării, dar ea nu reprezintă leacul universal în rezolvarea problemelor de învăţare în sisteme multiagent. Combinarea lor trebuie deci făcută cu grijă. În unele cazuri, se observă că, tocmai comunicarea poate introduce informaţii false sau incomplete în baza de informaţii a unui agent (datorită erorilor de transmisie) care îngreunează rezolvarea unei sarcini de învăţare.

### 7.3.6 Învăţarea despre alţi agenţi în sistemele multiagent dinamice

Un ***sistem multiagent dinamic*** este un sistem care se schimbă în timp, datorită interacţiunilor dintre mai mulţi agenţi, fiind un *sistem distribuit cu inteligenţă artificială*.

Un exemplu de sistem multiagent dinamic ar fi jocul de fotbal robotizat sau sistemele de informare în reţele.

Un sistem multiagent dinamic este o particularizare a unui sistem multiagent care poate fi folosit în rezolvarea problemelor complexe, al căror mediu este dinamic (se modifică în timp) şi în care un rol important îl joacă interacţiunea între mai mulţi agenţi. Din acest punct de vedere, sistemele multiagent dinamice constituie un exemplu de sisteme rezolutive folosite în Inteligenţa Artificială Distribuită (IAD).

Pentru a face parte dintr-un sistem multi-agent dinamic un agent trebuie să fie capabil :

- *să modeleze comportamentul altor agenţi*; un model corect este absolut necesar pentru succesul sau supravieţuirea agentului; de exemplu într-un joc de fotbal, abilitatea agentului de a prevedea mişcarea celorlalţi agenţi, va determina dacă agentul poate sau nu ajunge la minge;
- *să se adpteze pe la schimbările mediului*, adică agentul să-şi poată modifica comportamentul şi funcţiile (procedurile) de planificare a comportamentului în momentul în care primeşte noi informaţii.

Într-un astfel de sistem un aspect important este acela de a *învăţa* despre alţi agenţi prin :

- folosirea unei metode de *învăţare nesupervizate* - din moment ce agenţii sunt autonomi şi nu există nici un supervizor, *învăţarea supervizată* nu este utilă într-un sistem multi-agent dinamic;
- agentul trebuie să-şi formeze un *model* al celorlalţi agenţi şi să folosească metode de învăţare pentru a-şi corecta acest model.

#### 7.3.6.1 Aspecte formale în reprezentarea unui sistem multi-agent dinamic

Un *sistem multiagent dinamic* este format dintr-un număr $n \geq 2$ agenţi care evoluează în timp (într-un interval de timp agenţii îşi propun un scop).

Într-un astfel de sistem agenţii pot acţiona:

- *cooperativ* sau
- *non-coopeerativ*.

Vom considera în continuare cazul non-cooperativ, în care fiecare agent îşi urmăreşte propriile obiective .

Sistemul porneşte la momentul iniţial **0**, deplasându-se în mod discret (individual) spre momentul **T**, sau până când atinge o stare care satisface o condiţie de terminare. Agenţii îşi evaluează rezultatele în baza stării finale.

Din punct de vedere formal, un sistem multi-agent dinamic este reprezentat sub forma $< S, A, h, U >$, unde :

- $S = \prod S^i$ -spaţiul reunit al stărilor; $S^i$-spaţiul de stări perceput de agentul i ;
- $A = \prod A^i$-spaţiul reunit al acţiunilor; $A^i$ -spaţiul acţiunilor agentului i ;

- $h$ este funcţia de tranziţie între stările mediului, h: SxA→S funcţie ce asignează unei perechi (stare, acţiune) o nouă stare;
- $U = (U^1, U^2, ..., U^n)$ este vectorul funcţiilor de utilitate ale agenţilor.

Presupunem că spaţiul stărilor şi al acţiunilor sunt domenii continue.

La fiecare moment *t,* sistemul percepe starea $s_t \in S$, care e constituită din stări locale pentru fiecare agent individual $s_t = (s_t^1, s_t^2, ..., s_t^n)$. Agenţii îşi vor alege acţiunea $a_t^i \in A^i$, producând acţiunea reunită $a_t = (a_t^1, a_t^2, ..., a_t^n)$.

Stările evoluează conform relaţiei $s_{t+1} = h(s_t, a_t)$, unde $s_{t+1}^i = h^i(s_t^i, a_t)$. Altfel spus, starea agentului *i* la momentul de timp *t+1* depinde de starea curentă a agentului şi de acţiunea reunită curentă. Vom presupune că funcţia de tranziţie *h* este deterministă şi staţionară.

După cum s-a arătat, funcţia de utilitate a agentului *i* este $U^i$, o funcţie a stării finale a agentului $s_T^i$. Se va introduce noţiunea de ***variaţie a utilităţii agentului i la momentul t*** ca: $r_t^i = U^i(s_t^i) - U^i(s_{t-1}^i)$, fiind numită şi ***recompensa obţinută de agentul i la momentul t*** .

