Recurrent Reinforcement Learning

1st Richard Rusňák
Technická univerzita v Košiciach
Košice, Slovensko
richard.rusnak@student.tuke.sk

2nd Igor Savko
Technická univerzita v Košiciach
Košice, Slovensko
igor.savko@student.tuke.sk

3rd Martin Tovarňák
Technická univerzita v Košiciach
Košice, Slovensko
martin.tovarnak@student.tuke.sk

***Abstrakt*—Tento článok sa zameriava na matódu rekurentného reinforcement learningu. V jednotlivých kapitolách sú podrobne popísané základne princípy, ktoré súvisia s touto metódou. Poukážeme na jednoduchý reinforcement learning a jeho princípy a následne prejdeme na komplexnejší problém tejto metódy.**

***Kľúčové slová*—Reinforcement learning, Rekurentné neurónové siete, Markov rozhodovací proces**

I. Úvod

Reinforcement learning bol v posledných rokoch úspešne aplikovaný na viaceré typy úloh. V prvéj časti je spomenutý základný princíp reinforcement learningu a jeho model. Článok poukazuje aj na metódy, ktoré s ním úzko súvisia ako Markov rozhodovací model, pojem hodnotová funkcia a podstata dynamického programovania. Ukážeme si aj prístup Deep Reinforcement learningu, ktorý má uplatnenie v oblasti typu problému, ktorý si vyžaduje komplexnejšie riešenie, kde už základné metódy RL nestačia. Následne spomenieme rekurencie v neurónových sieťach a ich rôzne typy. Spomenieme architektúru týchto sieti a popíšeme najvýznamnejšie z nich. Podľa typov úloh sú rozdelené úlohy, kde sa využívajú spomenuté rekurencie.

Deep Q-Learning dosahuje obdivuhodné výsledky v hraní hier. Z dôvodu čoraz väčších výziev pri hraní 3D hier nastala potreba upravenia modelu tejto metódy, keďže tento model nie je vhodný pre čiastočne pozorovateľné prostredia. Dodanie rekurencie do tejto siete zabezpečuje integrovanie informácií naprieč snímkami.

II. Reinforcement learning

Pojem reinforcement learning (RL) je spojený s časťou umelej inteligencie. Tento termín je zaužívaný v anglickej literatúre, ktorý v preklade môžeme rozumieť ako "učenie sa zo skúseností". Ide o najprirodzenejší spôsob učenia.

Opiera o učenie, ktoré je charakterizované spojením medzi cieľom a skalárnym signálom, kde spätnou väzbou sa rozumie tzv. odmena, numerická hodnota, ktorá hovorí, aká vhodná bola akcia vykonaná v danom stave. RL je jednou zo základných paradigmov strojového učenia.

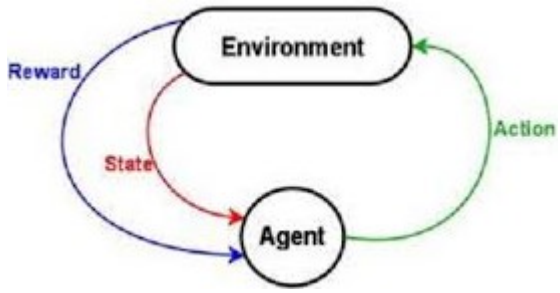
Je charakterizovaný spôsobom učenia, kde opakovaným skúšaním sa posilňujú tie akcie, ktoré vedú k získaniu určitej odmeny. Na druhej strane dochádza k oslabeniu tých akcií, ktoré predstavujú záporný signál.

Agent sa týmto prístupom snaží dosiahnuť cieľ v neistom, potenciálne komplexnom prostredí. Prístup, ktorý agent volí má spočiatku podobu pokusu a omylu, kým sa dopracuje k riešeniu danej problematiky. Agenta v danom prostredí môžu riadiť rôzne algoritmy. Je potrebné navrhnúť správny model, aby zistil, ako plniť úlohu, aby maximalizoval odmenu, počnúc úplne náhodnými skúškami a končiac sofistikovanou taktikou.

Využitím sily vyhľadávania a mnohých skúšok je v súčasnosti posilňovanie učenia jedným z najúčinnejších spôsobov, ako napomáhať "kreativite" stroju. Na rozdiel od ľudských bytostí môže umelá inteligencia získať skúsenosti z tisícok paralelných hier, ak je algoritmus výučby zosilnenia spustený na dostatočne výkonnej počítačovej infraštruktúre. [3]

III. Model reinforcement learning-u

Reinforcement learning nie je definovaný charakterizovaním učiacich metód, ale väčšinou charakterizovaním učiaceho problému.



Obr. 1. Základný model reinforcement learningu

Agent v čase t vníma v prostredí stav s, pričom vykoná akciu a, čím zmení svoj stav a hodnota tohto prechodu je agentovi prezentovaná cez odmenu r.

