Deep Reinforcement Learning. Studiu de caz: Deep Q-Network

Mihnea Horia VREJOIU

Institutul Naţional de Cercetare-Dezvoltare în Informatică – ICI Bucureşti

mihnea.vrejoiu@ici.ro

Rezumat: Inteligenţa artificială (*Artificial Intelligence* – AI) a ajuns astăzi poate cel mai de actualitate subiect în numeroase domenii. Unul dintre obiectivele principale ale AI este crearea de agenţi complet autonomi capabili să interacţioneze cu lumea înconjurătoare şi să înveţe prin încercare şi eroare comportamente optime în diferite contexte, perfectibile în timp. Între metodele AI de învăţare automată (*machine learning*), învăţarea cu întărire / prin recompensă (*reinforcement learning* – RL) prin interacţiuni repetate cu mediul înconjurător cu urmărirea unui scop are un rol deosebit de important, pe lângă învăţarea supervizată şi respectiv nesupervizată. Totuşi, metodele clasice de RL au limitări importante în scalabilitatea la probleme cu dimensionalitate mai mare. În ultimii ani, tehnologiile de învăţare supervizată şi nesupervizată bazate pe *deep learning* cu utilizarea reţelelor neuronale adânci (*deep neural networks*), având proprietăţi remarcabile de aproximare a funcţiilor complicate pe spaţii multidimensionale, precum şi de învăţare de reprezentări ierarhice caracteristice extrase automat direct din date, cu reducere dimensională semnificativă, au cunoscut o dezvoltare explozivă producând rezultate impresionante comparabile cu, sau chiar peste performanţa umană în domenii diverse ca: recunoaşterea obiectelor / imaginilor, recunoaşterea vorbirii, traducerea automată etc. Combinarea metodelor RL cu cele *deep learning* a condus la ceea ce se numeşte astăzi *deep reinforcement learning* (DRL), oferind noi posibilităţi în realizarea de agenţi autonomi performanţi în spaţii multidimensionale. Lucrarea de faţă îşi propune o prezentare succintă a domeniului DRL şi studierea şi analizarea în detaliu a uneia dintre primele metode DRL de succes şi anume Deep Q-Network, dezvoltată de Google DeepMind.

Cuvinte cheie: învăţare cu/prin întărire/recompensă, agent, stare, acţiune, politică, diferenţă temporală, Q-learning, învăţare profundă, reţea neuronală adâncă, reţea neuronală convoluţională.

Deep Reinforcement Learning. Case Study: Deep Q-Network

Abstract: Artificial Intelligence (AI) became today perhaps the most up-to-date topic in many areas. One of the main goals of AI is to create completely autonomous agents able to interact with the surrounding world and learn by trial and error optimal behaviors in different contexts, perfectible in time. Among the machine learning methods of AI, reinforcement learning (RL) by repetitive interactions with the environment while targeting a purpose plays a particularly important role, besides supervised and unsupervised learning. However, classical RL methods have important limitations in scalability to higher-dimensionality problems. In recent years, supervised and unsupervised learning technologies based on deep learning, using deep neural networks with remarkable properties of approximating complex functions on multi-dimensional spaces, as well as the learning of characteristic hierarchical representations automatically extracted directly from data, with significant dimensional reduction, have had an explosive development, producing astonishing results comparable with, or even surpasing human performance in areas such as object / image recognition, speech recognition, automatic translation etc. The combination of RL with deep learning methods has led to what is now called deep reinforcement learning (DRL), providing new possibilities for producing autonomous agents in multidimensional spaces. This paper is proposing a brief presentation of the DRL field, while also studying and analyzing in detail one of the first successful DRL methods, namely Deep Q-Network developed by Google DeepMind.

Keywords: reinforcement learning, agent, state, action, policy, temporal difference, Q-learning, deep learning, deep neural network, convolutional neural network.

1. Introducere

Atunci când se pune problema reprezentării şi înţelegerii mecanismelor de învăţare, atât la nivel empiric şi intuitiv dar şi în majoritatea teoriilor despre învăţare şi inteligenţă, una dintre primele metode avute în vedere apare ca fiind învăţarea bazată pe interacţiunea cu mediul înconjurător prin percepţii senzoriale şi acţiuni. În acest mod pot fi acumulate informaţii despre diverse legături cauzale, despre rezultate şi consecinţe ale acţiunilor, precum şi despre ce anume

trebuie făcut pentru atingerea unui obiectiv/scop, acest tip de interacţiune constituind pentru om una dintre principalele surse de cunoştinţe şi cunoaştere despre mediul înconjurător şi despre sine.

Crearea de agenţi complet autonomi cu capabilităţi de a interacţiona cu mediul înconjurător şi de a-şi antrena singuri, pe bază de încercare şi eroare, comportamentul cel mai adecvat (perfectibil în timp) în diferite situaţii, constituie unul dintre obiectivele principale în domeniul inteligenţei artificiale. Aria de aplicabilitate acoperă de la roboţi care pot „simţi" şi interacţiona cu lumea înconjurătoare, până la agenţi software care interacţionează cu limbajul natural şi multimedia (Arulkumaran et al., 2017 [1]). Metoda de învăţare prin interacţiuni repetate, cu urmărirea atingerii unui anumit scop, este denumită *reinforcement learning* – RL (învăţare cu întărire / prin recompensă) şi reprezintă un alt tip de învăţare automată (*machine learning*), pe lângă învăţarea supervizată şi cea nesupervizată.

RL furnizează mediul matematic principial pentru învăţarea autonomă bazată pe experienţă. În RL, un agent interacţionează pas cu pas prin acţiuni cu mediul înconjurător, acţiunea sa la fiecare pas conducând la modificarea stării acestui mediu (tranziţia la o nouă stare) care îi întoarce ca răspuns (*feedback*) agentului câte o mărime scalară denumită recompensă (*reward*). Scopul agentului este maximizarea recompensei cumulate pe termen lung, după o secvenţă de paşi. În acest sens, RL este asemănătoare controlului optimal, utilizând tehnici specifice programării dinamice (procese de decizie Markov) dar fără un model matematic exact al mediului şi funcţionând pe spaţii cu dimensiuni mari. Prin învăţare în RL este stabilită o aşa numită politică pentru o problemă dată, prin care se defineşte comportamentul agentului la un moment dat. Astfel, o politică π asociază/mapează stările mediului înconjurător percepute de agent cu acţiunea cea mai potrivită a fi întreprinsă de către acesta din urmă în fiecare din stările respective. Agentul va acţiona conform politicii şi/sau ocazional va încerca aleator şi alte acţiuni care eventual ar putea îmbunătăţi politica.

Deşi cu ajutorul metodelor RL clasice au fost obţinute unele rezultate foarte bune, acestea s-au dovedit totuşi a fi destul de limitate, mai ales în privinţa scalabilităţii pentru probleme cu număr crescut de dimensiuni unde apar dificultăţi ţinând de complexitate. Astfel de inconveniente au putut fi depăşite însă odată cu apariţia succeselor spectaculoase obţinute în ultimii ani în diferite alte domenii (cum sunt recunoaşterea imaginilor, înţelegerea vorbirii, traducerea automată etc.) de *deep learning* cu utilizarea reţelelor neuronale adânci (*deep neural networks*) având proprietăţi remarcabile de aproximare a funcţiilor, precum şi de învăţare de reprezentări ierarhice caracteristice extrase automat direct din date, cu reducere dimensională semnificativă. Astfel, utilizarea algoritmilor de *deep learning* în conjuncţie cu metodele de RL a condus la ceea ce se numeşte astăzi generic *deep reinforcement learning* (DRL), oferind noi posibilităţi în realizarea de agenţi autonomi performanţi în spaţii multidimensionale de stări şi acţiuni în perspectiva creării de sisteme capabile să înveţe cum să se adapteze în lumea reală. Cu siguranţă DRL va fi o componentă importantă în realizarea sistemelor generale de IA (Lake et al., 2016 [9]).

