Žilinská Univerzita v Žiline

Fakulta Riadenia a Informatiky

Učenie s posilňovaním s využitím hlbokých neurónových sietí

Dizertačná práca

28360020233003

Študijný program: Aplikovaná informatika

Študijný odbor: Informatika

Pracovisko: Katedra matematických metód a operačnej analýzy
Fakulta riadenia a informatiky, Žilinská univerzita v Žiline

Školiteľ: doc. Mgr. Ondrej Šuch, PhD.

Školiteľ špecialista: Ing. Peter Tarábek, PhD.

Žilina, 2023 **Ing. Marek Baláž**

Čestné výhlásenie

Čestne prehlasujem, že dizertačnú prácu som vypracoval sám, podľa smerníc a metodických usmernení univerzity, ako aj podľa odborných rád mojich školiteľov. Zdroje, z ktorých som čerpal, sú náležite citované a uvedené v zozname použitej literatúry. Vo výskume som aktívne využíval nadobudnuté znalosti a skúsenosti počas doktorandského štúdia.

podpis: _____

Abstrakt

BALÁŽ, Marek: *Učenie s posilňovaním s využitím hlbokých neurónových sietí* [Dizertačná práca] – Žilinská univerzita v Žiline. Fakulta riadenia a informatiky, Katedra matematických metód a operačnej analýzy. – Školiteľ: doc. Mgr. Ondrej Such, PhD. - Žilina: FRI ŽU, 2023. 107 strán

Oblasť učenia posilňovaním zažíva za posledných desať rokov významný rozmach. Väčšina algoritmov z tejto oblasti využíva jeden alebo viac predikčných modelov tvoriacich funkciu politiky. V dizertačnej práci sa zaoberáme skupinou modelovo založených algoritmov, ktoré využívajú v rozhodovacom procese model prostredia. Výskumná časť práce je rozdelená na tri časti. V prvej časti výskumu sa venujeme modifikácii rozhodovacieho procesu, resp. Monte Carlo prehľadávacieho stromu tak, aby bol silnejší agent schopný prispôsobiť svoju stratégiu slabšiemu oponentovi. V druhej časti výskumu sa zaoberáme návrhom a implementáciou algoritmu Monte Carlo prehľadávací strom pomocou tenzorových operácií. Takto implementovaný algoritmus dokáže výrazne zrýchliť svoj čas vykonávania a tak predbehnúť aj doposiaľ používané implementácie. V poslednej časti práce sa venujeme modifikácii algoritmu MuZero, ktorému pridávame schopnosť vnútornej motivácie ako podporu prieskumu stavového priestoru pomocou modulu Random Network Distillation.

Kľúčové slová: učenie posilňovaním, strojové učenie, hlboké neurónové siete, modelovo založené algoritmy, MuZero, Monte Carlo rozhodovací strom

Abstract

BALÁŽ, Marek: *Reinforcement learning with deep neural networks* [Dissertation thesis]– University of Žilina. Faculty of Management Science and Informatics; Department of Mathematical Methods and Operation Research. – Supervisor: doc. Mgr. Ondrej Šuch, PhD. - Žilina: FRI ŽU, 2023. 107 pages

The field of reinforcement learning has been growing for the past ten years. Most significant algorithms use one or more prediction models as policy function. In the dissertation thesis, we deal with a group of model algorithms that use the environment model in the decision-making process. In the first part, we modify the decision-making process (Monte Carlo tree search), so that a stronger agent is able to adapt its strategy to a weaker opponent. The second part is oriented to the design and implementation of the algorithm Monte Carlo tree search by using tensor operations. This implementation beats popular implementations that used multiprocessing. The last part is oriented to the modification of algorithm MuZero that is extended by intern motivation (intern motivation is handled by the Random Network Distillation module).

Keywords: reinforcement learning, machine learning, deep neural networks, model-based algorithms, MuZero, Monte Carlo tree search

Obsah

Zoznam skratiek

NN	Neurónová sieť, resp. neurónové siete (Neural networks)
MSE	Priemerná štvorcová chyba (Mean square error)
CNN	Konvolučné neurónové siete (Convolutional neural networks)
RNN	Rekurentné neurónové siete (Recurrent neural network)
RL	Učenie posilňovaním (Reinforcement learning)
MDP	Markovov rozhodovací proces (Markov decision process)
DP	Dynamické programovanie (Dynamic programming)
TD	Časové rozdiely (Temporal difference)
DQN	Hlboká Q sieť (Deep Q network)
DDQN	Double deep Q network
A2C	Advantage actor-critic
A3C	Asynchronous advantage actor-critic
PPO	Proximal policy optimization
TRPO	Trust Region Policy Optimization
AZ	AlphaZero
MCTS	Monte Carlo prehľadávací strom (Monte Carlo tree search)
PUCT	Predictor + Upper Confidence bounds applied to Trees
MZ	MuZero
ICM	Intrinsic Curiosity Module
I2A	Imagination-Augmented Agent
RND	Random Network Distillation
CPU	Centrálna procesorová jednotka
GPU	Grafická procesorová jednotka
ID	Identifikačné číslo

Zoznam obrázkov

Zoznam algoritmov

Úvod

Metódy strojového učenia sú čoraz viac využívané pre riešenie každodenných úloh, ako napríklad triedenie užívateľov internetových stránok podľa spoločných vlastností, predpoveď počasia, detekcia a spracovanie obrazu, rozpoznávanie reči, autonómne vozidlá a pod. Rozmach strojového učenia, resp. hlbokého učenia je zapríčinený predovšetkým rastúcou výpočtovou silou hardvéru, zvyšovaním kapacít pamäťových zariadení a možnosťami efektívneho získavania dát.

V prvých dvoch kapitolách sa zaoberáme objasnením základných pojmov a prístupov strojového učenia, neurónových sietí a oblasti učenia posilňovaním. Druhá kapitola je obzvlášť dôležitá z hľadiska porozumenia komplexnejším algoritmom učenia posilňovaním.

V tretej kapitole popisujeme algoritmy bez modelu prostredia. Keďže rôzne vedecké články využívajú rozličné značenie rovnakých veličín, resp. premenných, tak sme sa rozhodli začiatok kapitoly venovať nami zvolenému značeniu, ktoré používame vo zvyšku práce.

Štvrtá kapitola zakončuje opis analýzy súčasného stavu prostredníctvom opisu modelovo založených algoritmov učenia posilňovaním. V kapitole sú popísané významné algoritmy, ktoré výrazne ovplyvnili náš výskum.

V piatej kapitole popisujeme ciele dizertačnej práce, ktoré sú definované na základe vykonanej analýzy predchádzajúcej kapitoly.

V šiestej kapitole sa zaoberáme adaptívnosťou naučeného algoritmu v hre proti slabšiemu súperovi. Dôkladne opisujeme nami navrhnuté metódy, ktoré znižujú výkonnosť naučeného algoritmu vzhľadom na súperovu úroveň. V kapitole sa taktiež nachádza popis experimentov, použitých modifikácií z dôvodu zefektívnenia tréningu a dosiahnuté výsledky.

Siedma kapitola sa venuje implementácií heuristiky Monte Carlo Tree Search, ktorá

tvorí významnú časť niektorých modelovo založených algoritmov. Navrhnutá implementácia je založená na princípe vektorových, resp. tenzorových operácií, čo umožňuje paralelné vykonávanie inštrukcií na grafickej karte. V časti experimentov porovnávame implementáciu s tromi ďalšími prístupmi.

Ôsma kapitola sa venuje návrhu a testovaniu modelovo založeného algoritmu učenia posilňovaním (MuZero), ktorý využíva model prostredia pre podporu rozhodovania a zároveň pre podporu prieskumu stavového priestoru.

Kapitola 1

Strojové učenie

Metódy strojového učenia je možné chápať ako skupinu učiacich sa algoritmov, ktorých cieľom je hľadanie špecifických vzorov v dátach. Učiaci algoritmus zvyčajne obsahuje množinu učiacich sa parametrov, ktorých hodnoty sa snaží naučiť vzhľadom na množinu dostupných dát (datasetu) a kritérium konkrétnej úlohy. Vzhľadom na kritérium úlohy je potrebné zvoliť vhodný typ učiaceho sa algoritmu.

V predikcií medzi najznámejšie typy úloh patrí úloha regresie a úloha klasifikácie. V úlohe regresie sa algoritmus snaží predikovať požadovanú numerickú hodnotu (napr. výpočet hodnoty nehnuteľnosti). V úlohe klasifikácie je zvyčajne potrebné predikovať požadovanú triedu z množiny výstupných možností (napr. klasifikácia objektu z obrazu).

Samotný proces učenia (nazývaný aj tréning) je ovplyvnený množinou dát, ich výberom a vzťahom voči kritériu úlohy. Medzi tri základné prístupy učenia algoritmov patrí učenie bez učiteľa, učenie s učiteľom a učenie posilňovaním.

Učenie bez učiteľa - Algoritmy s učením bez učiteľa sa vyznačujú predovšetkým neoznačenými dátami, t.j. ku dátam nie sú priradené požadované hodnoty. Algoritmy sa snažia nájsť, resp. vhodne interpretovať vnútornú štruktúru dát. Príkladom algoritmov s učením bez učiteľa je metóda zhlukovej analýzy. Tá rozdeľuje dataset do tzv. zhlukov. V jednotlivých zhlukoch sa nachádzajú vzájomne si podobné dáta. Metóda sa používa napr. pri triedení zákazníkov podľa spoločných vlastností.