Cieľom je naučiť agenta optimálnu stratégiu (policy) $\pi : S \rightarrow A$, ktorá bude pre každý jeden stav vyberať takú akciu, ktorá bude viesť k zvýšeniu celkovej sumy posilňovacích signálov. Učenie tejto stratégie prebieha systémom pokus a omyl. [3]

IV. Markov rozhodovací model

Vyššie uvedený proces zaraďujeme do triedy postupných rozhodnutí, ktorý reprezentuje sériu stavov, v ktorých sa agent musí rozhodnúť, akú akciu je vhodné vykonať na základe súčasného stavu.

Markov rozhodovací proces (ďalej MDP) sa vyznačuje tým, že prechod do budúcich stavov je podmienený nezávisle od

minulých. Je to z dôvodu, že poznáme súčasný stav. Vhodnosť daného stavu určuje okamžitá odmena, ktorú agent obdrží. Cieľom je vybrať také akcie, ktoré maximalizujú dlhodobú odmenu. [3]

Prechod agenta z jedného stavu do druhého je charakterizovaný rozdelením pravdepodobnosti, čo odzrkadľuje neistotu vo výsledku rozhodnutia.

MDP je definovaný ako štvorica komponentov (S,A,R,T), kde:

- S je množina stavov,
- A je množina akcií,
- R je odmeňovacia funkcia,
- T je prechodová funkcia, ktorá vracia pravdepodobnostnú distribúciu cez všetky možné nasledujúce stavy.

Jedným z ďalších termínov, ktoré sa využívajú v súvislosti s MDP je policy (stratégia alebo politika), ktorá je označovaná ako $\pi(s)$, a predstavuje akciu, ktorú má agent v danom stave vykonať. Hovoríme o tzv. riešení MDP.

Ak je množina stavov MDP a množina akcií konečná, hovoríme o konečnom Markovom rozhodovacom procese. Konečné MDP sa väčšinou spájajú s teóriou reinforcement learning. [3]

V. Hodnotová funkcia

Princípom algoritmov reinforcement learningu je hľadanie hodnotovej funkcie (value function). Táto funkcia nám hovorí, nakoľko je správna akcia, ktorá bola vykonaná v danom stave. Správnosť vykonanej akcie udáva veľkosť odmeny, ktorú dostaneme, ak začíname v nejakom stave.

Veľkosť budúcej odmeny závisí od toho, aké akcie vykonávame, a preto sa hodnotová funkcia definuje vzhľadom na nejakú stratégiu π. [2]

Existujú dva typy hodnotovej funkcie:

- stavová,
- akciová.

Celková odmena je rovná postupnosti jednotlivých okamžitých odmien zo všetkých prejdených predchádzajúcich stavov (1):

$$R_t = r_{t+1} + r_{t+2} + ... + r_T \tag{1}$$

kde T je posledný časový krok. [2]

VI. Metódy RL

V tejto časti budú spomenuté niektoré zo základných algoritmov RL. Tieto algoritmy sú reprezentované vo forme numerických tabuliek, teda hodnotová funkcia je charakterizovaná ako pole hodnôt. To môže byť problém nepoužiteľnosti pri styku s väčšími a komplexnejšími problémami. V dôsledku toho sa často používajú ako základ pre ďalšie, komplexnejšie algoritmy. [3] [1]

A. TD learning

Temporal Difference Learning algoritmy sú RL algoritmy, ktoré iteratívne učia funkciu $V*(s)$ pomocou odmeny a dočasných rozdielov medzi po sebe idúcimi stavmi. Upravovanie $V(s)$ v smere $V(s')$ je vlastne spätná propagácia odmeny.

V tomto prípade sa používa za aktualizačné pravidlo nasledujúci vzťah (2): [3]

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{2}$$

kde:

- parameter α určuje rýchlosť učenia, ktorý používa väčšina učiacich metód v oblasti umelej inteligencie,
- parameter γ je diskontný faktor.

B. Q learning

Q-learning zaraďujeme medzi jeden z veľmi dôležitých pokrokov v teórií reinforcement learningu, s ktorým prišiel v roku 1989 Watkins. Jeho hlavnou úlohou je učiť sa funkciu $Q(s,a)$. Tento algoritmus má za úlohu aktualizovať v čase t hodnotu $Q(s_t, a_t)$ po tom, ako agent v prostredí vykoná akciu a_t, čo vedie k tomu, že zmení svoj stav z s_t na s_{t+1} a obdrží odmenu r_t.

Aktualizačným pravidlom pre tento typ algoritmu je platí vzťah (3): [3]

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{3}$$

Začínajúc s náhodnými hodnotami Q-funkcie, algoritmus iteratívne aplikuje aktualizačné pravidlo a za istých podmienok konverguje k optimálnej hodnote $Q*(s,a)$.