Lucrarea de faţă îşi propune o prezentare succintă a domeniului DRL, cu studierea şi analizarea în detaliu a uneia dintre primele metode de DRL de succes şi anume Deep Q-Network (DQN), dezvoltată de Mnih et al., 2015 [15] de la Google DeepMind, ca metodă de referinţă în domeniu care îmbină o metodă RL (*Q-learning*) cu *deep learning* utilizând o reţea neuronală convoluţională. În continuare, lucrarea este structurată după cum urmează. Secţiunea 2 trece sumar în revistă elementele definitorii principale pentru RL, cu enumerarea în final a câtorva metode de RL de referinţă, menţionarea limitărilor acestora, precum şi a îmbunătăţirilor produse de noile metode de RL care valorifică avantajele aduse de revoluţia *deep learning*, fiind menţionate câteva aplicaţii cu succes ale acestora. În secţiunea 3 este analizată metoda DQN, cu detalierea arhitecturii, algoritmului şi tehnicilor speciale utilizate, a experimentelor efectuate şi a rezultatelor obţinute. Lucrarea se încheie cu secţiunea 4 în care sunt enunţate câteva concluzii relevante.

2. Reinforcement Learning

Reinforcement learning – RL (învăţarea cu întărire / prin recompensă) reprezintă un tip de metode de învăţare automată (*machine learning*) din cadrul inteligenţei artificiale, prin interacţiuni (continue) cu mediul cu urmărirea unui scop. În general, aceeaşi denumire – RL – referă totodată atât o problemă, o clasă de metode de soluţionare potrivite pentru aceasta, precum şi domeniul care

studiază aceste probleme şi metodele de soluţionare a lor. De aceea trebuie avută permanent în vedere distincţia conceptuală între acestea, în particular între probleme şi metode de soluţionare, pentru a se evita potenţiale confuzii. Practic, RL nu este definită prin descrierea/caracterizarea metodelor/algoritmilor de învăţare, ci prin caracterizarea problemei de învăţare, esenţială fiind perceperea aspectelor cele mai importante ale problemei reale puse în faţa agentului interacţionând cu mediul său înconjurător pentru a atinge scopul urmărit. Orice metodă/algoritm care rezolvă o astfel de problemă se consideră a fi metodă/algoritm de RL (Sutton & Barto, 1998 [20]).

În RL, un agent autonom controlat de un algoritm de învăţare automată descoperă prin încercări care acţiuni ale sale în raport cu mediul sunt cele mai recompensatoare în fiecare moment/situaţie. Agentul observă/percepe starea mediului său înconjurător s_t la momentul t, interacţionează cu acest mediu efectuând acţiunea a_t în starea s_t, determinând tranziţia mediului la momentul $t+1$ într-o nouă stare s_{t+1} care furnizează agentului o recompensă scalară r_{t+1} (*reward, reinforcement*) ca *feedback*. De cele mai multe ori, o acţiune nu are efect doar asupra recompensei imediate, ci şi asupra situaţiei/stării următoare şi, prin aceasta, asupra recompenselor ulterioare (viitoare) corespunzătoare. Căutarea de tip încercare-eroare (*try-error search*) şi recompensa întârziată (*delayed reward*) reprezintă cele două caracteristici definitorii în RL. Secvenţa de stări (şi acţiuni) parcursă de un agent în interacţiunea sa cu mediul este denumită **traiectorie**.

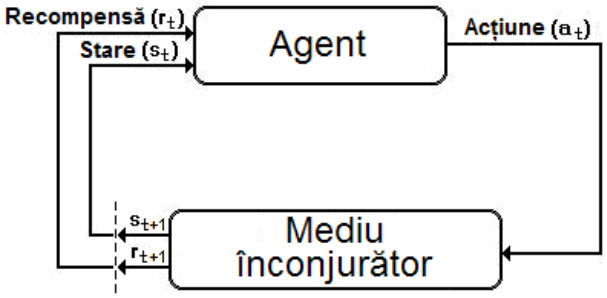

Figura 1. Interacţiunea agentului cu mediul (adaptare după: Sutton & Barto, 1998 [20])

Pe lângă **agentul** capabil să înveţe din experienţa proprie luând decizii de acţiune în funcţie de aceasta şi **mediul înconjurător** cu care agentul respectiv interacţionează, un sistem RL mai este compus din următoarele elemente:

a) o **politică** (*policy*),

b) un semnal de **recompensă** (*reward signal*),

c) o **funcţie de evaluare** (*value function*) şi

d) opţional, un **model al mediului** (*environment model*).

a) O **politică** π asociază/mapează stările mediului înconjurător percepute de agent cu acţiunea cea mai potrivită a fi întreprinsă de către acesta din urmă în fiecare din stările respective, determinând astfel comportamentul agentului RL la un moment dat. În unele cazuri, politica poate fi reprezentată printr-o tabelă de asocieri (*lookup table*) sau printr-o funcţie simplă, dar în alte cazuri poate implica calcule extensive cum sunt procesele de căutare. Politicile pot fi în general stohastice, precizând probabilităţi asociate fiecărei acţiuni posibile în fiecare stare.

b) La fiecare pas (în timp), urmare a unei acţiuni a agentului RL acesta primeşte de la mediu ca *feedback* o valoare numerică R denumită **recompensă** (*reward*). Un semnal de recompensă defineşte practic obiectivul unei probleme RL, indicând evenimentele „bune” şi respectiv „rele” pentru agent ca şi caracteristici definitorii imediate ale problemei abordate de către acesta. Pe baza semnalului de recompensă se poate realiza ajustarea/modificarea politicii: dacă o acţiune selectată de politică este urmată de o penalizare (sau recompensă slabă), atunci politica poate fi modificată să selecteze pe viitor o altă acţiune în respectiva situaţie. Semnalele de recompensă sunt adesea funcţii stohastice (probabilistice) de starea mediului şi de acţiunile întreprinse.

c) Scopul agentului este maximizarea recompensei cumulate pe termen lung. O **funcţie de evaluare** determină valoarea (utilitatea) unei stări *V(s)* ca măsură a recompensei totale pe care agentul se aşteaptă să o acumuleze în viitor pornind de la starea respectivă. Astfel, valoarea indică atractivitatea pe termen lung a unei stări a mediului ţinând cont şi de stările probabile să urmeze şi de recompensele disponibile în acestea, în timp ce recompensa determină atractivitatea intrinsecă imediată a stării. O stare *s* poate avea o recompensă imediată *R(s)* scăzută, dar totuşi o valoare *V(s)* mare dacă este urmată de alte stări cu recompense în general mari, sau vice-versa. În evaluarea şi luarea deciziilor se ţine cont de valori, fiind alese acele acţiuni care conduc la stări cu valoarea cea mai mare (şi nu neapărat cu recompensa cea mai mare), care indică obţinerea cantităţii celei mai mari de recompensă pe termen lung. În timp ce recompensele sunt oferite direct de mediu, valorile trebuie estimate şi re-estimate din secvenţa de observaţii pe care le face un agent de-a lungul existenţei sale. Analog valorii unei stări *V(s)* este definită şi valoarea unei acţiuni pentru o anumită stare (*action-value, quality* sau *Q-value*), *Q(s, a)*. La limită, valoarea unei stări este maximul dat de valoarea acţiunii optime din starea respectivă: *V(s) = Q*(s, a)*. Funcţia valorilor acţiunii *Q* asociază/mapează perechi stare-acţiune cu combinaţia cea mai valoroasă între recompensa imediată şi toate recompensele viitoare (cu ponderi diminuate la fiecare pas în timp) ce pot fi obţinute de acţiuni ulterioare pe traiectorie. O metodă eficientă de estimare a valorilor stărilor sau, respectiv, acţiunilor constituie practic componenta cea mai importantă a majorităţii algoritmilor RL.

d) Un al patrulea element component al unor sisteme RL poate fi reprezentat de un **model al mediului** care simulează comportamentul acestuia, permiţând inferenţe despre cum se va comporta mediul. De exemplu, pentru o pereche stare-acţiune dată, modelul poate „prezice" tranziţia la următoarea stare, precum şi următoarea recompensă. Modelele sunt utilizate pentru planificare, prin care se înţelege orice modalitate de decizie de acţiune considerând situaţiile viitoare posibile înainte ca acestea să fie efectiv experimentate. Metodele de rezolvare a problemelor RL care utilizează modele şi planificare se numesc metode bazate pe model (*on-model*), spre deosebire de metodele simple fără model (*off-model*) care învaţă politici optime exclusiv din încercări şi greşeli. Sunt utilizate şi metode RL mixte în care se învaţă din încercări şi erori şi simultan se învaţă şi un model al mediului care este folosit pentru planificare.