Se observă că $\sum_{t=1}^{T} r_t^i = \sum_{t=1}^{T} [U^i(s_t^i) - U^i(s_t^i)] = U^i(s_T) - U^i(s_0)$, iar $U^i(s_0)$ este o constantă independentă de acţiunile agentului *i,* deci maximizarea sumei recompenselor primite de agentul *i* este echivalentă cu maximizarea utilităţii corespunzătoare stării finale.

Din cele prezentate mai sus rezultă că scopul unui agent de învăţare într-un sistem multiagent dinamic care evoluează de la momentul iniţial **0** la un moment final **T**, este maximizarea sumei recompenselor pe care le primeşte de la starea iniţială la starea finală.

Chiar dacă maximizăm recompensele pe o singură perioadă de timp,

$$r_{t+1}^i = U^i(h^i(s_t^i, a_t)) - U^i(s_t^i),$$

este nevoie să cunoaştem acţiunile celorlalţi agenţi, $a_t^{-i}$. Scopul în problema învăţării este să prezicem aceste acţiuni – explicit sau implicit – în aşa fel încât agentul să poată alege efectiv acţiunea lui.

### 7.3.6.2 Modelarea altor agenţi

Pentru a simplifica problema învăţării, presupunem că agentul ia deciziile considerând numai perioada curentă de timp. Pentru a maximiza recompensa curentă la momentul t(1), agentul rezolvă :

$$\arg\max_{a_t^i \in A^i} U^i(h^i(s_t^i, a_t^i, a_t^{-i})).$$

O abordare generală este aceea de a face o estimare, $a_t^{-i}$, a acţiunilor celorlalţi agenţi şi de a rezolva problema ca şi cum estimarea ar fi corectă. Decizia se reduce astfel la modul în care se face estimarea.

**Nivele de modelare**

Cea mai directă cale de a estima pe $a_t^{-i}$ se bazează pe o analiză în serii a observaţiilor anterioare, $\{a_\tau^{-i}\}, \tau < t$. Agenţii care folosesc această abordare nu raţionează explicit asupra faptelor ce îl determină pe $a_t^{-i}$. Astfel de agenţi se numesc agenţi nivel-0. Un exemplu de estimare nivel-0 este considerarea lui $a_t^{-i}$ ca o funcţie liniară de observaţii importante,

$$a_t^{-j} = \sum_{k=1}^{t} w_k a_{t-k}^j .$$

Un agent care încearcă să modeleze politica altor agenţi se numeşte agent nivel-1.

Extensia naturală este să considerăm agenţi care modelează alţi agenţi ca agenţi nivel-1. Aceşti agenţi se numesc agenţi nivel-2. Ei modelează politica agentului j ca

Se pot defini agenţi nivel-3, …, agenţi nivel-N repetând extinderea acestui model.

**Agenţi non-estimatori**

Un agent nivel-0 poate învăţa o politică, fără a considera explicit valoarea $a_t^{-i}$, caz în care politica sa, $f^i(s_t^i)$, este o funcţie ce depinde doar de starea locală. Un exemplu de astfel de astfel de agent este agentul competitiv. A fi competitiv într-un context de piaţă înseamnă a presupune că efectul celorlalţi asupra mediului este neglijabil, şi deci specularea acţiunilor celorlalţi nu aduce nici un avantaj. Acest tip de agent tinde să se comporte reactiv, cu toate că această caracterizare nu e la fel de precis definită ca şi competiţia în contextul de piaţă.

Ca şi exemplu, fie un agent nivel-1 care modelează agenţi nivel-0 non-estimatori. Pentru a estima politica agentului j se aplică o regresie liniară logică asupra datelor istorice disponibile, $\{(s_\tau^j, a_\tau^j) | \tau < t\}$. Dându-se starea curentă $s_t^j$, se iau cei mai apropiaţi k vecini (definiţi prin distanţa euclidiană), $\{s_{\tau 1}^j, ..., s_{\tau k}^j\}$ şi se aplică regresia liniară pe punctele $\{(s_{\tau 1}^j, a_{\tau 1}^j), ..., (s_{\tau k}^j, a_{\tau k}^j)\}$. Rezultă o estimare a parametrilor $\alpha$ şi $\beta$ în modelul

$$a^j = \alpha + \beta s^j \quad .$$

**7.3.7 Domenii de aplicare ale Învăţării în Sistemele Multiagent**

În ciuda faptului că este un domeniu destul de nou, domeniul Sistemelor Multiagent conţine un număr foarte mare de domenii de probleme. În cele ce urmează vom prezenta câteva dintre problemele rezolvate folosind sisteme multiagent, majoritatea abordărilor de rezolvare folosind tehnici de învăţare pentru a asigura autonomia agenţilor.

### 7.3.7.1 Agenţi robotici

În ultima vreme, cercetările în domeniul roboticii multiagent cooperative au cunoscut o mare amploare. Operaţiile care nu pot fi efectuate cu roboţi reali, pot fi, acum, simulate; totuşi, comunitatea roboticii încurajează încă validare rezultatelor pe roboţi reali.