Učenie s učiteľom - Algoritmy s učiteľom využívajú označené dáta, t.j. každá entita v datasete má presne určenú požadovanú hodnotu, resp. triedu. Označenie dátovej entity sa nazýva aj požadovaný výstup. Cieľom prediktívneho modelu je naučiť sa správne predikovať požadované výstupy vzhľadom na prichádzajúce vstupy. V

porovnaní s predchádzajúcim prístupom, v učení s učiteľom vzniká potreba označiť všetky entity v datasete. Medzi algoritmy využívajúce učenie s učiteľom patrí napr. regresná analýza, analýza podporných vektorov a rôzne spôsoby učenia sa neurónových sietí.

Učenie posilňovaním - Učenie algoritmov (nazývaných aj agenti) prebieha na základe odmien a trestov. Dataset nemusí byť pevne daný, pretože dáta sú väčšinou zbierané z prostredia (simulátora), v ktorom sa agent učí vykonávať správne rozhodnutia. Výhodou prístupu učenia posilňovaním je, že agent sa môže naučiť vykonávať lepšie rozhodnutia ako človek. Dôkazom takého tvrdenia je dosiahnutie nadľudských výkonov v doméne Atari hier [1], prehra svetového šampióna v hre GO [2], alebo nedávne úspechy v komplexných hrách akými sú Dota 2 [3] a Starcraft 2 [4].

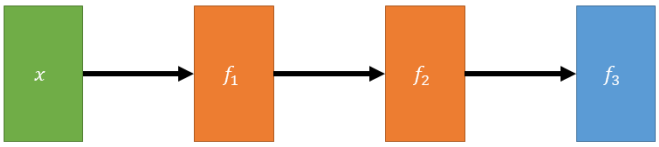
Z úspechov učenia posilňovaním je zrejmé, že výskum sa z veľkej časti realizuje na čoraz komplexnejších hrách. V tomto prípade je možné chápať hru ako vopred vytvorený zložitý simulátor, ktorý má pevne definované pravidlá (povolené ťahy) a cieľ (skóre, výhra / prehra). Ku hrám taktiež existuje veľké množstvo záznamov z majstrovských partií, ktoré môžu byť užitočné v procese vyhodnocovania dosiahnutých riešení.

Napriek tomu, že testovanie nových algoritmov sa vykonáva prevažne v hrách, tak učenie posilňovaním má širokú škálu využitia, od riadenia svetelnej dopravy [5], cez zdravotníctvo až po robotiku [6] [7].

1.1 Neurónové siete

Neurónové siete (NN) sa radia k najpopulárnejším prístupom strojového učenia. Neurónovú sieť možno chápať ako nelineárnu funkciu $y = f(x|\theta)$, v ktorej x predstavuje informáciu na vstupe, y predikovaný výstup a θ reprezentuje množinu učiacich sa parametrov. NN sa často označujú za dopredné, pretože informácie prechádzajú zo vstupu x, cez vrstvy siete definované funkciou f, z ktorých sa počíta výstup y. Skladajú sa z viacerých funkcií, kde model siete je asociovaný ako orientovaný acyklický graf popisujúci štruktúru funkcií. Každá funkcia predstavuje jednu vrstvu, resp. zoskupenie učiacich sa parametrov. Napríklad funkcia reprezentujúca NN $f(x) = f_3(f_2(f_1(x)))$ je zložená z funkcií f_1, f_2 a f_3 (obrázok 1.1). V takomto prípade vstup x predstavuje vstupnú vrstvu, f_1 a f_2 sú skryté vrstvy, f_3 reprezentuje vrstvu výstupnú [8]. Hlboké

neurónové siete sa vyznačujú predovšetkým väčším počtom skrytých vrstiev.



Obrázok 1.1: Zložená funkcia

Na konci každej vrstvy (s možnou výnimkou výstupnej) sa nachádza aktivačná funkcia, ktorá zabezpečuje nelinearitu predikčného modelu. Medzi najpoužívanejšie aktivačné funkcie patria:

- Sigmoid $f(x) = \frac{1}{1+e^{-x}}$
- Hyperbolický tangens $f(x) = tanh(x)$
- ReLU $f(x) = max(0, x)$
- Leaky ReLU $f(x) = max(0.1x, x)$
- ELU $f(x) = \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$

Parametre sa učia na základe chybovej funkcie, ktorá závisí od typu úlohy. V regresných úlohách sa používa prevažne priemerná štvorcová chyba (MSE) vyjadrená ako $L_{MSE}(\hat{y}, y) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$, kde N je počet entít vo vzorke dát, y predstavuje vektor požadovaných hodnôt a $\hat{y}$ vektor predikovaných hodnôt.

V úlohách klasifikácie sa využíva chybová funkcia krížovej entropie, definovaná ako $L_{CE}(\hat{y}, y) = -\sum_{i=1}^{N} y_i \cdot \log(\hat{y}_i)$, kde y_i predstavuje požadované rozdelenie pravdepodobnosti i-teho prvku a $\hat{y}_i$ predikované rozdelenie pravdepodobnosti i-teho prvku. Chybová funkcia často tvorí hlavnú časť účelovej funkcie $J(\theta)$, ktorá môže obsahovať aj ďalšie kritéria (napr. rôzne regularizácie).

Využitím chybovej funkcie sa vytvára chybový priestor, ktorý je závislý od parametrov siete θ. Počet dimenzií chybového priestoru sa rovná počtu učiacich sa parametrov $|\theta|$. V chybovom priestore sa nachádzajú body so zaujímavými vlastnosťami a to síce globálne minimum a kritické body. V globálnom minime je hodnota chybovej funkcie najnižšia. Kritické body, akými sú lokálne minimá, lokálne maximá a sedlové body,

sa vyznačujú nulovou deriváciou. Tréning NN možno prirovnať pohybu v chybovom priestore.

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, J(\theta) \tag{1.1}$$

Cieľom tréningu je aproximácia funkcie f^*, tá predstavuje funkciu s optimálnymi parametrami siete θ^* (1.1). Optimálne parametre dosahujú najnižšiu hodnotu účelovej funkcie $J(\theta)$, resp. dosiahnu globálne minimum v chybovom priestore [8]. Napriek uvedenému cieľu, konvergencia ku globálnemu minimu v chybovom priestore nie je garantovaná a v mnohých prípadoch je efektívnejšie uspokojiť sa s vhodným lokálnym minimom.

Hľadanie vhodných parametrov začína výpočtom hodnoty účelovej funkcie, resp. chyby na vzorke dát. Chyba je následne derivovaná vzhľadom na parametre θ pomocou parciálnych derivácií, resp. zreťazeného pravidla ($\frac{\partial c}{\partial a} = \frac{\partial c}{\partial b} \cdot \frac{\partial b}{\partial a}$), čím sa vypočíta gradient pre každý jeden parameter NN, ktorý sa podieľal na výpočte výstupu.

$$\theta_{i+1} = \theta_i - \alpha \nabla_\theta J(\theta) \tag{1.2}$$

Metóda aplikácie vypočítaných gradientov na učiace sa parametre NN (rovnica 1.2) sa nazýva metóda klesajúceho gradientu. Koeficient miery učenia α je hyperparameter, ktorý určuje veľkosť vplyvu gradientov na hodnoty učiacich sa parametrov, resp. udáva veľkosť kroku v chybovom priestore.

Opak metódy klesajúceho gradientu je metóda stúpajúceho gradientu, ktorý sa používa pri maximalizácií hodnoty účelovej funkcie $J(\theta)$.

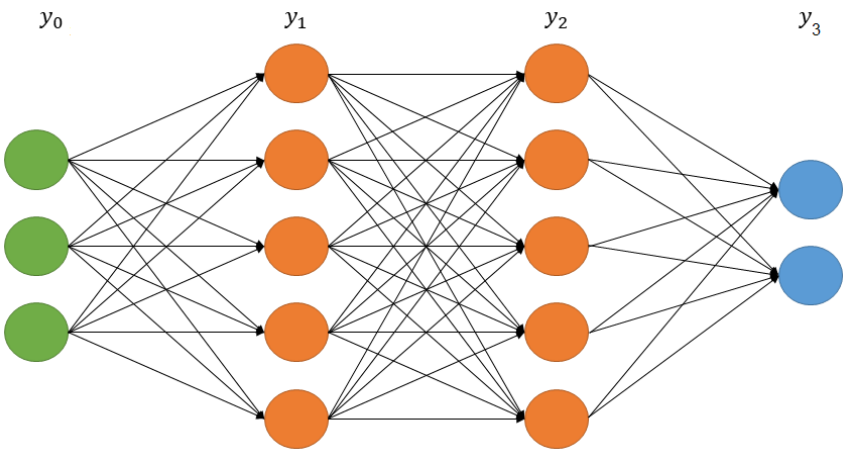
Metódy úpravy parametrov siete majú problémy v kritických bodoch (derivácia je nulová), v ktorých nastáva uviaznutie. Riešením je využitie tzv. optimalizátorov, ktoré sú rozšírením základnej metódy. Tie pri úprave hodnôt učiacich sa parametrov zohľadňujú aj zotrvačnosť smeru a rýchlosti pohybu v chybovom priestore. Zotrvačnosť umožňuje posun aj v kritických bodoch. Optimalizátorov existuje pomerne veľa, najznámejšie sú AdaGrad [8], RMSprop [8] a ADAM [9].

Pred tréningom je potrebné vybrať vhodné hyperparametre pre konkrétnu úlohu. Za ďalší dôležitý hyperparameter sa označuje architektúra NN, resp. počet a typ vrstiev spolu s počtami parametrov v jednotlivých vrstvách.