Obiectivul agentului este acela de a învăţa o politică (strategie de control) π care să maximizeze recompensa cumulată aşteptată din paşii următori (aportul recompenselor individuale fiind diminuat la fiecare astfel de pas cu un factor subunitar γ). Fiind dată o stare *s*, o politică π întoarce o acţiune de executat *a*. O politică optimală π^* maximizează recompensa aşteptată. În acest sens, RL este similar cu controlul optimal. Formalismul matematic al problemelor RL are la bază idei din teoria sistemelor dinamice şi programarea dinamică (Bellman, 1952; 1957 [4; 5]), cum este cazul controlului optimal în procesele de decizie Markov (*Markov Decision Processes* - MDP) incomplet cunoscute. În esenţă, pentru a învăţa, agentul trebuie:

- să aibă capabilitatea de a „simţi" (într-o anumită măsură) starea mediului înconjurător;
- să poată efectua acţiuni prin care să afecteze/modifice această stare;
- să aibă unul sau mai multe scopuri/obiective legate de starea mediului respectiv.

Procesele de decizie Markov includ aceste trei aspecte în formele lor cele mai simple fără a le trivializa pe niciunul. În RL, în general, nu este disponibil un model al dinamicii tranziţiilor de stare, agentul trebuind să înveţe despre consecinţele acţiunilor sale în mediu prin încercare şi eroare. Fiecare interacţiune cu mediul aduce informaţie pe care agentul o utilizează pentru a-şi actualiza cunoştinţele. Tabelul 1 prezintă sintetic o paralelă între MDP şi RL.

Tabelul 1. Paralelă sintetică între MDP şi RL

MDP sunt definite prin: • mulţimea se stări *S*, mulţimea de acţiuni *A*; • modelul de tranziţii $T(s_i, a, s_{i+1})$ cunoscut (determinist sau stohastic); • funcţia de recompensă $R(s)$ cunoscută; • se calculează o politică optimă π^* ($\pi^*(s)$ = acţiunea recomandată în starea *s*).	**RL** se bazează pe MDP, dar: • modelul de tranziţii este necunoscut; • funcţia de recompensă este necunoscută; • se învaţă o politică optimă.

Metodele de învăţare RL pot fi pasive, în care agentul execută o politică fixă pe care doar o evaluează (*on-policy*), sau active, fără politică fixă explicită (*off-policy*), în care agentul îşi actualizează politica pe măsură ce învaţă. Necesitatea compromisului între explorare şi exploatare (*exploration* vs. *exploitation*) este specifică metodelor de învăţare RL active. Pe de o parte, agentul trebuie să aleagă acele acţiuni dovedite din experimentări anteriorare ca eficiente în aducerea de recompensă (exploatare a experienţei acumulate), dar pe de altă parte, pentru descoperirea acestora, este necesară experimentarea de acţiuni neîncercate anterior, în căutarea uneia mai „profitabile" (explorare pentru acumularea de experienţă nouă). Aceasta este „dilema" agentului la un moment dat: trebuie să favorizeze explorarea stărilor şi acţiunilor necunoscute la momentul respectiv, sau exploatarea stărilor şi acţiunilor despre care ştie deja că aduc recompense mari? În general, agentul trebuie să mixeze acţiuni de amble tipuri şi, în mod progresiv, să le favorizeze pe acelea care apar ca fiind cele mai bune (e.g. metoda ε-greedy prin care se alege cu probabilitate ε o acţiune aleatoare şi cu probabilitate $1 - \varepsilon$ o acţiune optimă cunoscută, ε scăzând de la valoarea $1{,}0$ la o valoare reală mică subunitară pozitivă de prag pe parcursul unui număr mare de paşi, după care nu mai scade sub această valoare). Totodată, într-o sarcină stohastică, fiecare acţiune trebuie încercată de mai multe ori, pentru a se obţine o estimare de încredere privind recompensa aşteptată în urma acesteia.

Dintre metodele RL cele mai utilizate amintim (Sutton & Barto, 1998 [20]): programarea dinamică adaptivă, metoda Monte Carlo, metoda diferenţei temporale (*temporal difference* – TD), SARSA (*State-Action-Reward-State-Action*), *Q-learning* (Watkins, 1989 [34]; Watkins & Dayan, 1992 [35]), metode de tip actor-critic etc. Totuşi, deşi au condus la anumite succese, abordările clasice de RL prezintă limitări serioase în ceea ce priveşte scalabilitatea la un număr crescut de dimensiuni, fiind restricţionate de probleme ţinând de complexitate (de memorie, computaţională, de exemple).

Progresele spectaculoase din ultimii ani în domeniul *deep learning* bazate pe proprietăţile excepţionale ale reţelelor neuronale adânci (*deep neural networks*) de aproximare a funcţiilor şi de găsire automată de reprezentări compacte de dimensiuni reduse (caracteristici) ale datelor de mari dimensiuni, au oferit noi mijloace pentru depăşirea acestor limitări. Utilizarea algoritmilor *deep learning* în RL defineşte astăzi ceea ce numim *deep reinforcement learning* (DRL), deschizând noi perspective în realizarea de agenţi autonomi performanţi în spaţii multidimensionale de stări şi acţiuni. O primă aplicaţie revoluţionară utilizând DRL a putut învăţa să joace cu performanţe comparabile cu, sau chiar peste cele umane o gamă de jocuri video Atari 2600 direct din pixelii din imaginile acestora (Mnih et al., 2015 [15]), demonstrând că agenţi RL pot fi antrenaţi direct din date brute multidimensionale (doar) pe baza unui semnal de recompensă. Un al doilea succes important a fost reprezentat de dezvoltarea sistemului DRL hibrid AlphaGo, care a învins campionul mondial la Go (Silver et al., 2016 [19]), utilizând reţele neuronale antrenate prin învăţare supervizată şi RL, combinate cu un algoritm tradiţional de căutare euristică.

Algoritmi DRL au fost de asemenea deja utilizaţi într-o gamă largă de aplicaţii de robotică, în care politicile de control al roboţilor pot fi învăţate direct din imagini capturate din lumea reală (Levine et al., 2016 [10; 11]). Mai mult, DRL a permis crearea de agenţi capabili să „înveţe să înveţe" – meta-învăţare – (Duan et al., 2016 [8]; Wang et al., 2017 [32]), ajungând astfel să poată generaliza la medii vizuale complexe pe care nu le-au mai „văzut" anterior (Duan et al., 2016 [8]).

DRL oferă posibilităţi reale pentru crearea de sisteme capabile să înveţe cum să se adapteze în lumea reală, permiţând automatizarea a tot mai multe sarcini fizice concrete, de la gestionarea consumului de energie (Tesauro et al., 2008 [25]) la apucarea şi depunerea obiectelor (Levine et al., 2016 [11]). Totodată, RL fiind o metodă generală pentru abordarea problemelor de optimizare prin încercare şi eroare, DRL a fost deja utilizată în diverse sarcini legate de învăţarea automată, de la proiectarea de modele avansate (*state-of-the-art*) de traducere automată (Zoph & V Le, 2017 [36]), la crearea de noi funcţii de optimizare (Li & Malik, 2017 [12]). DRL va fi o componentă importantă în realizarea sistemelor generale de IA (Lake et al., 2016 [9]).