Následne sa dopracujeme k optimálnej stratégii $\pi*$, ktorá je charakterizovaná akciou s najvyššou hodnotou $Q*(s_t, a_t)$ (4):

$$\pi*(s) = argmax Q*(s_t, a_t) \tag{4}$$

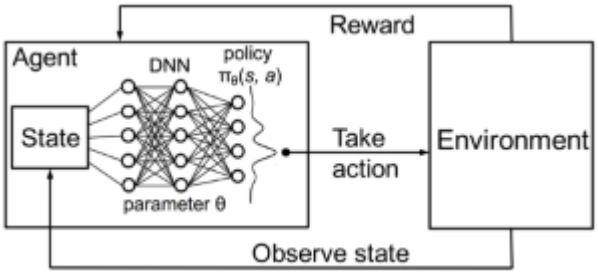
Tento typ algoritmu sa riadi logikou, aby bola Q-funkcia priebežne aktualizovaná pre všetky dvojice stav-akcia. Ak chceme dosiahnuť, aby algoritmus konvergoval k optimálnej Q-funkcii, je potrebné, aby agent v prostredí vykonával neustále objavovanie (exploration). [3]

VII. Deep reinforcement learning

V predošlej časti sme si opísali najznámejšie algoritmy reinforcement learningu. Spomenuli sme si Q-learning, ktorý produkuje Q-tabuľku, pomocou ktorej agent dokáže nájsť najvhodnejšiu akciu, ktorú má vykonať v určitom stave.

Avšak, v prípade komplexnejšieho a zložitejšieho problému môže nastať problém nepoužiteľnosti týchto numerických tabuliek. Môže dôjsť k situácii, kedy sa agent bude nachádzať v prostredí, ktoré bude obsahovať veľké množstvo stavov.

V tejto kapitole si spomenieme prístup Hlbokej Q neurónovej siete (Deep Q Neural Network), ktorá pri väčšom počte stavov zoberie stav, a následne aproximuje hodnoty Q pre každú akciu založenú v tomto stave. Neurónové siete sú



Obr. 2. Základná schéma Deep Q-Learning

agentom, ktorý sa učí mapovať stavové dvojice na odmeny. Podobne ako všetky neurónové siete, používajú koeficienty na priblíženie funkcie týkajúcej sa vstupov k výstupom a ich učenie spočíva v nájdení správnych koeficientov a váh.

Princíp Hlbokej Q neurónovej siete je často uplatnený v oblasti hier. Vo väčšine prípadov býva na vstupe nejaký počet snímok, ktoré prechádzajú cez sieť, a vydávajú vektor Q-hodnôt pre každú možnú akciu v danom stave. Je potrebné nájsť najväčšiu hodnotu tohto vektora, aby sme našli našu najlepšiu akciu.

Spočiatku sú akcie agenta zlé, robí chyby. Postupom času však začne spájať stavy s najlepšími krokmi. Snímky sú zväčša spracované konvolučnými vrstvami. Tieto vrstvy umožňujú využívať priestorové vzťahy v obrazoch ako aj dokážu rozpoznať stav agenta na obrazovke. To znamená, že vykonávajú svoju typickú úlohu rozpoznávania obrazu.

VIII. Rekurentné neurónové siete

A. Popis

Rekurzia v neurónových sieťach je dnes veľmi populárna a považuje sa za jednu zo základných vlastností hlbokého učenia. Rekurentné neurónové siete sú pomerne staré, vytvorené boli v 80-tych rokoch minulého storočia, no svoj potenciál ukazujú až dnes, vďaka nárastu výpočtovej sily a obrovskému množstvu údajov. Tieto siete sú charakteristické vďaka šíreniu so spätnou väzbou a sú zovšeobecnením dopredných sieti. V ich štruktúre sa nachádzajú prepojenia medzi neurónmi v oboch smeroch a sú dovolené prepojenia jednotlivého neurónu samého so sebou, vďaka čomu sa komplikuje návrh vhodného algoritmu. Narozdiel od dopredných sieti, rekurentné si vyžadujú väčšie časové a pamäťové nároky. Tieto siete sú prispôsobené na lepšie uchovávanie svojich predošlých vlastností, tým pádom prostredníctvom spätného šírenia signálu sa dokážu adaptovať na zložitejšie úlohy. Tým, že si tieto siete pamätajú vstupné dáta, dokážu predpovedať následujúce stavy a tým pádom lepšie pochopiť konkrétnú úlohu. Vďaka tomu, sú využívané pri rozpoznávaní reči, textu, zvuku alebo videa. Takisto ich dnes využíva napríklad Siri od Applu alebo Voice Search od Google. [8] [9]