1.2 Plne prepojené neurónové siete

Základným typom NN sú plne prepojené neurónové siete. Výstupný vektor každej vrstvy y_i je plne prepojený s výstupným vektorom predchádzajúcej vrstvy y_{i-1} za pomoci učiacich sa parametrov nazývaných váhy (obrázok 1.2). Výnimkou je vstupná vrstva y_0, ktorá reprezentuje informáciu na vstupe. Váhy medzi vrstvou i a $i-1$ sú označované ako matica W_i s rozmermi $|y_i| \times |y_{i-1}|$ Výpočet hodnoty neurónov nasledujúcej vrstvy je definovaný rovnicou 1.3, v ktorej f_i predstavuje aktivačnú funkciu $i-tej$ vrstvy, W_i je matica váh, $\otimes$ reflektuje operáciu súčinu matíc a b_i je vektor učiacich sa parametrov vychýlenia, tzv. bias. Z rovnice vyplýva, že výpočet jedného výstupného neurónu zahŕňa súčet násobení každého neurónu predchádzajúcej vrstvy s príslušnou váhou, pripočítanie parametru vychýlenia a aplikáciou aktivačnej funkcie.

$$y_i = f_i(W_i \otimes {y_{i-1}}^T + b_i) \tag{1.3}$$



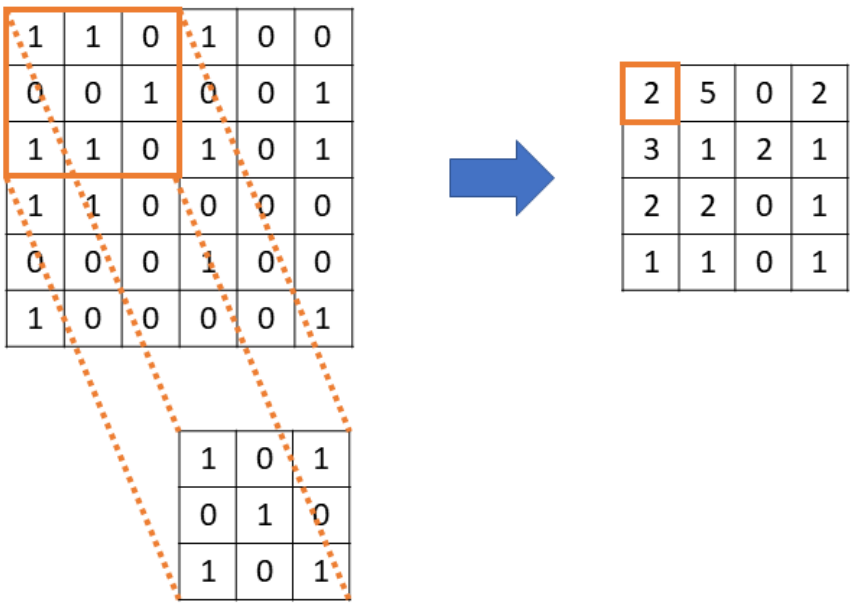
Obrázok 1.2: Plne prepojená neurónová sieť

1.3 Konvolučné neurónové siete

Plne prepojené neurónové siete sú neefektívne pri spracovaní priestorovej informácie, akou môže byť priestorový vstup. Napríklad pri spracovaní farebného obrazu veľkosti 100×100 pixelov (s hĺbkou kanálov 3 pre RGB obrázok) by vstupná vrstva mala veľkosť 30000 neurónov, takže každý neurón v nasledujúcej vrstve by potreboval 30000 váh pre plné prepojenie, čím narastá počet učiacich sa parametrov a s ním aj výpočtové a

pamäťové nároky na tréning. Preto sa v úlohách s priestorovou informáciou používajú konvolučné neurónové siete (CNN) [10]. Pre šírenie informácie zo vstupu na výstup využívajú tenzory váh nazývané filtre. CNN aplikuje filter postupne na celý obrázok za pomoci operácie konvolúcie (obrázok 1.3), čím vznikne výstupná matica, ktorá prechádza aktivačnou funkciou. Takáto výstupná matica sa nazýva matica príznakov a tvorí výstup konvolučnej vrstvy (akýsi ekvivalent pre neuróny v plne prepojených neurónových sieťach). Operáciu konvolúcie v CNN môžeme vypočítať rovnicou 1.4, kde D, C, a R sú rozmery filtra, X je vstup z predchádzajúcej vrstvy s hĺbkou D, $Y_{i,j}$ je výstup konvolúcie vo výstupnej matici $Y_{W\times H}$. Rozmer výstupnej matice závisí od vstupného obrázku a veľkosti filtra, ktorý sa vypočíta $W_Y = W_X - C + 1$ pre šírku a $H_Y = H_X - R + 1$ pre výšku. V praxi sa v jedenej konvolučnej vrstve používa viac filtrov, ktorých výstupné matice príznakov sa vkladajú na seba a tak vzniká tenzor príznakov. Tenzor reprezentuje výstupnú vrstvu. Teda s počtom filtrov narastá hĺbka výstupu konvolučnej vrstvy. Každý filter sa učí rozpoznávať iný vzor z výstupu predchádzajúcej vrstvy. Napríklad pri spracovaní obrázku sa filtre prvej vrstvy učia rozpoznávať hrany, filtre druhej vrstvy rozpoznávajú kontúry a ďalšie vrstvy stále komplikovanejšie vzory.

$$Y_{i,j} = \sum_{d=0}^{D-1} \sum_{c=0}^{C-1} \sum_{r=0}^{R-1} X_{d,i+c,j+r} \cdot w_{d,c,r} \tag{1.4}$$



Obrázok 1.3: Operácia konvolúcie

V úlohách klasifikácie sa CNN používajú za účelom extrakcie príznakov z obrazu. Za

poslednou konvolučnou vrstvou sa zvyčajne nachádza jedna alebo viac plne prepojených vrstiev.

Pri použití filtra s rozmermi $C > 1$ alebo $R > 1$ sa vstupný tenzor zmenší o jeden alebo viac pixelov (záleží na rozmeroch filtra) na výstupe. Ak v CNN nasleduje viac vrstiev s takýmito filtrami za sebou, môže nastať problém s rozpoznávaním vzorov v okrajoch vstupu. Riešením je použiť vyplnenie (padding), ktoré vypĺňa okraje vstupu nulami.

Ďalšou metódou používanou v CNN je podvzorkovanie (pooling), ktoré dokáže zmenšiť výstup z vrstvy n-násobne. Podvzorkovanie sa používa pre zníženie výpočtovej náročnosti pri väčších rozmeroch vstupu. Často sa používa rozmer podvzorkovacieho filtra 2×2, ktoré zmenší vstup 2-násobne. Stratégia výberu počas podvzorkovania môže byť napr. výber najvyššej hodnoty z aktuálne vzorkovanej oblasti, alebo priemer všetkých hodnôt vzorkovanej oblasti. Zmenšenie vstupu je taktiež možné dosiahnuť aj nastavením posunu, resp. kroku konvolúcie o $k > 1$, čím sa operácia konvolúcie bude vykonávať každých k krokov a tým sa zmenší rozmer výstupného tenzoru príznakov.

1.3.1 Reziduálne bloky

Reziduálne bloky [11] sa využívajú najmä v hlbokých sieťach, zložených z veľkého počtu skrytých vrstiev. Ich hlavnou úlohou je riešenie problému miznutia gradientov na počiatočných vrstvách (spätné šírenie chyby začína od konca neurónovej siete a veľkosť gradientov sa počas šírenia do prvých vrstiev znižuje). Hlavným princípom je sčítanie vstupu vchádzajúceho do reziduálneho bloku s jeho výstupom, resp. $f(x) + x$, kde x je vstup a funkcia f reprezentuje reziduálny blok. Myšlienka reziduálnych blokov bola použitá v architektúre ResNet [11], v ktorej jeden blok obsahoval dve konvolučné vrstvy spolu s aktivačnými funkciami. Takáto sieť mala hĺbku až 152 vrstiev.

1.4 Rekurentné neurónové siete

Pri spracovaní videa či textu, vzniká potreba zapamätať si predchádzajúce vstupy, resp. sekvenciu na seba nadväzujúcich vstupov. Riešením sú rekurentné neurónové siete (RNN) [12].

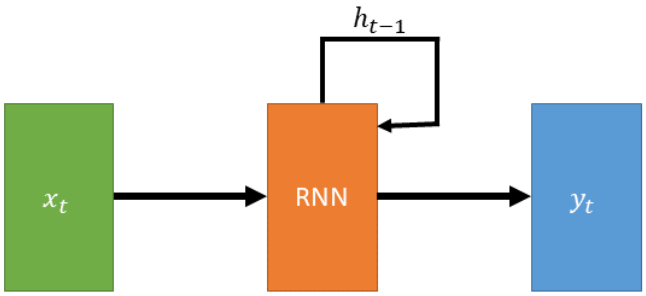
Hlavnou myšlienkou je zapamätanie si tzv. skrytého stavu, ktorý reprezentuje časť

historických informácií (obrázok 1.4). Skrytý stav sa označuje ako h_t v čase t (index t môže okrem času predstavovať aj index v sekvencii vstupných informácií atď.). Pôvodný návrh výpočtu skrytého stavu (1.5) závisí od predchádzajúceho skrytého stavu h_{t-1} spracovaného parametrami W_{hh} a od aktuálneho vstupu x_t spracovaného parametrami W_{xh}. Hyperbolický tangens sa používa pre zamedzenie nárastu hodnôt skrytého stavu a tým predchádza "explózii" gradientov. Výsledok RNN (1.6) teda závisí aj od skrytého stavu h_t a parametrov W_{hy}.