3. Deep Q-Network (DQN)

Modul de reprezentare şi stocare a funcţiilor de evaluare şi/sau a politicilor constituie una din provocările majore la aplicarea RL în probleme din lumea reală. O reprezentare exhaustivă printr-o tabelă de asocieri (*lookup table*) este posibilă numai în cazuri simple, pentru care mulţimea stărilor este finită şi suficient de restrânsă. În general, trebuie utilizată o aproximare printr-o funcţie parametrică, liniară sau neliniară după caz, pe baza unor caracteristici (*features*) conţinând informaţia necesară şi care să fie disponibile pentru sistemul de învăţare. Astfel, majoritatea aplicaţiilor RL depind în mare măsură de seturi de caracteristici atent şi migălos manufacturate anterior pe baza cunoştinţelor şi intuiţiei umane despre fiecare problemă specifică abordată.

Odată cu popularizarea algoritmului *backpropagation* ca metodă de învăţare a reprezentărilor interne (a caracteristicilor) specifice problemei / sarcinii (Rumelhart et al., 1986 [17]), au început să fie utilizate pentru aproximarea funcţiilor în RL reţelele neuronale artificiale (RNA) cu mai multe straturi ascunse. Au apărut astfel aplicaţii cu performanţe comparabile cu, sau chiar peste cele umane ca de exemplu: TD-Gammon (Tesauro et. al., 1992; 1994; 1995; 2002 [21; 22; 23; 24]) în abordarea jocului de table (*backgammon*), sistem care a fost ulterior adaptat şi la crearea strategiei de pariere „Daily Double" utilizate de IBM Watson în Jeopardy! (Tesauro et al., 2012; 2013 [26; 27]) alături de alte strategii avansate. Majoritatea acestor aplicaţii necesitau ca intrare a reţelei un set de caracteristici spcifice, special manufacturate pentru problema dată.

Mnih et al. (2013; 2015) [14; 15] de la Google DeepMind au realizat o demonstraţie impresionantă a modului în care o reţea neuronală artificială adâncă poate automatiza procesul de generare/extragere a caracteristicilor. Aceştia au dezvoltat un agent RL numit „Deep Q-Network" (DQN), combinând pentru prima dată *Q-learning* cu o reţea neuronală adâncă de tip convoluţional (*convolutional neural network*) specializată în procesarea imaginilor (Vrejoiu, 2019 [31]) şi au arătat că acest agent poate atinge un nivel înalt de performanţă pentru o colecţie de probleme diferite fără a necesita câte un set de caracteristici prefabricate specifice fiecăreia dintre acestea.

Reţeaua convoluţională este utilizată pentru aproximarea funcţiei optimale de evaluare a valorii aşteptate a acţiunilor (*optimal action-value function*):

$$Q^*(s, a) = max_\pi \boldsymbol{E}[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + ... \mid s_t = s, a_t = a, \pi], \quad (1)$$

dată de valoarea aşteptată maximă a sumei recompenselor R, diminuate cu factorul de reducere γ la fiecare pas ulterior în timp, care pot fi obţinute cu o politică $\pi = P(a|s)$ după ce se face o observaţie s şi se execută o acţiune a. Ponderile reţelei convoluţionale reprezintă parametrii funcţiei de evaluare a valorii acţiunilor (*action-value function*) care aproximează funcţia optimală, $Q(s, a; w) \approx Q^*(s, a)$.

Ideea de bază a mai multor algoritmi de RL utilizată şi de DQN constă în îmbunătăţirea estimării funcţiei valorii acţiunii Q – practic a aproximării parametrice a acesteia – prin actualizări iterative pe baza estimării curente (*bootstrapping*) utilizându-se ecuaţia Bellman (Bellman, 1952 [4]) pentru valorile din oricare două iteraţii succesive:

$$Q_{i+1}(s, a; w) = \boldsymbol{E}_{s'}[r + \gamma max_{a'} Q_i(s', a'; w) \mid s, a]. \quad (2)$$

DQN a fost antrenat să joace prin interacţiune cu un emulator un număr de 49 de jocuri video Atari 2600 diferite. Practic, DQN a învăţat câte o politică diferită pentru fiecare dintre cele 49 de jocuri – ponderile reţelei convoluţionale fiind resetate la valori aleatoare înainte de învăţarea fiecărui nou joc, – dar a utilizat aceeaşi intrare (pixeli imagine), aceeaşi arhitectură a reţelei şi

aceleaşi valori ale parametrilor (e.g. mărimea pasului, factorul de reducere, parametrii de explorare şi alţi parametri specifici implementării) pentru toate jocurile. Chiar dacă aceste jocuri au în comun faptul că sunt jucate prin urmărirea fluxurilor de imagini video, ele sunt foarte diferite în alte aspecte: acţiunile au efecte diferite, dinamica tranziţiei stărilor e diferită şi necesită politici diferite pentru a se obţine scoruri mari. Totuşi, DQN a atins niveluri de joc comparabile sau superioare nivelului uman pentru o mare parte dintre aceste jocuri, reţeaua convoluţională învăţând să transforme intrarea comună tuturor jocurilor (cadrele video) în caracteristici specifice, specializate în reprezentarea valorii acţiunilor necesare pentru a juca la nivelul înalt atins de DQN pentru majoritatea jocurilor.

În perioada 1977-1992, Atari Inc. a produs şi comercializat cu mare succes diverse versiuni ale consolei de jocuri video pentru acasă Atari 2600, care a introdus sau popularizat numeroase jocuri video Arcade clasice (e.g.: Pong, Breakout, Space Invaders, Asteroids etc.), încă jucate cu entuziasm şi astăzi. În paralel, aceste jocuri au reprezentat de asemenea şi excelente medii de testare pentru dezvoltarea şi evaluarea metodelor de RL (Diuk et al., 2008 [7]; Naddaf, 2010 [16]; Cobo et al., 2011 [6]; Bellemare et al., 2012 [2]). Pentru a facilita utilizarea jocurilor Atari 2600 în testarea şi evaluarea algoritmilor de învăţare şi planificare, Bellemare et al., 2013 [3] au dezvoltat emulatorul şi mediul de învăţare disponibil public denumit Arcade Learning Environment (ALE).

Toate acestea au constituit premizele care au condus la alegerea colecţiei de jocuri Atari 2600 pentru demonstraţia grupului Mnih et al., care a fost de asemenea inspirat şi de performanţa la nivel uman atinsă de TD-Gammon. Astfel, DQN se aseamănă până la un punct cu TD-Gammon în utilizarea unei reţele neuronale artificiale adânci ca metodă de aproximare de funcţii neliniare cu forma cu semi-gradient a algoritmului TD al *Q-learning*, fără model şi fără politică fixă explicită (*model-free* şi *off-policy*) şi gradienţii calculaţi cu *backpropagation*.

Arhitectura de bază a DQN, reprezentată schematic în Figura 2, este una de tip reţea neuronală convoluţională adâncă, cu reducere dimensională (*sub-sampling*) în fiecare strat convoluţional şi hărţi de caracteristici (*feature maps*) compuse din neuroni conectaţi fiecare doar la câte un grup local de neuroni din stratul anterior, care alcătuiesc câmpul lor receptor (*receptive field*). Arhitectura exactă este prezentată în cele ce urmează, detaliile fiind sintetizate (şi) în Tabelul 2. La intrare, reţeaua DQN a utilizat cadrele 210×160 pixeli cu 128 de culori (cu componente R, G, B) la 60 Hz ale celor 49 de jocuri Atari, dar nu ca atare în mod direct. Din considerente de reducere a memoriei şi procesărilor necesare, aceste cadre au fost mai întâi preprocesate fiind transformate în matrici de 84×84 de valori ale luminanţei ($Y = 0{,}2126 \cdot R + 0{,}7152 \cdot G + 0{,}0722 \cdot B$). Totodată, deoarece stările complete ale multora din jocurile Atari nu sunt integral observabile dintr-un singur cadru video, dar şi pentru a putea fi surprinsă dinamica elementelor de interes şi viteza acestora, au fost grupate „stive” (*stacks*) de câte 4 astfel de matrici în niveluri de gri provenite din cele mai recente 4 cadre video, acestea constituind practic intrarea reţelei, de dimensiune 84×84×4.