B. Rozdelenie typov rekurentných sieti

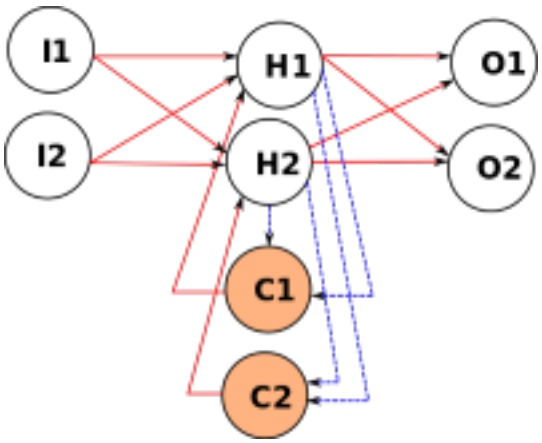
- **Plne rekurentné** sú najvšeobecnejším typom rekurentných sieti, kde sú dovolené vzájomné prepojenia všetkých neurónov a taktiež prepojenia neurónov samých so sebou. [8]
- **Čiastočne rekurentné**, kde prevláda počet dopredných, ale aj menší počet späných prepojení. Tieto spätné prepojenia sú navrhované tak, aby sa nenarušili zachované vlastnosti typické pre dopredné siete. Tento typ je považovaný za podmnožinu plne rekurentných sieti, kde chýbajúce späné prepojenia sú nahradené prepojeniami s nulovou váhou, tým pádom algoritmy vhodné pre plne rekurentné siete sú použiteľné aj pri čiastočne rekurentných sietiach. [8]
- **Siete so symetrickými prepojeniami**, kde každá dvojica neurónov má vždy rovnakú váhu, aj v prípade, že ide o opačné prepojenie. [8]
- **Siete s nesymetrickými prepojeniami** sú opakom sieti so symetrickými prepojeniami, čiže nie je splnená podmienka symetrie, tým pádom prepojenia medzi neurónmi môžu mať rôzne váhy. V tomto prípade nieje zaručený stabilný stav (stav kedy sa sieť nemení), pričom siete splňajúce podmienku symetrie, vždy spomenutý stav dosiahnu. [8]
- **Siete s diskrétnym vstupom**, kde vstupné hodnoty sú nezávislé na čase. V každom časovom kroku príchádza na vstup siete jedna množina vstupných hodnôt, ktorá pri prechode do ďalšieho kroku je nahradená novou množinou hodnôt. [8]
- **Siete so spojitým vstupom**, kde signál v tomto type predstavuje spojitú veličinu, ktorá je funkciou času a mení sa počas práce siete. V tomto prípade je dôležitý spôsob, ako je reprezentovaný čas a na ňom závislé veličiny. No tento typ si ajtak našiel svoje uplatnenie, najmä v robotike pri rozpoznávaní reči. [8]

C. Architektúra rekurentných sieti

V minulosti vzniklo niekoľko modelov rekurentných sieti, ktoré su viacvrstvové s rekurentnými neurónmi. Ako príklad možeme ukázať trénovaciu množinu, ktorá by v sebe zahŕňala tieto dvojice: a$\rightarrow\alpha$, b$\rightarrow\beta$, b$\rightarrow\alpha$, b$\rightarrow$, c$\rightarrow\alpha$, c$\rightarrow$, d$\rightarrow\alpha$, kde prvé znaky z dvojice znamenajú vstupné vektory a druhé požadované výstupné vektory. K jednému vstupu môže prislúchať viacero výstupov v závislosti na časovom kontexte. To znamená, že výstup zo siete nie je len výsledkom vstupu, ale aj minulých stavov. Najjednoduchším príkladom je **neurónová sieť s oknom do minulosti**. (Time Delay Neural Network, ďalej TDNN), kde okno do minulosti ma konečnú nemennú dĺžku, ozn. D. Táto sieť viacvrstvovým dopredným sieťam umožňuje okrem momentálnych vstupov v čase t, uchovať informácie aj z vstupov z predošlých D krokov v časoch *t-1, t-2...t-D*. V tomto prípade je možné trénovať sieť pomocou metódy spätného šírenia chýb, kde je veľmi dôležité zachovať poradie trénovacích vzoriek v trenovacej množine. Nedostatkom tejto metódy je, že na základe trénovacej množiny nie je jednoduché odhadnút dĺžku okna do minulosti (D). K tomu, aby sme zlepšili výsledky spracovania časových postupností dát s použitím okna do minulosti a jeho nestálou konečnou dlžkou, bola zavedená nová architektúra

- rekurentné neurónové siete (RNN). Pokiaľ nastáva prípad, kedy TDNN nestačí na zaznamenanie časovej štruktúry dát sa postupnosti generujú **automatmi**. Toto môže byť omnoho úspornejšie ako reprezentácia časoveho kontextu pomocou spomínaného okna do minulosti. Nastávajú prípady, kedy je potrebné neobmedzene dlhé okno do minulosti, kde by bola architektúra siete TDNN nepostačujúca. Medzi najznámejšie RNN patria Elmanová, Jordanová, Bengiová a Hopfieldová. [10]