Využívajú sa najmä pri spracovaní textu, detekcii objektov [13] alebo ako súčasť agenta AlphaStar pôsobiaceho v hre Starcraft 2 [4].

$$h_t = tanh(W_{hh} \otimes h_{t-1} + W_{xh} \otimes x_t) \tag{1.5}$$

$$y_t = W_{hy} \otimes h_t \tag{1.6}$$



Obrázok 1.4: Rekurentná neurónová sieť

Pokročilejšími RNN sú napríklad LSTM [12] alebo GRU [14], ktoré dosahujú lepšie výsledky.

Kapitola 2

Učenie posilňovaním

Učenie posilňovaním (RL) je založené na riešení mapovania vzniknutých situácií do možných akcií s cieľom maximalizácie signálov s odmenami. Učiteľ nedefinuje, ktorá akcia sa má v danom stave vykonať, algoritmus na to musí prísť sám postupným skúšaním akcií. [15]

Agent komunikuje s prostredím, v ktorom vykonáva akcie. Za vykonanú akciu získa z prostredia signál s odmenou r_t v čase t. Odmena môže byť kladná, záporná alebo nulová. Úlohou odmien je správne usmerniť agenta v jeho rozhodnutiach. Odmeny môžu byť okamžité, tie agent obdrží hneď po vykonanej akcií, alebo oneskorené, ktoré dostane po vykonaní sekvencie akcií, prípadne na konci simulácie.

Prostredie predstavuje simulátor s konkrétnou úlohou, jasne definovanými pravidlami a odmenami. Tieto podmienky spĺňajú hry, ktoré sú často využívané ako prostredia pre testovanie nových algoritmov. Ako uvádzame vyššie, počítačovú hru možno chápať ako vopred vytvorený simulátor s jasne definovanou úlohou, akciami a odmenami vo forme skóre. Výnimočne sú obohatené množinou záznamov z majstrovských zápasov. Stav prostredia s_t v čase t popisuje aktuálnu situáciu v prostredí, hodnoty premenných prostredia a pod. (napr. v šachu môže stav prezentovať rozloženie figúrok na šachovnici). Na základe stavu sa agent rozhoduje, akú akciu vykonať. Po vykonaní akcie obdrží od prostredia okamžitú odmenu spolu s informáciou o novovzniknutom stave.

Hustota výskytu odmien môže byť rôzna a závisí od typu úlohy a prostredia. Niektoré prostredia obsahujú veľký počet odmien, čím sa agent dokáže rýchlejšie naučiť vykonávať správne rozhodnutia. Ak sú odmeny do prostredia pridávané "na silu" (napr. v šachu funkciou hodnotiacou výhodnosť stavu podľa šachových expertov), tak naučená

stratégia bude silne ovplyvnená subjektívnymi pocitmi autora odmien. Takýto agent sa predovšetkým naučí stratégie, ktoré autori odmien pokladajú za správne. Ďalšou možnosťou je použitie len koncových odmien (odmeny na konci hry) - agent sa učí dlhšie, avšak má potenciál objaviť nové stratégie.

2.1 Terminológia

V terminológii učenia posilňovaním sa spôsob rozhodovania agenta nazýva politika π. Politika definuje spôsob správania agenta v stavoch. Inými slovami, politika predstavuje mapovanie získaného stavu z prostredia do výberu akcií. V niektorých prípadoch politika môže byť jednoduchá funkcia alebo vyhľadávanie v tabuľke hodnôt, v iných zas vyžaduje rozsiahle výpočty napr. v podobe hlbokej NN. Politika tvorí základ agentov učenia posilňovaním. Vo všeobecnosti môže byť stochastická $\pi(s, a)$, popisujúca pravdepodobnosti vykonania každej akcie alebo deterministická $\pi(s)$ - v konkrétnom stave sa vyberá práve jedna akcia. [15]

Epizóda predstavuje jeden beh simulačného prostredia, v ktorom agent vykonáva sériu rozhodnutí - napr. jedna odohratá hra. Sekvencia stavov, akcií a získaných odmien, ktoré agent vykonal v prostredí počas jednej epizódy sa nazýva trajektória $\tau = \{s_0, a_0, r_0, s_1, a_1, r_1, ..., s_{T-1}, a_{T-1}, r_{T-1}, s_T\}$, kde T je celkový čas epizódy a s_T predstavuje terminálový stav - stav, v ktorom končí epizóda.

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \sum_{t=0}^{T-1} \gamma^t \cdot r_t \tag{2.1}$$

Cieľom agenta je nájsť takú politiku π^*, ktorá maximalizuje sumu budúcich zrážaných odmien (2.1). Pri zohľadňovaní budúcich odmien je potrebné brať do úvahy faktor zrážania γ, ktorý určuje do akej miery agentovi záleží na odmenách v budúcich stavoch. Jedná sa o parameter algoritmu s konštantnou hodnotou z intervalu $[0, 1]$. Ak $\gamma = 0$, agent pri rozhodovaní nebude brať do úvahy budúce odmeny. Ak $\gamma = 1$, pre agenta budú nadchádzajúce odmeny rovnako dôležité ako okamžitá odmena. Pri $\gamma = 1$ nastáva problém, v ktorom sa agent nesnaží získavať budúce odmeny čo najskôr, to môže mať za následok neefektívne rozhodnutia a vznik cyklov v trajektórii.

Suma budúcich zrážaných odmien do konca hry počas epizódy v ľubovoľnom čase t sa počíta ako $G_t = \sum_{i=t}^{T-1} \gamma^{i-t} \cdot r_i$.

$$V(s) = \mathbb{E}(G_t|s_t = s) = \mathbb{E}(r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \cdots + \gamma^{T-t-1} \cdot r_{T-1}|s_t = s) \quad (2.2)$$

Ďalšou dôležitou funkciou v RL je hodnotová funkcia stavu. Hodnotová funkcia stavu (ďalej len hodnota stavu) $V(s)$ je očakávaná stredná hodnota sumy budúcich zrážaných odmien v konkrétnom stave s (2.2). Taktiež ako v prípade politiky, funkcia môže byť reprezentovaná jednoduchou matematickou funkciou, vyhľadávaním v tabuľke hodnôt alebo predikciou hlbokej NN. Hodnota stavu sa často uvádza s horným indexom π, resp. $V^\pi(s)$ a označuje očakávanú hodnotu stavu vzhľadom na politiku π. Horný index taktiež môže byť uvedený ako $*$, resp. $V^*(s)$, ten značí očakávanú hodnotu stavu podľa rozhodnutí optimálnej politiky π^*.

Posledným z hlavných pojmov terminológie RL je model prostredia. Model prostredia predstavuje správanie prostredia. Napr. môže byť reprezentovaný funkciou, ktorá z kombinácie stavu a akcie vráti nasledujúci stav. Takto koncipované modely prostredia umožňujú zvažovať budúce následky ešte pred tým, ako boli uskutočnené. Agenti tak môžu zahrnúť možnosť plánovania do rozhodovacieho procesu.

Metódy RL, ktoré využívajú model prostredia pre podporu budúcich rozhodnutí sa nazývajú modelovo založené metódy. Opakom sú metódy bez modelu prostredia, ktoré sa učia metódou "pokus-omyl". [15]

V aktuálnej kapitole sú popísané základné postupy učenia posilňovaním, ktoré využívajú tabuľkové metódy pre uchovanie dôležitých hodnôt. V nasledujúcich kapitolách sú popísané pokročilejšie algoritmy s využitím hlbokých NN.

2.2 Markovov rozhodovací proces

RL využíva formálnu štruktúru Markovových rozhodovacích procesov (MDP) ako definovanie interakcií medzi učiacim agentom a prostredím z hľadiska stavov, akcií a odmien. Takáto štruktúra prezentuje jednoduchou cestou dôležité vlastnosti problematiky RL. [15]

MDP pre úlohy RL možno zapísať ako štruktúru (S, A, P, R), ktorá sa skladá z:

- S je množina všetkých stavov prostredia.
- A je množina všetkých akcií v prostredí.

- P je model prechodov prostredia medzi stavmi, ktorý popisuje pravdepodobnosť prechodu $p(s_{t+1} = s'|s_t = s, a_t = a)$, resp. aká je pravdepodobnosť, ak agent v stave s vykoná akciu a, tak sa dostane do stavu s'.

- R je funkcia odmien, ktorá mapuje odmenu pre dvojice stav s a akcia a. Možno ju zapísať ako $R(s_t = s, a_t = a) = \mathbb{E}(r_t|s_t = s, a_t = a)$

V RL úlohách nastávajú prípady, kedy je vhodné vedieť očakávanú sumu budúcich zrážaných odmien nie len vzhľadom na stav, podľa aktuálnej politiky, ale aj vzhľadom na akciu a. Takáto suma sa označuje ako q hodnota a je získaná funkciou $Q^\pi(s, a)$, ktorá mapuje dvojicu stav a akcia. V MDP štruktúre s modelom prostredia môžu byť q hodnoty získané pomocou funkcie $V^\pi(s)$ (rovnica 2.3). V úlohách bez modelu prostredia môže Q funkcia obsahovať podobne ako v prípade politiky, jednoduché prehľadávanie tabuliek, jednoduchú funkciu alebo komplexné výpočty.

$$Q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} [P(s'|s, a) \cdot V^\pi(s')] \tag{2.3}$$

2.3 Dynamické programovanie

Vďaka modelu s MDP štruktúrou (modelovo založená úloha) vzniká možnosť riešiť RL úlohy s menším počtom stavov a akcií pomocou dynamického programovania (DP). Z dôvodu jednoduchšieho pochopenia základných postupov, sme sa rozhodli uvedené vzorce a algoritmy v sekcii DP demonštrovať na deterministickej politike π. Cieľom je nájsť takú politiku, ktorá maximalizuje hodnotu stavu v každom stave prostredia (rovnica 2.4). Riešenie úlohy cez DP zahŕňa správny odhad stavových hodnôt a následné zlepšenie politiky.