Reţeaua DQN este compusă din 3 straturi ascunse convoluţionale succesive, urmate de un al patrulea strat ascuns complet conectat, urmat de stratul de ieşire la rândul său complet conectat la acesta din urmă. Cele 3 straturi convoluţionale succesive (Conv1 – Conv3) utilizează: 32 filtre 8×8 cu pas (*stride*) 4, 64 filtre 4×4 cu pas 2 şi respectiv 64 filtre 3×3 cu pas 1, producând: 32 de hărţi de caracteristici de dimensiuni 20×20, 64 de hărţi de caracteristici de dimensiuni 9×9 şi respectiv 64 de hărţi de caracteristici de dimensiuni 7×7. Funcţia de activare a fiecărui neuron din fiecare hartă de caracteristici este ReLU (*Rectified Linear Unit*: max(0, x)). Cei 3.136 (7×7×64) neuroni din cel de-al treilea strat convoluţional sunt fiecare conectaţi la fiecare dintre cei 512 neuroni ReLU din stratul complet conectat (*fully connected*) FC1, care la rândul lor sunt fiecare conectaţi la toate cele 18 unităţi ale stratului de ieşire FC2, câte una pentru fiecare acţiune posibilă într-un joc Atari. Nivelurile de activare ale ieşirilor DQN sunt date de valorile optimale ale acţiunilor, $Q^*(s, a)$, estimate/prezise pentru starea reprezentată de intrarea respectivă a reţelei.

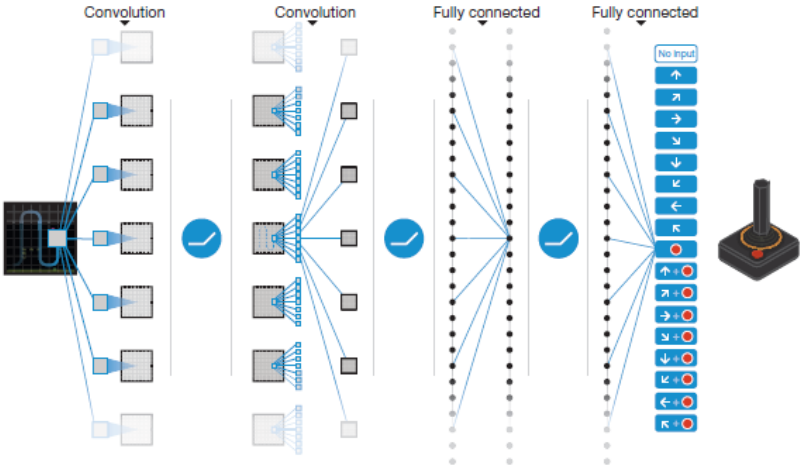

Figura 2. Ilustrarea schematică de principiu a rețelei neuronale convoluționale DQN (sursa: Mnih et al., 2015 [15])

Tabelul 2. Detalierea sintetică a arhitecturii DQN

Strat	Intrare	Dim. filtre	Pas	Nr. filtre	Activare	Ieşire
Conv1	84 × 84 × 4	8 × 8	4	32	ReLU	20 × 20 × 32
Conv2	20 × 20 × 32	4 × 4	2	64	ReLU	9 × 9 × 64
Conv3	9 × 9 × 64	3 × 3	1	64	ReLU	7 × 7 × 64
FC1	7 × 7 × 64			512	ReLU	512
FC2	512			18	Liniară	18

Principalul avantaj al acestui tip de arhitectură este reprezentat de faptul că permite estimarea valorilor pentru toate acțiunile posibile într-un singur pas de parcurgere înainte (*forward*) a rețelei. Asocierea unităților de ieşire cu acțiunile diferă de la un joc la altul şi, deoarece numărul acțiunilor posibile a variat între 4 şi 18, nu toate unitățile de ieşire au avut un rol funcțional în toate jocurile. Rețeaua poate fi gândită ca 18 rețele separate, câte una pentru estimarea valorii optime pentru fiecare acțiune posibilă. În realitate, aceste rețele partajază straturile inițiale, dar unitățile de ieşire învață să utilizeze caracteristicile extrase de acestea în moduri diferite.

La învățare, semnalul de recompensă utilizat în DQN a fost standardizat pentru toate jocurile printr-un singur parametru asociat fiecărui pas, care indică modul în care se modifică scorul jocului de la un pas la următorul în timp prin valorile:

- +1 pentru cazurile în care scorul creşte,
- −1 pentru acelea în care scorul scade,
- 0 dacă scorul rămâne nemodificat,

ceea ce a permis buna funcționare pentru toate jocurile în ciuda plajelor variate de evoluție a scorurilor acestora. Această modalitate de trunchiere standardizată a recompenselor limitează amplitudinea derivatelor erorii (la calculul gradienților în *backpropagation*) şi permite utilizarea aceleiaşi rate de învățare α pentru mai multe jocuri.

DQN utilizează un algoritm fără model (*model-free*), folosind direct cadrele video (imaginile) provenite de la emulator fără estimare explicită a dinamicii recompenselor şi tranziților P(r, s'|s, a), precum şi fără politică fixă explicită (*off-policy*), învățând o politică (*greedy*) $a = argmax_{a'}(Q(s, a'; w))$ urmărind o distribuție de comportamente care să asigure explorarea adecvată a spațiului stărilor. Practic, o politică ε-greedy asigură selecția acestei distribuții la fiecare pas, cu probabilitate $1 - \varepsilon$ de alegere aleatoare uniformă a unei acțiuni optime deja experimentate anterior,

alternând cu selectarea cu probabilitate ε a unei acţiuni noi. Parametrul $\varepsilon \in [0, 1]$ descreşte liniar de la *1,0* la *0,1* pentru primul milion de cadre, rămânând apoi nemodificat la valoarea *0,1* pentru tot restul sesiunii de învăţare. Astfel, în primele etape ale învăţării este favorizată explorarea situaţiilor care iniţial sunt toate noi, pentru ca treptat să ajungă să fie favorizată exploatarea experienţelor deja învăţate anterior, permiţându-se totuşi ocazional şi explorarea câte unei acţiuni potenţial noi care poate eventual duce la o îmbunătăţire a experienţei. Valorile celorlalţi diferiţi hiperparametri şi parametri de optimizare, precum mărimea pasului de învăţare (*learning step size*), factorul de reducere (*discount rate*) şi alţii specifici implementării, au fost selectate empiric prin căutări informale pentru un număr mic (cinci) de jocuri selectate pentru a se vedea care valori au funcţionat cel mai bine. Aceste valori au fost apoi păstrate fixe pentru toate celelalte jocuri.

După ce DQN selectează la un moment dat t o acţiune A_t, aceasta este executată de către emulatorul de jocuri care întoarce o recompensă R_{t+1} şi următorul cadru video. Acest nou cadru este preprocesat şi actualizează stiva ultimelor 4 cadre cele mai recente S_t care devine astfel următoarea intrare S_{t+1} pentru reţea.

Pentru ajustarea ponderilor reţelei, DQN utilizează următoarea formă cu semi-gradient de diferenţă temporală (*temporal difference* – TD) din *Q-learning*:

$$w_{t+1} = w_t + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a; w_t) - Q(S_t, A_t; w_t)\right] \nabla Q(S_t, A_t; w_t) \quad (3)$$

unde w_t este vectorul ponderilor reţelei, A_t este acţiunea selectată la momentul t, S_t şi S_{t+1} repezintă stivele de imagini preprocesate aplicate la intrarea reţelei la momentele t şi respectiv $t+1$, α este rata de învăţare ($\alpha = 0{,}00025$), R_{t+1} este valoarea recompensei la momentul $t+1$, γ este factorul de reducere a recompensei de la fiecare pas următor ($\gamma = 0{,}99$), a sunt acţiunile posibile la momentul $t+1$, iar Q este valoarea funcţiei de evaluare a valorii acţiunii (*action-value function*) pentru o pereche stare-acţiune, aproximată parametric prin ponderile curente ale reţelei, w_t .