**Urmărirea pradă-prădător**

Este una dintre cele mai cunoscute medii în cercetările de învăţare în SMA, fiind simplu de implementat. Jocurile de urmărire constă într-un număr de agenţi (prădători) care vânează în mod cooperativ o pradă. Agenţii individuali prădători sunt, în general, mai puţin rapizi decât prada, şi, de cele mai multe ori, agenţii simt prada doar dacă sunt în apropierea ei. Ca urmare, agenţii au nevoie să coopereze în mod activ pentru a reuşi să captureze prada.

Una din abordările de învăţare folosite în literatură pentru a rezolva problema prăzii şi prădătorilor este ***învăţarea bazată pe cazuri*** (***case-based learning***).

**Explorare**

Domeniul constă într-o hartă mare cu agenţi (roboţi) şi elemente care trebuie găsite, spre exemplu comori. Sarcina este transportarea comorilor în locuri speciale. Eficienţa unei abordări constă în cât de repede sunt găsite toate comorile şi de numărul de comori descoperite într-un interval de timp fixat.

**Fotbal**

Jocul de fotbal, atât în simulare cât şi cu roboţi reali, este unul dintre cele mai utilizate domenii de cercetare în sistemele multiagent. Domeniul constă într-un teren de fotbal cu două porţi, o minge şi două echipe cu un anumit număr de agenţi (de la 5 la 11, depinzând de dimensiunea roboţilor şi de teren). Performanţa unei echipe este în general bazată pe numărul de goluri marcate, dar pot fi luate în calcul şi alte criterii de performanţă. Cea mai cunoscută astfel de competiţie, RoboCup, are diferite ligi împărţite pe continente şi pe tipuri de roboţi sau a mediilor de simulare (*www.robocup.org*).

### 7.3.7.2 Aplicaţii în lumea reală

În această secţiune vom prezenta câteva probleme derivate din domenii ale lumii reale, folosite în investigaţii anterioare în domeniul Sistemelor Multiagent. Cele mai multe astfel de domenii de probleme sunt probleme de logistică, planificare, satisfacere a constrângerilor, probleme care necesită decizii distribuite, luate în timp real. Deoarece aplicaţiile sunt în general distribuite şi foarte complexe, tehnicile de învăţare au fost destul de rar aplicate acestora, ca urmare, le vom prezenta în cee ce urmează ca fiind probleme provocatoare pentru învăţare în SMA.

**Monitorizarea distribuită a vehiculelor**

Sarcina în acest domeniu este optimizarea traficului într-o reţea de intersecţii. Fiecare intersecţie este echipată cu un semnalizator (semafor) controlat de un agent. Agenţii vor trebui să se coordoneze pentru a putea gestiona fluctuaţiile în trafic.

**Controlul traficului aerian**

În scopuri de securitate, spaţiul aerian este împărţit în regiuni tri-dimensionale folosite de controlori de trafic aerian pentru a ghida avioanele spre destinaţiile lor. Fiecare astfel de regiune are o limită superioară (numită *capacitatea sectorului*) în număr de avioane de care se poate ocupa la un moment dat. Scopul final este de a ghida avioanele dintr-un sector într-altul de-a lungul unor rute de lungime minimă, asigurând în acelaşi timp verificarea constrângerilor existente; soluţia trebuie să fie rapidă pentru a gestiona date în timp real.

**Gestionarea reţelelor şi rutare în reţele**

Acest domeniu constă în într-un număr mare de reţele distribuite. Agenţii sunt lansaţi în control cooperativ şi distribuit şi gestionează reţeaua, tratează avariile şi îşi echilibrează sarcinile.

**Gestionarea distribuirii electricităţii**

Aici problema este de a menţine o configuraţie de putere optimală în reţea pentru a servi toţi clienţii şi a minimiza pierderile; în acelaşi timp se gestionează posibilele distrugeri în reţea, cereri diverse din partea clienţilor, deteriorarea materialelor, etc.

**Îngrijire medicală distribuită**

Acest domeniu aplică Inteligenţa Artificială pentru a asista personalul clinic în diagnosticare, decizii legate de terapii şi teste, determină prescripţii şi efectuează alte sarcini medicale specifice. Problema este foarte potrivită pentru sisteme multiagent datorită descentralizării datelor şi resurselor, caracterului stohastic şi dinamic al datelor.

**Producţia industrială**

Această problemă clasică de planificare şi programare implică gestionarea procesului de producţie a elementelor complexe prin secvenţe de paşi, fiind diferite constrângeri şi costuri asociate fiecărui pas. Sarcina constă în construirea unui plan (o *secvenţă de producţie*) care specifică ordinea operaţiilor pentru diefrite elemente astfel încât costurile de producţie să fie minimizate, satisfăcând în acelaşi timp instrucţiunile clienţilor.