- **Elmanová sieť** Je to sieť s tromi vrstvami (Obr. 3) l,H,o s pridaním kontextových jednotiek (Obr. 3) C. Skrytá vrstva (H), je spojena s kontextovými jednotkami (C) s váhou 1. V každom časovom kroku sa uloží kópia predchádzajúcich stavov v týchto pridaných kontextových jednotkách. To umožňuje sieti udržiavať určitý stav a vykonávať predikčné úlohy ako je sekvenčná predikcia. [11]



Obr. 3. Elmanová sieť.

- **Jordanová sieť** Táto sieť je podobná predošlej Elmanovéj, ale kontextové jednotky sú spojené z výstupnou vrstvou namiesto skrytej.
- **Bengiová sieť** Je to kombinácia predošlých sieti. Kontextové jednotky sú spojené so skrytou vrstvou ale aj s výstupnou.

D. Typy úloh s rekurentnými neuronovými sieťami

- **Typ 1**, kde rekurentná neuronová sieť rozhoduje, či ukončená postupnosť vstupov patrí, alebo nepatrí do určitéj triedy, poprípade do ktoréj z možných tried patrí s určitou pravdepodobnosťou. Ako príklad tohto typu môžme určiť klasifikáciu postupností znakov z konečnej abecedy. Požadovaný výstup by boli konktrétne znaky. Ďalší priklad môže byť, či postupnosť určitých signálov v zariadení vedie, alebo nevedie k jeho poruche, pri najlepšiom k akéj konkrétnej poruche. [10]
- **Typ 2**, kde úlohou je z časových štruktúr dát pred určitým časom t predpovedať nasledujúce dáta po čase t. Na trénovanie predikcie týchto údajov je potrebný velký počet dát. [10]
- **Typ 3** je podobný ako predchádzajúci typ, len je určený pre zložitejšie predikčné úlohy. V tomto type nejde len o predikovanie dát v nasledujúcom čase, ale podľa sledovania úseku vývoja dát sa pokračuje v časovom rade dát. Ako príklad možeme pozorovať úsek dát: *23123123123123123*, kde pokračovanie by bolo *123123* a tak ďalej. V realnom svete však takýto úsek dát je omnoho zložitejší. Predikcia budúcich dát v časovom úseku sa realizuje, keď po dovršení úseku v čase *t*, sa generujú predikované hodnoty v časovom úseku *t+1*. Tieto predikované hodnoty sa priradia k pôvodným dátam a následne sa z doplnených dát generujú nové predikované dáta v časovom úseku *t+2*. Táto postupnosť pokračuje ku žiadaným hodnotam. [10]

IX. Long Short Term Memory Neurónová sieť

Rekurentná sieť založená na bloku LSTM rieši problém zapamätania si kontextu a zároveň rieši problém efektívneho trénovania dlhých sekvencií. Na blok LSTM sa môžeme pozerať ako na špeciálny typ neurónu, ktorý závadza dodatočné regulujúce vstupy označovaných ako Gate. Tento špeciálny typ neuónu má 4 vrstvy, ktoré fungujú inak, ako u klasických rekurentných sietí. Základom v tejto jednotke je stav siete, vodorovná čiara, ktorá prenáša informácie z predchádzajúcej jednoty $C_{(t-1)}$ na novú informáciu $C_{(t)}$. Prvá sigmoidná vrstva *"forget gate"* má na vstupe vstup X_t a výstup z predchádzajúcej jednotky LSTM $h_{(t-1)}$ a stará sa o rozhodnutie, ktorú časť zo znalosti posledného výstupu odstrániť. [4]

$$f_(t) = \sigma(W_f * [h_{t-1}, x_t] + b_f) \tag{5}$$

V nasledujúcom kroku je potrebné rozhodnúť, aké nové informácie uložíme do stavu jednotky, čo zahŕňa 2 časti. Po prvé, sigmoidná vrstva nazývaná *"inputgate"* určuje, ktoré hodnoty budeme aktualizovať. Vrstva *tanh* potom vytvorí vektor nových kandidátskych hodnôt $\tilde{C}_t$, ktorý by mohol byť pridaný do stavu. Následne je aktualizovaný stav $C_{(t)}$. [4]