În DQN, gradientul este calculat cu *backpropagation*, considerându-se câte o reţea separată pentru fiecare acţiune, pentru ajustarea la momentul t aplicându-se *backpropagation* numai pe reţeaua corespunzătoare acţiunii A_t. A fost utilizată funcţia *loss* Huber (pătratică pentru valori mici ale argumentului şi liniară în rest) şi metoda *stochastic gradient descent* (SGD) pe mini-loturi (*mini-batches*), cu ajustarea ponderilor numai după acumularea informaţiei de gradient pe câte un lot restrâns de imagini (în acest caz, 32), faţă de procedura *gradient descent* standard cu ajustarea ponderilor la fiecare iteraţie (acţiune). Pentru accelerarea învăţării a fost utilizat algoritmul de optimizare RMSProp (Tieleman & Hinton, 2012 [28]) bazat pe creşterea gradientului (*gradient-ascent*), cu determinarea dinamică a pasului de ajustare pentru fiecare pondere pe baza mediei mobile (*moving average*) a magnitudinilor gradienţilor recenţi pentru ponderea respectivă.

În *Q-learning*, atât intrările cât şi ţintele se modifică continuu de-a lungul procesului şi fac ca învăţarea să fie instabilă. Valorile ţintă Q depind în fiecare iteraţie de valoarea Q la momentul respectiv (ţintă nestaţionară). Totodată, în cazul învăţării secvenţiale din stări consecutive, apropiate pe o anumită traiectorie, corelaţiile inerente între acestea vor determina şi creşterea valorii *Q ţintă* odată cu creşterea valorii Q curente, ceea ce pune probleme de convergenţă. Pentru a depăşi aceste inconveniente, metoda utilizată efectiv de grupul de la DeepMind pentru funcţionarea DQN utilizează trei tehnici importante care modifică din punct de vedere funcţional procedura standard *Q-learning*.

Prima tehnică utilizată este denumită „reluarea experienţei” (*experience replay*) şi a fost introdusă iniţial de Lin (1992) [13], şi constă în păstrarea/stocarea iniţială a experienţei agentului la fiecare moment de timp într-o „memorie de reluare” (*replay memory*) şi utilizarea fiecărei astfel de experienţe memorate pentru actualizarea ponderilor mai târziu, la momente ulterioare. Astfel, în DQN, după executarea de către emulatorul de jocuri a unei acţiuni A_t din starea reprezentată de stiva de imagini S_t urmată de întoarcerea recompensei R_{t+1} asociate stivei de imagini S_{t+1}, tuplul (S_t, A_t , R_{t+1}, S_{t+1}) este adăugat în această memorie de reluare – practic un *buffer* ciclic – în care se acumulează experienţele mai multor sesiuni/episoade pentru acelaşi joc (concret, cele mai recente 1 milion de cadre video). La învăţare, la fiecare pas se fac mai multe actualizări *Q-learning*, fiecare pe baza câte unui mini-lot (*mini-batch*) de experienţe eşantionate aleator uniform din această memorie de reluare. Astfel, în loc ca S_{t+1} să devină noua S_t şi să fie utilizată pentru următoarea

actualizare ca în *Q-learning*, o experienţă neconectată din mini-lotul curent extras din memoria de reluare va fi cea utilizată la următoarea actualizare. Aceasta este posibil deoarece *Q-learning* este un algoritm fără politică fixă explicită (*off-policy*) şi nu necesită să fie aplicat de-a lungul unor traiectorii conectate (secvenţe de stări corelate). Această tehnică de reluare a experienţei oferă câteva avantaje faţă de forma uzuală a *Q-learning*. În acest mod, DQN învaţă mai eficient din propriile experienţe prin utilizarea fiecărei experienţe stocate în memoria de reluare pentru (mai) multe actualizări. De asemenea, este redusă varianţa deoarece actualizările succesive nu mai sunt corelate între ele ca în *Q-learning* standard. Totodată, este eliminată dependenţa experienţelor succesive de aceleaşi ponderi curente care se actualizează, ceea ce elimină o sursă de instabilitate.

A doua tehnică utilizată de DQN şi care modifică *Q-learning* standard a vizat direct îmbunătăţirea stabilităţii metodei. Valoarea ţintă utilizată la o actualizare în *Q-learning* depinde de estimarea curentă a funcţiei de evaluare a valorii acţiunii (*action-value function*, *Q*) care este reprezentată în DQN printr-o aproximare parametrică, depinzând chiar de parametrii care se actualizează ai reţelei. Astfel, pentru actualizarea dată de relaţia (3) valoarea ţintă este: $\gamma \max_a Q(S_{t+1}, a; w_t)$, dependenţa acesteia de ponderile w_t (care nu apare în cazul învăţării supervizate simple în care ţintele nu depind de parametrii care se actualizează) putând conduce la oscilaţii şi/sau divergenţă (Tsitsiklis & Roy, 1997 [29]). Pentru a se evita astfel de situaţii, în DQN este utilizată o tehnică prin care valorile ponderilor reţelei valorii acţiunii la care s-a ajuns după un număr C ($C = 10.000$) de actualizări efectuate asupra lor sunt preluate/„îngheţate" într-o (altă) reţea duplicat şi menţinute fixe în aceasta pentru următoarele C actualizări ş.a.m.d. Pentru fiecare actualizare, este utilizată ca valoare ţintă *Q-learning* ieşirea corespunzătoare a acestei reţele duplicat pentru intrarea respectivă aplicată. Notând cu Q^- ieşirea reţelei duplicat/„ţintă" şi cu w^- ponderile fixe ale acesteia, regula de actualizare (3) devine:

$$w_{t+1} = w_t + \alpha\,[R_{t+1} + \gamma \max_a Q^-(S_{t+1}, a; w^-) - Q(S_t, A_t; w_t)]\ \nabla Q(S_t, A_t; w_t), \quad (3')$$

Reluarea experienţei este principala „responsabilă" pentru îmbunătăţirea performanţei DQN. Reţeaua ţintă aduce şi ea o îmbunătăţire semnificativă dar nu atât de critică, însă devine mai importantă când capacitatea reţelei este mică.

În fine, o a treia modificare a mai fost adusă *Q-learning* standard, având de asemenea drept scop îmbunătăţirea stabilităţii şi a constat în limitarea termenului de eroare TD din (3'), $\delta = R_{t+1} + \gamma \max_a Q^-(S_{t+1}, a; w^-) - Q(S_t, A_t; w_t)$, astfel încât să se menţină în intervalul $[-1, 1]$:

$$\delta' = \max(-1, \min(1, \delta)) \in [-1, 1]. \quad (4)$$

Această limitare corespunde utilizării unei funcţii valoare absolută (modul) pentru erorile situate în afara intervalului $(-1, 1)$ deoarece funcţia *loss* valoare absolută $|x|$ are derivata -1 pentru toate valorile x negative şi derivata 1 pentru toate valorile x pozitive.