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, V^\pi(s) \tag{2.4}$$

2.3.1 Zlepšenie odhadu stavových hodnôt

Keďže cieľ agenta závisí od funkcie $V^\pi(s)$ (rovnica 2.4), je nevyhnutné aby hodnotová funkcia stavu správne popisovala hodnoty jednotlivých stavov podľa aktuálnej politiky. Pre popis stavových hodnôt je potrebný správny odhad skutočných hodnôt, ktorú

rieši algoritmus (alg. 1). Algoritmus začína inicializáciou stavových hodnôt. Odhady sa postupne zlepšujú pomocou Bellmanovej rovnice, resp. techniky dynamického programovania, kde pre získanie novej hodnoty $V_k^\pi(s)$ sa využíva hodnota z predchádzajúcej iterácie označovaná ako $V_{k-1}^\pi(s)$. Podmienka pokračovania cyklu b sa dá realizovať pomocou kontroly konvergencie cez L_1 normu.

Algoritmus 1 Zlepšenie odhadu stavových hodnôt

$V_0(s) \leftarrow 0$ pre $s \in S$

$k \leftarrow 1$

$b \leftarrow true$

while b **do**

  for $s \in S$ **do**

    $V_k^\pi(s) \leftarrow R(s, \pi(s)) + \gamma \sum_{s' \in S} \left[P(s'|s, \pi(s)) \cdot V_{k-1}^\pi(s') \right]$

  end for

  $b \leftarrow \|V_k^\pi - V_{k-1}^\pi\| > 0$

  $k \leftarrow k + 1$

end while

2.3.2 Zlepšenie politiky

Deterministickú politiku možno chápať ako tabuľku, v ktorej pre každý stav existuje práve jedna akcia, ktorú má agent v danom stave vykonať. Algoritmus pre zlepšenie politiky (alg. 2) najskôr vypočíta q hodnoty pre všetky možné kombinácie stav-akcia s už aproximovanými stavovými hodnotami V^{π_i} podľa aktuálnej politiky. Následne prebieha výber novej politiky π_{i+1} tak, že stavu s sa priradí akcia s najvyššou hodnotou $Q(s, a)$, pretože $\max_a Q^{\pi_i}(s, a) \geq V^{\pi_i}(s)$.

Algoritmus 2 Zlepšenie politiky

for $s \in S$, $a \in A$ **do**

  $Q^{\pi_i}(s, a) \leftarrow R(s, a) + \gamma \sum_{s' \in S} [P(s'|s, a) \cdot V^{\pi_i}(s')]$

end for

for $s \in S$ **do**

  $\pi_{i+1}(s) \leftarrow \text{argmax}_a \, Q^{\pi_i}(s, a)$

end for

2.3.3 Algoritmus učenia

Dve vyššie uvedené metódy (alg. 1 a 2) sú dôležitými časťami algoritmu pre riešenie RL úloh pomocou DP. Výsledný algoritmus (alg. 3) najskôr náhodne inicializuje politiku, potom v cykle vykonáva aproximáciu stavových hodnôt a zlepšenie politiky. Podmienka pokračovania cyklu (s výnimkou prvej iterácie) je zmena politiky kontrolovaná L_1 normou.

Algoritmus 3 Riešenie RL úloh pomocou dynamického programovania

$i \leftarrow 0$

Náhodná inicializácia $\pi_0(s)$ pre $s \in S$

while $i = 0$ OR konvergencia π **do**

  $V^{\pi_i} \leftarrow$ Aproximácia stavových hodnôt

  $\pi_{i+1} \leftarrow$ Zlepšenie politiky

  $i \leftarrow i + 1$

end while

2.4 Hľadanie politiky bez modelu prostredia

V úlohách bez pomocného modelu prostredia sa agent musí vedieť zaobísť bez pravdepodobnostného modelu prechodov a odmenovej funkcie. Učenie môže aplikovať len na zažité situácie a tak pri preskúmavaní prostredia často využíva prístup "pokus-omyl".

2.4.1 Vyhodnotenie politiky

Cieľ agenta je rovnaký ako v úlohách s modelom prostredia, t.j. nájdenie politiky, ktorá maximalizuje hodnoty stavov (rovnica 2.4). Pre vyhodnotenie politiky sa používajú metódy ako Monte Carlo, metóda časových rozdielov a ich vzájomná kombinácia. Nižšie je popísaná aproximácia stavových hodnôt, avšak metódy môžu aproximovať aj q hodnoty zmenou zoznamu stavov na zoznam stavov a akcií.

Metóda Monte Carlo

Hlavným princípom metódy Monte Carlo (alg. 4) je získavanie trajektórií z epizód, v ktorých sa agent rozhodoval podľa aktuálnej politiky π a následne spätné sčítanie zrážaných odmien pre každý stav. Agent zahrá epizódu do konca hry a potom spätne prepočíta sumu zrážaných odmien pre každý stav. Metóda si pamätá počty navštívení stavu $N(s)$ a sumu zrážaných odmien $G(s)$ pre všetky stavy.

Výsledné hodnoty stavov môžu byť vychýlené, pretože agent môže viackrát v jednej trajektórii navštíviť rovnaký stav. Riešením je úprava počtu navštívení a sumy zrážaných odmien len po prvom navštívení počas epizódy.

Algoritmus 4 Metóda Monte Carlo

$N(s) \leftarrow 0$, $G(s) \leftarrow 0$ pre $s \in S$
$i \leftarrow 0$
while konvergencia V^π **do**
  Získaj trajektóriu $\tau_i = \{s_{i,0}, a_{i,0}, r_{i,0}, ..., s_{i,T}\}$ podľa politiky π
  $R \leftarrow 0$
  for $t = T - 1$; $t \geq 0$; $t = t - 1$ **do**
    $R \leftarrow r_{i,t} + \gamma \cdot R$
    $N(s_t) \leftarrow N(s_t) + 1$
    $G(s_t) \leftarrow G(s_t) + R$
    $V^\pi(s_t) \leftarrow \frac{G(s_t)}{N(s_t)}$
  end for
  $i \leftarrow i + 1$
end while

Metóda časových rozdielov

Nevýhodou metódy Monte Carlo je fakt, že agent musí odohrať epizódu až do konca hry (jedine tak môže spätne sčítavať skutočné odmeny). Opakom metódy Monte Carlo je metóda časových rozdielov (alg. 5), kde sa hodnota stavu s_t počíta po vykonaní akcie a_t, resp. po získaní okamžitej odmeny r_t a nového stavu s_{t+1}. Predpokladom metódy je $V^\pi(s_t) = r_t + \gamma \cdot V^\pi(s_{t+1})$. Ak rovnosť dosiahnutá nie je, tak sa pôvodná hodnota $V^\pi(s_t)$ upraví o rozdiel vynásobený koeficientom učenia α.

Algoritmus 5 Metóda časových rozdielov

$V(s) = 0$ pre $s \in S$

while konvergencia V^π **do**

Získaj štvoricu (s_t, a_t, r_t, s_{t+1})

$V^\pi(s_t) = V^\pi(s_t) + \alpha([r_t + \gamma V^\pi(s_{t+1})] - V^\pi(s_t))$

end while

Kombinácia metód

Metóda Monte Carlo sa označuje ako $TD(1)$ a vďaka uchovaniu si všetkých získaných odmien počas trajektórie má zaujímavé vlastnosti, t.j. nízke vychýlenie a veľký rozptyl. Metóda časových rozdielov sa označuje ako $TD(0)$, jej vlastnosti sú opačné, t.j. vysoké vychýlenie a nízky rozptyl. Každá z uvedených metód dokáže aproximovať stavové hodnoty. Avšak existujú typy úloh, v ktorých sa prvá metóda vyznačuje lepšou aproximáciou, v iných typoch úloh zas lepšie aproximuje metóda druhá. V niektorých prístupoch sa používa kombinácia týchto dvoch metód $TD(\lambda)$, ktorá spočítava odmeny získané za posledných k krokov, pričom nakoniec sa použije hodnota stavu $V^\pi(s_{t+k})$, čím sa zníži rozptyl aj vychýlenie.

2.4.2 Hľadanie najlepšej politiky

V tabuľkových metódach pre hľadanie najlepšej politiky sa najčastejšie používajú dve metódy - Q učenie a SARSA. Obe metódy sú založené na aproximácii Q hodnôt, keďže nemajú prístup ku všetkým odmenám v ľubovoľnom stave.