$$i_(t) = \sigma(W_f * [h_{t-1}, x_t] + b_i) \tag{6}$$

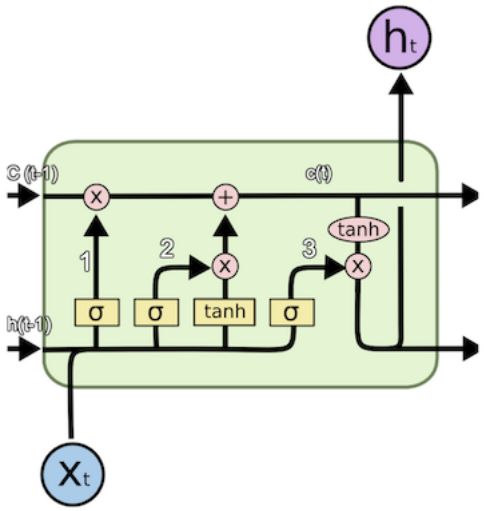
$$\tilde{C}_t = tanh(W_C * [h_{t-1}, x_t] + b_c) \tag{7}$$

$$C_t = f_{(t)} * C_{(t-1)} + i_{(t)} * \tilde{C}_t \tag{8}$$

Nakoniec sa musíme rozhodnúť, ktoré časti zo stavu jednotky budú na výstupe. O tomto nám rozhodne sigmoidná vrsta. Vynásobením vystupu tejto sigmoidnej vrstvy a vrstvy $tanh(C_t)$ dostaneme nami želaný výstup z jednotky. [4]

$$o_(t) = \sigma(W_o * [h_{t-1}, x_t] + b_o) \tag{9}$$

$$h_t = o_(t) * tanh(C_t) \tag{10}$$



Obr. 4. Jednotka LSTM siete. [4]

X. Recurrent Neural Reinforcement Learning

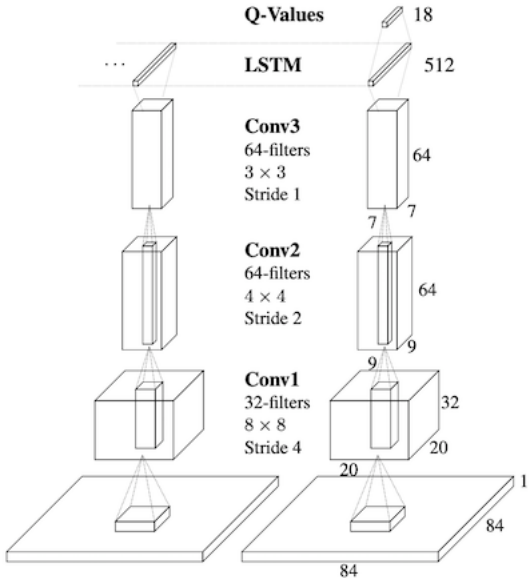
Ako uz bolo spomenuté vyššie, Deep Q-Network(DQN) ukázala skvelé výsledky, ale hlavnou myšlienkou jeho architektúry je predpoklad, že prijaté pozorovanie obsahuje všetky informácie o aktuálnom stave prostredia. V kontexte 3D videohier, agent vidí len zlomok celého prostredia. V [5] tento problém obišiel využítím posledných štyroch snímok obrazu ako vstupu do ich DQN a umožnil tak modelu prístup k viacerým informáciám, než len aktuálneho pozorovania. Toto riešenie funguje pre krátke potreby závislostí, ale nefunguje v prostredí, kde agent si musí pamätať staršie informácie. Na riešenie takýchto prostredí v [6] upravili architektúru DQN pridaním rekurentnej vrstvy medzi konvolučné a dopredné vrstvy. Ich model, nazvaný Deep Recurrent Q-Network (DRQN), teraz obsahuje $Q(o_t, a_t, h_t)$ namiesto $Q(s_t, a_t)$, kde o_t je aktuálne pozorovanie, narozdiel od s_t, ktorý bol považovaný za aktuálny stav a h_{t-1} je skrytý stav agenta v predchádzajúcom kroku. Využili Long short-term memory(LSTM) neuróny v rekurentnej vrstve s $h_t = LSTM(o_t, h_{t-1})$ a teda odhad $Q(h_t, a_t)$.

A. POMPD

Úlohy v reálnom svete často obsahujú neúplné informácie o stave vyplývajúce z čiastočnej pozorovateľnosti prostredia. Ak sa pozrieme na problém väčšiny hier Atari, napríklad hra Pong. Aktuálna obrazovka zobrazuje iba polohu "raketyä lopty, nie však rýchlosť a smer pohybu lopty. Poznanie smeru pohybu lopty je rozhodujúcim prvkom pre určenie najlepšieho umietnenia "rakety". Z tohto dôvodu sa hra stáva Čiastočne-pozorovateľným Markovským rozhodovacím procesom(Partially-Observable Markov Decision Process (POMDP)). POMDP lepšie zachytáva dynamiku prostredí reálneho sveta tým, že uznáva, že pocity, ktoré agent získal, sú len čiastočnými pohľadmi na základný stav systému. Formálne môže byť POMDP opísaný ako 6 násobok (S, A, P, R, Ω, O), kde S, A, P, R sú stavy, akcie, prechody a odmeny. Agent už nieje zasvätený skutočnému stavu systému a namiesto toho prijíma pozorovanie $o \in \Omega$. Toto pozorovanie sa generuje zo základného systému podľa rozdelenia pravdepodobnosti $o \sim O(s)$. Experimenty ukazujú, že pridanie rekurentnosti do Deep QLearningu umožňuje sieti lepšie ohdadnúť stav systému, teda dokážu lepšie priblížiť skutočné hodnoty Q zo sledov pozorovaní [6].