Algoritmul *deep Q-learning* cu reluarea experienţei aplicat în DQN pe M episoade (sesiuni) ale unui joc poate fi descris după cum urmează (Φ este stiva ultimelor 4 cadre preprocesate 84×84):

```
Iniţializare memorie de reluare D de capacitate N;

Iniţializare funcţie aproximare valori acţiuni Q cu ponderi aleatoare w;

Iniţializare funcţie valori ţintă acţiuni Q⁻ cu ponderi w⁻ = w;

Pentru episod ep = 1 ÷ M execută:
   {
   Iniţializare secvenţă S1 = {x1} şi stivă cadre preprocesate Φ1 = Φ(S1);

   Pentru pas t = 1 ÷ T execută:
      {
      Cu probabilitate ε: selectare acţiune aleatoare At,
      altfel: selectare At = argmaxa Q(Φ(St), a; w);

      Execuţie acţiune At în emulator, cu întoarcere recompensă Rt şi
      imagine xt+1;
```

```
        Setare S_{t+1} = S_t, A_t, x_{t+1} şi preprocesare Φ_{t+1} = Φ(S_{t+1});

        Memorare tranziţie (Φ_t, A_t, R_t, Φ_{t+1}) în D;

        Eşantionare mini-lot de 32 tranziţii (Φ_j, a_j, r_j, Φ_{j+1}) din D;

        Pentru fiecare tranziţie din mini-lot:
           dacă episodul se termină la pasul j + 1, atunci: setare y_j = r_j,
           altfel: setare y_j = r_j + γ max_{a'} Q⁻(Φ_{j+1}, a'; w⁻);

        Execuţie un pas gradient descent pe suma pe mini-lotul curent a
        erorilor pătratice (y_j - Q(Φ_j, a_j; w))² relativ la parametrii
        reţelei w, cu actualizare corespunzătoare a acestora;

        La fiecare C paşi: resetare Q⁻ = Q (cu ponderi w⁻ = w);
        }
    }
```

Pentru dezvoltarea şi implementarea iniţială a DQN a fost utilizat limbajul Lua pe o maşină Linux cu GPU şi NVIDIA® CUDA® (v.5.5), LuaJIT şi Torch 7.0, nngraph, Xitari (o ramificaţie a Arcade Learning Environment, ALE – Bellemare et al., 2013 [3]) şi AleWrap (o interfaţă Lua la Xitari) instalate. Codul sursă original este disponibil online pentru utilizare în scopuri necomerciale la adresa *https://sites.google.com/a/deepmind.com/dqn*.

Mnih et al., 2015 [15] au evaluat şi analizat contribuţia diferitelor tehnici caracteristice utilizate în proiectarea DQN amintite mai sus şi modul cum afectează acestea performanţa, prin efectuarea mai multor sesiuni/episoade de învăţare pentru o selecţie de 5 dintre jocurile Atari cu testarea tuturor celor 4 combinaţii posibile cu reluarea experienţei şi reţeaua duplicat ţintă incluse sau neincluse. Chiar dacă rezultatele au diferit de la joc la joc, s-a constatat că atât fiecare dintre aceste caracteristici individual a condus la o îmbunătăţire a performanţei, cât mai ales utilizarea ansamblului tuturor acestora a îmbunătăţit substanţial performanţa. Totodată, a fost analizat rolul jucat de reţeaua neuronală convoluţională adâncă în abilitatea de învăţare a DQN prin compararea cu o versiune utilizând o reţea cu doar un singur strat liniar, ambele primind la intrare aceleaşi cadre video preprocesate şi grupate în stive de câte 4. Îmbunătăţirea adusă de versiunea convoluţională adâncă faţă de cea liniară a fost remarcabilă pentru toate cele 5 jocuri de test. Per global, scorurile obţinute de DQN au fost comparate cu cele obţinute de sistemul de învăţare cel mai performant până la momentul respectiv (Bellemare et al. 2013 [3]), de un specialist expert în testare uman, precum şi de un agent selectând acţiunile aleator. DQN a învăţat fiecare joc interacţionând cu emulatorul pentru 50 de milioane de cadre, ceea ce corespunde la circa 38 de zile de experienţă cu jocul. La începutul învăţării fiecărui joc, ponderile reţelei DQN au fost resetate la valori aleatoare. Pentru evaluare, au fost mediate scorurile a câte 30 de sesiuni/episoade pentru fiecare joc, fiecare sesiune/episod începând dintr-o stare iniţială aleatoare a jocului şi durând până la maxim 5 minute. DQN a obţinut performanţe mai bune ca cel mai bun sistem RL anterior pentru 40 de jocuri din cele 49 şi mai bune ca expertul uman în 22 de jocuri, iar comparabil cu acesta (cu scoruri peste 75% din scorurile acestuia) în încă alte 7 jocuri din cele 49. În afara acestor performanţe remarcabile, ceea ce este cel mai important este faptul că acelaşi sistem de învăţare a atins aceste niveluri de performanţă pe o gamă variată de jocuri fără a se baza pe cunoştinţe şi modificări specifice pentru fiecare dintre acestea.

Ulterior, au fost aduse (tot la DeepMind) mai multe îmbunătăţiri DQN, ca de exemplu: **Double DQN** (van Hasselt et al., 2016 [30]), utilizând practic reţeaua *online* pentru alegerea *greedy* a acţiunii şi reţeaua ţintă pentru estimarea valorii *Q*; **Prioritized experience replay** (Schaul et al., 2016 [18]) acordând o pondere mai mare ca frecvenţă de apariţie în eşantionările din memoria de reluare (*replay memory*) acelor tranziţii cu diferenţe mai mari faţă de ţintă (cu eroare TD mai mare), care pot aduce un progres mai rapid al învăţării; **Dueling DQN** (Wang et al., 2016 [33]) calculând valorile funcţiei *Q(s, a)* prin compunerea funcţiei valorilor stărilor *V(s)* cu funcţia avantajului acţiunii *A(s, a)*; **Noisy Nets for exploration**, adăugând zgomot parametric stratului liniar final pentru facilitarea explorării, ca alternativă la selectarea *ε-greedy* a acţiunilor.

4. Concluzii

Crearea de agenţi artificiali autonomi capabili să exceleze într-o colecţie de sarcini dificile diverse a reprezentat un deziderat dificil de atins al inteligenţei artificiale. Multă vreme, necesitatea manufacturării de reprezentări specifice pentru fiecare tip de problemă în parte, precum şi problemele de scalabilitate în cazul spaţiilor cu dimensionalităţi mari au constituit piedici serioase în abordările de învăţare automată (*machine learning*) vizând atingerea acestui scop. Proprietăţile şi performanţele remarcabile ale *deep learning* utilizând reţele neuronale adânci (*deep neural network*) combinate cu metode de învăţare cu întărire / prin recompensă (*reinforcement learning*) din interacţiuni repetate cu mediul au permis recent depăşirea acestor neajunsuri. Deep Q-Network (DQN) este unul dintre primele exemple de referinţă în *deep reinforcement learning*.

Prin DQN s-a demonstrat că un acelaşi agent poate învăţa caracteristici specifice problemei astfel încât să capete abilităţi peste, sau comparabile cu cele umane pentru o plajă relativ variată de sarcini. Chiar dacă jocurile Atari necesită aptitudini/abilităţi diverse, DQN a învăţat să le joace cu performanţe remarcabile exclusiv prin observarea imaginilor video ale acestora. În acest context, utilizarea reţelei neuronale convoluţionale a reprezentat practic alegerea naturală potrivită. Chiar dacă DQN nu este un agent care excelează simultan în toate sarcinile (învăţarea fiind făcută separat pentru fiecare dintre jocuri), a dovedit cu succes faptul că prin utilizarea tehnologiei *deep learning* se poate reduce şi posibil elimina necesitatea proiectării şi ajustării pentru fiecare problemă specifică. Totuşi, trebuie făcută precizarea că DQN nu reprezintă o soluţie completă pentru problema învăţării independente de sarcină (*task-independent learning*). Performanţa DQN a fost chiar considerabil sub nivelurile de abilitate umană pe câteva dintre jocurile Atari 2600, unele dintre acestea necesitând planificare pe un orizont lung de timp (*deep planning*), ceea ce excede capabilităţilor DQN. Pe de altă parte, trebuie menţionat şi faptul că abilitatea de învăţare controlată prin exersare extensivă repetată, aşa cum DQN a învăţat să joace jocurile Atari, este doar una dintre multiplele modalităţi de învăţare pe care oamenii le realizează în mod obişnuit.