Politika ϵ-greedy

Na začiatku učiaceho procesu sú q hodnoty inicializované náhodne. Ak v takomto prípade bude agent vyberať len akcie s najvyššími q hodnotami ($\pi_{i+1}(a) = \text{argmax}_a\, Q^{\pi_i}(s, a)$), tak sa mu nemusí priblížiť ku optimálnej politike. Preto je potrebné vyskúšať aj akcie s nižšími q hodnotami, aby tak preskúmal čo najväčšiu časť stavového priestoru. Pre dosiahnutie akéhosi kompromisu, ktorý určuje, kedy má agent zvoliť akciu s najvyššou q hodnotou a kedy zvoliť akciu s nižšou hodnotou sa používa parameter ϵ. ϵ je číslo z intervalu $[0; 1]$ a udáva percentuálny pomer medzi výberom najlepších a náhodných akcií. Najčastejšie sa vygeneruje číslo z intervalu $[0; 1)$ za pomoci generátora rovnomerného

rozdelenia pravdepodobnosti. Ak je číslo menšie ako ϵ, tak sa vykoná náhodná akcia, v opačnom prípade sa vykoná akcia s najvyššou q hodnotou. Takýto prístup sa nazýva politika ϵ-greedy. Na začiatku učenia je vhodné, aby hodnota ϵ bola čo najvyššia. Počas učenia sa hodnota ϵ znižuje s počtom epizód (napr. $\epsilon_i = \frac{1}{i}$ pre i-tu epizódu. Často sa nastavuje dolná hranica pre ϵ, pod ktorú hodnota ϵ nemôže klesnúť. Nasledujúce dva prístupy učenia uvádzame práve s využitím ϵ-greedy politiky, avšak tá nemusí byť ich nevyhnutnou súčasťou.

Q učenie

Metóda Q učenia (alg. 6) sa často používa v úlohách RL. K aproximácií q hodnôt využíva štvoricu dát (s_t, a_t, r_t, s_{t+1}), resp. stav s_t, akcia a_t, odmena r_t a nasledujúci stav s_{t+1}. Agent pri učení pôsobí optimistickejšie, pretože predpokladá vykonanie najvýhodnejšej akcie v nasledujúcom stave. Takýto "optimizmus" je prínosom v širokej škále úloh.

Algoritmus 6 Q učenie

$t \leftarrow 0$, $s_t \leftarrow s_0$

while konvergencia Q^π **do**

$\epsilon \leftarrow max\left(\frac{1}{t+1}, \epsilon_{min}\right)$

Vykonaj akciu $a_t = \pi(s_t)$ s pravdepodobnosťou $1 - \epsilon$, inak náhodná akcia

Preskúmaj $(r_t,\ s_{t+1})$

$Q^\pi(s_t, a_t) \leftarrow Q^\pi(s_t, a_t) + \alpha \cdot [r_t + \gamma \cdot \max_a Q^\pi(s_{t+1}, a) - Q^\pi(s_t, a_t)]$

$\pi(s_t) \leftarrow \text{argmax}_a\, Q^\pi(s_t, a_t)$

$t \leftarrow t + 1$

end while

SARSA

Metóda SARSA (alg. 7) využíva k aproximácii q hodnôt päticu dát $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$, resp. stav s_t, akcia a_t, odmena r_t, nasledujúci stav s_{t+1} a akcia v nesledujúcom stave a_{t+1}. Na rozdiel od Q učenia, agent pristupuje k vykonávaniu akcií kontroverzne, t.j. berie do úvahy skutočne vykonanú akciu v nasledujúcom stave namiesto najlepšej. Takýto výber znižuje možnosť preceňovania akcií a tým aj rozptyl odhadovaných hodnôt. Kontroverzný prístup je vhodný v prípadoch, ak je potrebné minimalizovať získavanie

záporných odmien počas tréningu (napr. pri trénovaní agenta v reálnom svete) alebo ak vykonanie nevhodnej akcie môže rýchlo a nepriaznivo ukončiť hru (napr. pohyb po útese).

V prípade nekonečného počtu navštívení všetkých kombinácií stavov a akcií, metóda SARSA konverguje s pravdepodobnosťou 100% k optimálnej politike [15].

Algoritmus 7 SARSA

$t \leftarrow 0$, $s_t \leftarrow s_0$

Vykonaj akciu $a_t = \pi(s_t)$

Preskúmaj $(r_t,\ s_{t+1})$

while konvergencia Q^π **do**

$\epsilon \leftarrow max\left(\frac{1}{t+1}, \epsilon_{min}\right)$

Vykonaj akciu $a_{t+1} = \pi(s_{t+1})$ s pravdepodobnosťou $1 - \epsilon$, inak náhodná akcia

Preskúmaj $(r_{t+1},\ s_{t+2})$

$Q^\pi(s_t, a_t) \leftarrow Q^\pi(s_t, a_t) + \alpha \cdot [r_t + \gamma \cdot Q^\pi(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$

$\pi(s_t) \leftarrow \text{argmax}_a\, Q^\pi(s_t, a_t)$

$t \leftarrow t + 1$

end while

Kapitola 3

Algoritmy bez modelu prostredia

Zvyčajným problémom tabuľkovo založených metód je práve pamäťová náročnosť, resp. obmedzený počet kombinácií stavov a akcií, ktoré môže zariadenie uchovať v pamäti (napríklad hra GO s veľkosťou hernej plochy 19×19 má počet možných stavov 10^{170} [2]). Preto sa miesto tabuľkových metód častejšie používajú prediktívne modely, resp. hlboké NN, ktoré sa snažia pomocou parametrov θ vhodne predikovať požadované hodnoty. V kapitole sú predstavené dve skupiny algoritmov bez modelu prostredia: Q učenie a politika gradientu.

3.1 Značenie

Vo vedeckých článkoch z oblasti RL sa čitateľ často môže stretnúť s rôznym značením jedného pojmu. Preto považujeme za dôležité stanoviť si jednotné značenie pred opisom algoritmov využívajúcich NN.

Stav prostredia sme doteraz označovali podľa pôvodnej terminológie s, resp. s_t, avšak niektoré vedecké články uprednostňujú pojem pozorovanie, resp. pozorovaný stav s označením o, resp. o_t. Pri vysvetľovaní algoritmov bez modelu prostredia by sme sa v teoretickej rovine zaobišli len s pôvodným značením s, avšak v kapitole venovanej algoritmom s modelom prostredia by vznikali kolízie pojmov. Taktiež vstup NN nemusí reprezentovať úplný stav prostredia, čím agent vidí len jeho časť - aktuálne pozorovaný stav. Preto považujeme za nevyhnutné značiť vstup NN z prostredia ako pozorovanie o, resp. o_t.

Takže stav prostredia s_t obsahuje všetky informácie potrebné pre úplný popis

prostredia v aktuálnom čase t, pričom agent dostáva z prostredia len pozorovaný stav o_t, ktorý nemusí obsahovať všetky informácie stavu. V doméne Atari hier predstavuje jedno pozorovanie posledné 4 zachytené obrazovky prostredia kvôli zachyteniu dynamických objektov (let loptičky, pohyb nepriateľov a pod).

Predikované hodnoty, ako napr. q hodnoty, sme doteraz označovali s horným indexom politiky π. Avšak pre úplnosť značenia, budeme ku predikovaným hodnotám uvádzať aj parametre NN θ, aby bolo zrejmé, ktorá NN predikovala dané hodnoty. Značenie $Q^{\pi}(s, a|\theta)$ teda obsahuje aj značenie politiky a zároveň parametre NN.

Hoci v niektorých nižšie uvedených algoritmoch sú parametre θ, (formou predikovaných hodnôt) plne zodpovedné za rozhodovací proces, u iných je rozhodovanie ovplyvnené aj ďalšími udalosťami (napr. ϵ-greedy politika). Preto funkciu politiky π podporenú parametrami θ v stave s budene označovať ako $\pi(s|\theta)$, avšak ak budeme spomínať len funkciu politiky všeobecne (nie pre konkrétny stav), tak budeme používať značenie π_{θ}.

3.2 Q učenie

Podobne ako pri tabuľkovo založených metódach, tak sa prístup Q učenia osvedčil aj v kombinácii s hlbokým učením. Tak vznikla skupina agentov využívajúca tzv. hlboké Q siete (DQN).

3.2.1 DQN

DQN agent [1] predikuje Q hodnoty pre každú akciu pomocou hlbokej NN s parametrami θ. Učenie Q hodnôt prebieha pomocou metódy časových rozdielov, ktorá je využitá pri výpočte cieľových hodnôt (rovnica 3.1).

Tréning NN je komplikovanejší ako aproximácia tabuľkových hodnôt, preto využíva rôzne vylepšenia a to síce cieľovú (druhú) NN a zoznam skúseností (alg. 8).

$$y_t = r_t + \gamma \cdot \max_a Q^{\pi}(o_{t+1}, a|\theta_{t-1}) \tag{3.1}$$

Cieľová neurónová sieť

Ak agent počíta cieľové hodnoty za pomoci predikcie aktuálnych parametrov θ_t a zároveň sa tie isté parametre aj učí, tak tento jav môže mať negatívny vplyv na tréning agenta vo forme nestability. Bolo zistené, že tréning je stabilnejší, ak sú cieľové hodnoty istý čas pevne ukotvené, preto DQN agent využíva dve siete - online sieť a cieľovú sieť [1]. Online NN sa učí nové poznatky zo zozbieraných dát z prostredia. Cieľová sieť sa využíva na predikciu q hodnôt v budúcom pozorovanom stave o_{t+1}, teda $Q^{\pi}(o_{t+1}, a|\theta_{t-1})$. Cieľová NN je oneskorenou kópiou parametrov online siete. Parametre θ sa kopírujú po stanovenej perióde, resp. po vykonaní K krokov. Hoci sa parametre zmenia až za dlhší čas, značenie parametrov sa udáva ako θ_t pre online sieť a θ_{t-1} pre sieť cieľovú.