B. Deep Recurent Q-Network(DRQN) Atari 2600

Obr.5 znázorňuje DRQN, ktorá bola využitá v [6]. Aby boli izolované efekty rekurentnosti, bola minimálne zmodifikovaná DQN, pričom bola nahradená iba prvá plne prepojená vrstva rekurerentnou LSTM rovnakej veľkosti. Obraz je spracovaný tromi konvolučnými vrstvami a výstupy sú privádzané do plne prepojenej vrstvy LSTM. Nakoniec posledná vrstva vydáva Q hodnotu pre každú akciu. Počas učenia sa parametre pre konvolučné a rekurentné vrstvy siete naučia spoločne od nuly [6].



Obr. 5. DRQN využitá v hrách Atari [6].

Aktuálizácia rekurentnej a konvolučných vrstiev vyžaduje aby každý spätný chod obsahoval mnoho časových krokov obrazoviek z hier a cieľové hodnoty. Počiatočný skrytý stav LSTM môže byť nulovaný alebo prenesený z predchádzajúcich hodnôť. V tomto prípade sú uvažované dva typy aktualizácií [6]:

- ***Sekvenčné aktualizácie***: Epizódy sú vyberané z replay pamäte náhodne, aktualizácie začínajú na začiatku epizódy a pokračujú v čase až do konca danej epizódy. Ciele sú generované v každom časovom úseku z cieľovej Q hodnoty a skrytý stav rekurencie sa prenáša v celej epizóde [6].
- ***Náhodné aktualizácie***: V tomto prípade boli využité Náhodné aktualizácie siete. Epizódy sú vyberané z replay pamäte náhodne, ale aktualizácie začínajú v náhodných bodoch v epizóde. Počiatočný stav rekurencie sa na

začiatku aktualizácie nuluje. Ciele sú generované z Q hodnôt. Náhodné aktualizácie lepšie dodržiavajú politiku náhodného vzorkovania, ale z tohto dôvodu musi byť skrytý stav LSTM na začiatku každej aktualizácie nulovaný [6].

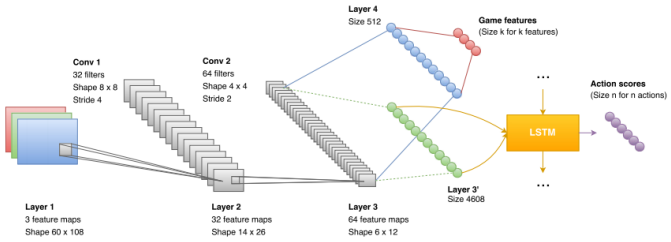
Výsledným efektom využitia modifikovanej DQL kombináciou LSTM je taký, že aj napriek tomu, že v každom časovom okamihu sieť vidí len jeden snímok, je schopná integrovať informácie naprieč snímkami a tak zistiť relevantné informácie ako napríklad rýchlosť objektov na obrazovke. Analýza taktiež ukázala, že DRQN natrenovaná s čiastočnými pozorovaniami, môže zovšeobecniť svoju politiku na prípad kompletných pozorovaní. Rekurentná sieť sa naučí politiky, ktoré sú dosť robustné na to, aby zvládali problém s vynechanými snímkami obrazovky a taktiež aby sa zväčšil výkon, keď bude k dispozícii viac údajov [6].

C. 3D videoha DOOM

V [7] využili DRQL algoritmus, ktorý bol využívaný pri hraní 3D videohry DOOM. Jadro systému je postavené na architektúre DRQN. Keďže ako už bolo spomínané, jednoduchá Deep Q Sieť, ktorá je využívaná pri hraní 2D hier, očakáva úplné alebo skoro úplné pozorovanie prostredia v každom kroku. V hre ako DOOM, kde zorné pole agenta je obmedzené na 90 stupňov okolo jeho pozície, vidí agent len zlomok celého prostredia.

V deathmatch hre ako je DOOM, musíte preskúmať mapu, zbierať predmety a bojovať proti nepriateľom. Ide o komplexný problém, ktorý nieje jednoduché vyriešiť. V tomto článku sa rozhodli využiť dve siete(Obr.6), ktorých účelom bolo [7]:

- Navigačná fáza - navigácia.
- Akčná fáza - Bojovať proti nepriateľovi.



Obr. 6. DDRQN využitá v hre DOOM [7].