Chiar şi cu limitările amintite mai sus, realizarea DQN a produs un important pas înainte în domeniul învăţării automate (*machine learning*), demonstrând noile posibilităţi deschise prin mixarea metodelor de învăţare cu/prin întărire/recompensă (*reinforcement learning*) cu tehnologiile moderne extrem de performante de *deep learning* şi capacităţile de reprezentare a caracteristicilor din datele de intrare şi de aproximare a funcţiilor neliniare ale reţelelor neuronale adânci.

Menţiuni

Prezenta lucrare are la bază parte din activităţile şi rezultatele etapei I a proiectului PN 1937-0601 derulat la ICI Bucureşti (2019) în cadrul Programului naţional nucleu „SMARTIC” finanţat de Ministerul Cercetării şi Inovării.

BIBLIOGRAFIE

1. Arulkumaran, K., Deisenroth, M. P., Brundage, M., Bharath, A. A. (2017). *A Brief Survey of Deep Reinforcement Learning*. IEEE Signal Processing Magazine, special issue on Deep Learning for image understanding (ArXiv extended version – *arXiv:1708.05866v2* [cs.LG]).
2. Bellemare, M. G., Veness, J. & Bowling, M. (2012). *Investigating contingency awareness using Atari 2600 games*. In Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI-12), pp. 864–871. AAAI Press, Menlo Park, CA.
3. Bellemare, M. G., Naddaf, Y., Veness, J., Bowling, M. (2013). *The Arcade learning environment: An evaluation platform for general agents*. Journal of Artificial Intelligence Research, 47:253–279.
4. Bellman, R. (1952). *On the Theory of Dynamic Programming*. PNAS, 38(8):716–719.
5. Bellman, R. (1957). *Dynamic Programming*. Princeton Univ. Press, New York.

6. Cobo, L. C., Zang, P., Isbell, C. L., Thomaz, A. L. (2011). *Automatic state abstraction from demonstration*. In Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI-11), pp. 1243–1248. AAAI Press.

7. Diuk, C., Cohen, A., Littman, M. L. (2008). *An object-oriented representation for efficient reinforcement learning*. In Proceedings of the 25th International Conference on Machine Learning, pp. 240–247. ACM, New York.

8. Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., Abbeel, P. (2016). *RL2: Fast Reinforcement Learning via Slow Reinforcement Learning*. NIPS Workshop on Deep Reinforcement Learning.

9. Lake, B.M., Ullman, T. D., Tenenbaum, J. B., Gershman, S. J. (2016). *Building Machines That Learn and Think Like People*. The Behavioral and Brain Sciences, page 1.

10. Levine, S., Finn, C., Darrell, T., Abbeel, P. (2016). *End-to-End Training of Deep Visuomotor Policies*. JMLR, 17(39):1–40.

11. Levine, S., Pastor, P., Krizhevsky, A., Quillen, D. (2016). *Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection*. ISER.

12. Li, K., Malik, J. (2017). *Learning to Optimize*. International Conference on Learning Representations – ICLR (preprint in *arXiv:1606.01885, 2016*).

13. Lin, L.-J. (1992). *Self-improving reactive agents based on reinforcement learning, planning and teaching*. Machine Learning, 8(3-4):293–321.

14. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M. (2013). *Playing Atari with deep reinforcement learning*. *arXiv:1312.5602*.

15. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G. et al. (2015). *Human-Level Control through Deep Reinforcement Learning*. Nature, 518(7540):529–533.

16. Naddaf, Y. (2010). *Game-Independent AI Agents for Playing Atari 2600 Console Games*. Ph.D. Thesis, University of Alberta, Edmonton.

17. Rumelhart, D. E., Hinton, G. E.; Williams R. J. (1986). *Learning representations by back-propagating errors*. Nature 323, 533–536.

18. Schaul, T., Quan, J., Antonoglou, I., Silver, D. (2016). Prioritized experience replay. *ICLR* (ArXiv version – *arXiv: 1511.05952v4* [cs.LG]).

19. Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. et al. (2016). *Mastering the Game of Go with Deep Neural Networks and Tree Search*. Nature, 529(7587):484–489

20. Sutton, R. S., Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press (second edition, 2018).

21. Tesauro, G. (1992). *Practical issues in temporal difference learning*. Machine Learning, 8(3-4):257–277.

22. Tesauro, G. (1994). *TD-Gammon, a self-teaching backgammon program, achieves master-level play*. Neural Computation, 6(2):215–219.

23. Tesauro, G. (1995). *Temporal difference learning and TD-Gammon*. Communications of the ACM, 38(3):58–68.

24. Tesauro, G. (2002). *Programming backgammon using self-teaching neural nets*. Artificial Intelligence, 134(1-2):181–199.

25. Tesauro, G., Das, R., Chan, H., Kephart, J., Levine, D., Rawson, F., Lefurgy, C. (2008). *Managing Power Consumption and Performance of Computing Systems using Reinforcement Learning*. NIPS.

26. Tesauro, G., Gondek, D. C., Lechner, J., Fan, J., Prager, J. M. (2012). *Simulation, learning, and optimization techniques in Watson's game strategies*. IBM Journal of Research and Development, 56(3-4):16–1–16–11.

27. Tesauro, G., Gondek, D. C., Lenchner, J., Fan, J., Prager, J. M. (2013). *Analysis of Watson's strategies for playing Jeopardy!* Journal of Artificial Intelligence Research, 47:205–251.

28. Tieleman, T., Hinton, G. (2012). *Lecture 6.5–RMSProp*. COURSERA: Neural networks for machine learning 4.2:26–31.

29. Tsitsiklis, J., Roy, B. V. (1997). *An analysis of temporal-difference learning with function approximation*. IEEE Trans. Automat. Contr. 42, 674–690.

30. van Hasselt, H., Guez, A., Silver, D. (2016). *Deep Reinforcement Learning with Double Q-learning*. Proc. of AAAI (ArXiv version – *arXiv: arXiv:1509.0646*).

31. Vrejoiu, M. H. (2019). *Reţele neuronale convoluţionale, Big Data şi Deep Learning în analiza automată de imagini*. Revista Română de Informatică şi Automatică, 29(1):91-114.

32. Wang, J. X., Kurth-Nelson, Y., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., Botvinick, M. (2017). *Learning to Reinforcement Learn*. CogSci.

33. Wang, Z., Schaul, T., Hessel, M., Van Hasselet, H., Lanctot, M., de Freitas, N. (2016). *Dueling Network Architectures for Deep Reinforcement Learning*. *arXiv:1511.06581v3 [cs.LG]*.Watkins, C. J. C. H. (1989). Learning from Delayed Rewards. Ph.D. Thesis, University of Cambridge.

34. Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. Ph.D. Thesis, University of Cambridge.

35. Watkins, C. J. C. H., Dayan, P. (1992). *Q-learning*. Machine Learning, 8(3-4):279–292.

36. Zoph, B., V Le, Q. (2017). *Neural Architecture Search with Reinforcement Learning*. ICLR.

Mihnea Horia VREJOIU este cercetător ştiinţific gradul III în Departamentul "Sisteme inteligente distribuite intensive ca date" din ICI Bucureşti. Domeniile şi subiectele sale de expertiză şi interes cuprind: vedere artificială (prelucrare şi analiză de imagini, recunoaşterea formelor, recunoaşterea optică de caractere – OCR, recunoaşterea numerelor de înmatriculare – LPR) şi învăţare automată (clasificatoare, memorii asociative).

Mihnea Horia VREJOIU is scientific researcher 3rd degree in the "Distributed and Data Intensive Intelligent Systems" Department at ICI Bucharest. His main areas and topics of expertise and interest cover: Artificial Vision (Image Processing and Analysis, Pattern Recognition, Optical Character Recognition – OCR, License Plate Recognition – LPR) and Machine Learning (classifiers, associative memories).