Zoznam skúseností

Ďalším problémom, ktorý nastáva pri tréningu sú silne korelované dáta, ktoré na seba nadväzujú v každej trajektórii. Klasickým príkladom problému korelovaných dát je dlhotrvajúci pohyb agenta v bludisku len do jednej strany, čím sa učenie zameriava len na jednu akciu. Riešením je použitie zoznamu skúseností [1], ktorý rozbíja korelácie dát tak, že agent získané informácie $(o_t, a_t, r_t, o_{t+1}, T_t)$ uloží do zoznamu skúseností. Počas tréningu z neho vyberá náhodné vzorky pomocou rovnomerného rozdelenia pravdepodobnosti. Kvôli presnejšej predikcií terminálových odmien, je potrebné uložiť do zoznamu skúseností aj informáciu o terminálovom stave T_t. Vedľajším efektom zoznamu skúseností je riešenie problému zabúdania - ak agent dlho nenavštívil konkrétny stav, môže sa presnosť predikcie v konkrétnom stave znížiť.

Vylepšením zoznamu skúseností je prioritný zoznam skúseností, v ktorom sa vzorka dát vyberá podľa priority p_t. Tá sa počíta z chyby predikcie, resp. $p_t = |r_t + \gamma \cdot \max_a Q^{\pi}(o_{t+1}, a|\theta_{t-1}) - Q^{\pi}(o_t, a_t|\theta_t)|$.

3.2.2 DDQN

Pôvodný DQN agent má v niektorých úlohách problém s preceňovaním predikovaných q hodnôt, napr. ak odmena za vykonanú akciu nie je statická, ale je získaná z rozdelenia pravdepodobnosti. V takomto prípade agentovi dlhšie trvá, kým správne identifikuje strednú hodnotu konkrétneho rozdelenia [15]. Problém rieši modifikácia pôvodného

Algoritmus 8 DQN

Inicializuj zoznam skúseností D

Inicializuj parametre θ_t a skopíruj ich do θ_{t-1}

$M \leftarrow$ počet epizód

$\epsilon \leftarrow 1$

$k \leftarrow 0$

$K \leftarrow$ počet krokov pre zmenu parametrov NN

for epizóda = 0, M **do**

  Preskúmaj pozorovaný stav o_0

  for t = 0, T **do**

    $rand \leftarrow$ náhodné číslo z intervalu $[0, 1)$

    if $\epsilon > rand$ **then**

      $a_t \leftarrow$ náhodná akcia

    else

      $a_t \leftarrow \text{argmax}_a\, Q^\pi(o_t, a|\theta_t)$

    end if

    Vykonaj akciu a_t a preskúmaj r_t a o_{t+1}

    Ulož dáta $(o_t, a_t, r_t, o_{t+1}, T_t)$ do D

    Vygeneruj vektor N indexov vzorky dát $\vec{j}$

    Získaj vzorku dát $(o_{\vec{j}}, a_{\vec{j}}, r_{\vec{j}}, o_{\vec{j}+1}, T_{\vec{j}})$ z D

    Vypočítaj $y_{\vec{j}} = r_{\vec{j}} + \gamma \cdot \max_a Q^\pi(o_{\vec{j}+1}, a|\theta_{t-1}) \cdot T_j$

    Aplikuj pokles gradientu podľa $J(\theta_t) = \frac{1}{N} \cdot (y_{\vec{j}} - Q^\pi(o_{\vec{j}}, a_{\vec{j}}|\theta_t))^2$

    Zníž hodnotu ϵ

    $k \leftarrow k + 1$

    if $k \,\%\, K = 0$ **then**

      Skopíruj parametre θ_t do θ_{t-1}

    end if

  end for

end for

algoritmu nazývaná DDQN agent [16], resp. dvojitý DQN agent. DDQN agent dokáže výrazne zefektívniť učenie aj v prostrediach so statickými odmenami.

Zmena spočíva v počítaní cieľových hodnôt (3.2), kde sa nevyberá najlepšia akcia podľa cieľovej siete (s parametrami θ_{t-1}), ale je vybraná akcia podľa online siete (s parametrami θ_t). Takto agent pri počítaní cieľových hodnôt zohľadňuje akciu aktuálnej politiky.

$$y_t = r_t + \gamma \cdot Q^\pi(o_{t+1}, \underset{a}{\operatorname{argmax}}\, Q^\pi(o_{t+1}, a|\theta_t)|\theta_{t-1}) \tag{3.2}$$

3.2.3 Dueling DQN

Ďalším vylepšením pôvodného DQN agenta je Dueling DQN agent [17], ktorého prediktívny model má dve výstupné vrstvy predikujúce hodnotu stavu $V^\pi(o_t|\theta_\beta)$ a výhodnosť akcie $A^\pi(o_t, a_t|\theta_\alpha)$ vzhľadom na hodnotu pozorovaného stavu o_t, resp. predikovanú hodnotu stavu $V^\pi(o_t|\theta_\beta)$. Výhodnosť akcie sa počíta ako rozdiel q hodnoty a hodnoty stavu $A^\pi(o_t, a) = Q^\pi(o_t, a) - V^\pi(o_t)$. Po odvodení teoretického vzorca sa výsledné q hodnoty počítajú rovnicou (3.3). V rovnici je od výhodnosti akcie odčítaná priemerná hodnota výhodností aktuálneho stavu, pretože pôvodná rovnica nie je identifikovateľná.

$$Q^\pi(o, a) = V^\pi(o|\theta_\beta) + \left(A^\pi(o, a|\theta_\alpha) - \frac{1}{|A|}\sum_{a' \in A} A^\pi(o, a'|\theta_\alpha)\right) \tag{3.3}$$

Dueling DQN agent dosahuje v kombinácii s prioritným zoznamom skúseností veľmi priaznivé výsledky a v doméne Atari hier dokázal vo väčšine úloh poraziť DDQN agenta [17].

3.3 Politika gradientu

Ďalším populárnym prístupom v oblasti RL je politika gradientu [18]. Agent na výstupe NN predikuje pravdepodobnostné rozdelenie akcií $P(o, \cdot|\theta)$, na rozdiel od q hodnôt. Následne vykoná akciu vzhľadom na predikované pravdepodobnosti jednotlivých akcií, takže NN s parametrami θ je plne zodpovedná za politiku π_θ, resp. za pravdepodobnostné rozdelenie $\pi(o, \cdot|\theta)$. Riešenie základného prieskumu stavového priestoru je zakomponované priamo vo výstupe NN.

Predikovanie pravdepodobností je výhodné v rôznych situáciách:

1. Ak je aproximácia správnych q hodnôt náročná vzhľadom na husté rozloženie odmien v prostredí
2. Ak je aproximácia správnych q hodnôt náročná vzhľadom na vysoký počet akcií
3. Ak prostredie disponuje čiastočne pozorovanými stavmi, v ktorých je vhodné vykonať aspoň dve akcie

Prístup politiky gradientu sa využíva aj v prostrediach so spojitými akciami (napr. v robotike), kde výstupná vrstva NN predikuje numerické hodnoty konkrétnych akcií, napr. uhol otočenia a pod. Avšak v tejto práci sa zaoberáme len politikou gradientu predikujúcou pravdepodobnosti akcií.

$$J(\theta) = \underset{\theta}{\operatorname{argmax}} \sum_{\tau} [P(\tau|\theta) \cdot R(\tau)] \tag{3.4}$$

Cieľom politiky gradientu (rovnica 3.4) je nájsť parametre θ, ktoré maximalizujú pravdepodobnosti $P(\tau|\theta)$ trajektórií τ, vedúce k dosiahnutiu čo najväčšieho súčtu odmien $R(\tau) = \sum_{t=0}^{T} R(s_t, a_t | s_t \in \tau, a_t \in \tau)$. Preto pre úpravu parametrov θ sa aplikuje metóda najväčšieho rastu.

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \sum_{\tau \in X} [P(\tau|\theta) \cdot R(\tau)] \\
&= \sum_{\tau \in X} [\nabla_\theta P(\tau|\theta) \cdot R(\tau)] \\
&= \sum_{\tau \in X} \left[\frac{P(\tau|\theta)}{P(\tau|\theta)} \cdot \nabla_\theta P(\tau|\theta) \cdot R(\tau) \right] \\
&= \sum_{\tau \in X} \left[P(\tau|\theta) \cdot R(\tau) \cdot \frac{\nabla_\theta P(\tau|\theta)}{P(\tau|\theta)} \right] \\
&= \sum_{\tau \in X} [P(\tau|\theta) \cdot R(\tau) \cdot \nabla_\theta \log P(\tau|\theta)]
\end{aligned} \tag{3.5}$$

Agent pracuje s dátami z prejdených trajektórií (vstupná množina X) podľa aktuálnej politiky π_θ. Aplikovaním gradientov vypočítaných vzhľadom na hodnotu účelovej funkcie (rovnica 3.5) sa zvyšujú pravdepodobnosti trajektórií s vysokou odmenou a znižujú pravdepodobnosti s odmenou nízkou.

Mnoho algoritmov z oblasti politiky gradientu pracuje na princípe tzv. "on-policy", kde sa prediktívny model učí na vzorke zozbieraných dát z práve aktuálnej politiky, takže sa nevyužíva zoznam skúseností, ako v q učení. Kvôli zníženiu efektu korelovaných dát je potrebné zozbierať viac, ako len jednu súvislú trajektóriu.