Aktuálna fáza hry sa určuje predpovedaním, či je(akčná fáza) alebo nieje(navigačná fáza) v aktuálnom časovom okamihu viditeľný nepriateľ. Existujú rôzne výhody rozdelenia úloh do dvoch fáz a učenia inej siete na inú fázu. Takéto riešenie robí architektúru modulárnou a umožňuje, aby boli rôzne modely naučené a testované nezávisle pre každú fázu. Obidve siete môžu byť trénované paralelne, čo robí trénovanie oveľa rýchlejšie v porovnaní s trénovaním jednej siete pre celú úlohu. Naviac navigačná fáza vyžaduje iba 3 akcie [7]:

- Pohyb dopredu.
- Otočenie doľava.
- Otočenie doprava.

Ešte dôležitejším faktorom takejto implementácie je, že sa zmierňuje vedľajší efekt čakať na nepriateľa ak by sa trénovala sieť iba na akčnú fázu.

D. Trénovanie

Spoločné trénovanie modelu DRQN a detekcie herných funkcií umožňuje jadrám konvolučných vrstiev zachytiť relevantné informácie o hre len s niekoľkými hodinami trénovania potreblého na dosiahnutie optimálnej presnosti detekcie neprieteľa na 90 %. Funkcia odmeňovania pre akčnú sieť zahŕňa [7]:

- Pozitívne odmeny za zabitia.
- Negatívne odmeny za samovraždy.
- Pozitívne odmeny za vyzdvihnutie predmetov.
- Negatívne odmeny za stratu zdravia.
- Negatívne odmeny za streľbu alebo stratu munície.

Navigačná sieť dostala jednoducho pozitívnu odmenu za vyzdvihnutie veci a negatívnu odmenu za chôdzu po láve. Ako najlepšia rovnováha medzi rýchlosťou trénovania a výkonom sa ukázal fakt, že je najefektívnejšie agentovi dávať na vstup každý piaty snímok obrazu [7].

Takto natrénovaná sieť dokázala mať pri hraní hry lepšie výsledky ako človek v scenári jedného hráča, ktorý bojoval proti 10. botom, ale aj v scenári multiplayerovej hry, kde agent hral proti reálnym hráčom.

ZÁVER

V tomto článku sme si predstavili Reinforcement Learning, jeho metódy a Rekurentné siete. Keďže v reálnom svete sa často objavujú neúplné informácie o stave a prostredí, bolo potrebné upriamiť pozornosť na čiastočnú pozorovateľnosť prostredia ktorá bola využitá pri Deep Recurrent Q-Network a je modifikáciou Deep Q-Network. DRQN dokázali integrovať informácie naprieč snímkami vďaka zavedenej rekurentnej LSTM vrste, aj napriek tomu, že v každom časovom okamihu bol ako vstup siete priradený jeden snímok obrazu.

Taktiež sme si ukázali zaujímavú architektúru siete, ktorá bola využitá pri 3D hre, kde bolo potrebné modifikovať architektúru siete tak, aby zahŕňala nezávislé siete zodpovedné za rôzne fázy. Tento model ukáxsl zaujímavé výsledky, keďže dokázal prekonať výsledky samotného človeka.

LITERATÚRA

[1] Loi Do, "Reinforcement learning pro řízení dynamických systémů", ČVUT, FEI, Katedra řídicí techniky, 2017.

[2] P. Y. Glorenne, "Reinforcement Learning: an Overview", IRISA, Aachen, Germany, 2000.

[3] P. Jurčo, "REINFORCEMENT LEARNING V ROBOTIKE", Univerzita Komenského v Bratislave, Fakulta matematiky, fyziky a informatiky, 2011.

[4] C. Olah, Ünderstanding LSTM Networks.", Colah.github.io, 2015. [Online] [http://colah.github.io/posts/2015-08-Understanding-LSTMs/]

[5] V. Mnih a kol., "Human-level control through deep reinforcement learning.", Nature, 2015.

[6] M. Hausknecht a P. Stone, "Deep Recurrent Q-Learning for Partially Observable MPSs", 2015 AAAI Fall Symposium Series, 2015.

[7] G. Lample a D. S. Chaplot,"Playing FPS Games with Deep Reinforcement Learning", Thirty-First AAAI Conference on Artificial Intelligence, 2017.
[8] R. Novotný, "Rekurentné siete", UPJS, 2002.
[9] Recurrent Neural Network and LSTM, 2018, [Online] [https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5].
[10] Návrat P., Bieliková M., Beňušková Ľ., Kapustník I. Unger M., Ümelá inteligencia", STU, Bratislava, 2002.
[11] Yuan-Chu Cheng, Wei-Min Qi Wei-You Cai, "Dynamic properies of Elman and modified Elman neural network", IEEE, Beijing, China, 2002.