3.3.1 Posilňujúci algoritmus (REINFORCE)

Základným algoritmom politiky gradientu je posilňujúci algoritmus (alg. 9), ktorý využíva sumu budúcich zrážaných odmien G. Na základe tejto sumy zvyšuje, resp. znižuje pravdepodobnosti vykonaných akcií. Pseudokód je pre jednoduchosť znázornený s jednou vykonanou trajektóriou počas epizódy, avšak ako sme vyššie uviedli, v praxi je potrebné zozbierať dáta z viacerých trajektórií.

Algoritmus 9 Posilňujúci algoritmus

Inicializuj parametre θ

Nastav počet epizód K

for $i = 0; i < K; i = i + 1$ **do**

Získaj trajektóriu $\tau_i = \{o_{i,0}, a_{i,0}, r_{i,0}, ..., o_{i,T}\}$ podľa politiky π_θ

$G_T \leftarrow 0$

for $t = T_i - 1; t \geq 0; t = t - 1$ **do**

$G_t \leftarrow r_t + \gamma \cdot G_{t+1}$

end for

Aplikuj pokles gradientu podľa $J(\theta) = \frac{1}{T_i - 1} \cdot \sum_{t=0}^{T_i} [G_t \cdot log\ \pi_\theta(o_t, a_t)]$

end for

3.3.2 A2C

Nevýhodami posilňujúceho algoritmu sú vysoký rozptyl a absencia porovnania výhodnosti vykonanej akcie voči ostatným akciám v aktuálnom stave. Tento problém rieši A2C agent (Advantage Actor-Critic) [19], ktorý okrem pravdepodobností predikuje aj hodnotu stavu $V^\pi(o|\theta_v)$. Pri učení výstupnej vrstvy pravdepodobností (nazývanej aktor) sa miesto jednoduchej sumy budúcich zrážaných odmien G aplikuje výhodnosť vykonanej akcie, počítaná pomocou rovnice (3.6). V praxi sa výhodnosť ráta pomocou metódy GAE [20].

$$A^\pi(o_t, a) = [r_t + \gamma \cdot V^\pi(o_{t+1}|\theta_v)] - V^\pi(o_t|\theta_v) \tag{3.6}$$

Pri učení výstupnej vrstvy predikujúcej hodnotu stavu (nazývanej kritik) sa aplikuje priemerná štvorcová chyba znížená o koeficient $\beta_v \in [0, 1]$. V algoritme (10) sme ju zapísali ako A^2.

Algoritmus 10 A2C

Inicializuj parametre θ a θ_v

Nastav počet epizód K a koeficienty β_v a β_e

Nastav počet krokov trajektórie k a počet trajektórií M

for $i = 0; i < K; i = i + 1$ **do**

  for $m = 0; m < M; m = m + 1$ **do**

    Získaj trajektóriu $\tau_m = \{o_{m,0}, a_{m,0}, r_{m,0}, ..., o_{m,k}\}$ podľa politiky π_θ

    $G \leftarrow V(o_{m,k}|\theta_v)$

    for $t = k - 1; t \geq 0; t = t - 1$ **do**

      $G \leftarrow r_{m,t} + \gamma \cdot G$

      $A_{m,t} \leftarrow G - V(o_{m,t}|\theta_v)$

    end for

  end for

  $J(\theta) \leftarrow \frac{1}{k \cdot M} \sum_{m=0}^{M-1} \sum_{t=0}^{k-1} [A_{m,t} \cdot log\ \pi_\theta(o_{m,t}, a_{m,t}) + \beta_v \cdot A^2 + \beta_e \cdot H(\pi_\theta(o_t, \cdot))]$

  Aplikuj pokles gradientu podľa $J(\theta)$

end for

Pre podporu prieskumu stavového priestoru sa do účelovej funkcie aplikuje entropia pravdepodobností predikovaných akcií v danom stave $H(\pi_\theta(o_t, \cdot))$. Entropia je znížená koeficientom $\beta_e \in [0, 1]$.

Prostredie môže poskytovať dlho trvajúce epizódy a tak sa berie iba časť získanej trajektórie, napr. k krokov. Pri počítaní sumy budúcich zrážaných odmien s ohľadom na celú dĺžku trajektórie $TD(1)$ sa začína hodnotou stavu, ktorý nasleduje po trajektórií, resp. $G = V^\pi(o_{t+k}|\theta_V)$, za predpokladu, že $s_{t+k} \neq s_T$, v opačnom prípade $G = 0$.

Agent A2C je úpravou agenta A3C (Asynchronous Advantage Actor-Critic) [19], ktorý je založený na paralelizovaní asynchrónnych agentov, pôvodne určených pre distribuované učenie na viacerých procesoch, resp. zariadeniach.

3.3.3 PPO

Hoci algoritmy A2C a A3C dosahujú zaujímavé výsledky, sú neefektívne vzhľadom na počet vzoriek, pretože zozbieranú vzorku dát "zahodia" po výpočte účelovej funkcie a aplikovaní spätného šírenia chyby. Zároveň ich tréning je často vysoko nestabilný a

tak je potrebné používať rôzne stabilizačné metódy, ako napr. normovanie gradientu.

Oba uvedené problémy rieši PPO agent [21], ktorý je vylepšením predchádzajúcich dvoch algoritmov. Ten zozbiera väčšie množstvo dát (oproti A2C), pričom dáta používa ako malý dataset. Na datasete vykoná E trénovacích epoch (v každej epoche sa učí NN zo vzoriek datasetu), takže agent sa učí na zozbieraných dátach opakovane.

Opakujúce sa učenie na jednej entite v stave s, resp. v pozorovanom stave o a akcií a je možné vďaka počítaniu pomeru medzi novou a pôvodnou hodnotou predikovanej pravdepodobnosti $\frac{\pi(o,a|\theta)}{\pi(o,a|\theta_{old})}$. Pomer pravdepodobností sa použije ako náhrada aktuálnej predikovanej pravdepodobnosti vybranej akcie. Tento krok je nevyhnutný, pretože získaná výhoda bola vypočítaná parametrami starej politiky $\pi_{\theta_{old}}$, resp. $A^{\pi_{\theta_{old}}}(o,a)$.

Kvôli dosiahnutiu vyššej stability je pomer pravdepodobností upravený do intervalu $[1-\epsilon, 1+\epsilon]$ (operáciou orezania), pričom ϵ je číslo z intervalu $[0,1]$. Chybová funkcia politiky sa získa výberom najnižšej hodnoty medzi pomerom a upraveným pomerom vynásobenými výhodou $A^{\pi_{\theta_{old}}}(o,a)$ (rovnica 3.7).

$$L(\theta)_{policy} = \frac{1}{T} \cdot \sum_{t=0}^{T-1} \left[\min \left(\frac{\pi(o_t, a_t|\theta)}{\pi(o_t, a_t|\theta_{old})} \cdot A^{\pi_{\theta_{old}}}(o_t, a_t), clip(\frac{\pi(o_t, a_t|\theta)}{\pi(o_t, a_t|\theta_{old})}, 1-\epsilon, 1+\epsilon) \cdot A^{\pi_{\theta_{old}}} \right) \right] \tag{3.7}$$

Ďalším podobným rozšírením s cieľom dosiahnuť vyššiu stabilitu tréningu je TRPO agent [22].

Kapitola 4

Modelovo založené algoritmy

Vo väčšine modelovo založených algoritmov môžeme model prostredia chápať učiaci sa prediktívny model (napr. hlboká NN), ktorý sa snaží aproximovať tranzitný model prostredia v podobe predikovania budúceho stavu, resp. prípadne tenzoru príznakov s_{t+1}. Model taktiež môže aproximovať funkciu odmeny, prípadne informáciu o terminálových stavoch. Avšak v niektorých prípadoch môže model prostredia predstavovať akúsi kópiu pôvodného prostredia, resp. simulátora.

Modelovo založené algoritmy možno rozdeliť na dve skupiny. Prvá skupina algoritmov využíva model prostredia ako podporu rozhodovania - agent má možnosť nahliadnuť do budúcnosti a tým podporiť proces rozhodovania, resp. plánovať. Druhá skupina agentov využíva model prostredia za účelom podpory prieskumu stavového priestoru v prostrediach, ktoré sa vyznačujú riedkymi odmenami.

4.1 AlphaZero

Agent AlphaZero (AZ) [23] je ďalšou generáciou pôvodného algoritmu AlphaGo [2], ktorý dokázal poraziť najlepšieho hráča na svete v komplexnej hre GO. Oproti AlphaGO, AZ obsahuje sofistikovanejšiu architektúru NN obohatenú o reziduálne bloky, jeho politika nie je ovplyvnená ľudským úsudkom a bol testovaný aj v ďalších doménach, ako napríklad šach a shogi. Predchodca AlphaGO využíval na začiatku tréningu dataset zložený z majstrovských partií. AZ sa začína učiť od nuly hraním proti sebe samému. Postupne sa začne zlepšovať, čo vedie ku objaveniu komplexnejších stratégií. V pôvodných článkoch algoritmy získavali odmeny až na konci hry v závislosti od

výsledku (-1 za prehru, 0 za remízu a $+1$ za výhru).

Z hľadiska efektivity tréningu, predstavuje model prostredia využívaný agentom manuálne implementovanú kópiu herného simulátora a tak nie je potrebné ho aproximovať pomocou NN.

Neurónová sieť obsahuje dva výstupy (podobne ako algoritmy A2C a PPO). Prvý výstup predikuje hodnotu stavu $V^{\pi}(s|\theta)$ a druhý výstup vracia pravdepodobnostné rozdelenie vykonania akcií $P(s, \cdot|\theta)$.

Pri konečnom výbere akcií sa využíva metóda Monte Carlo prehľadávací strom (MCTS). Jedná sa o stromovú heuristiku, určenú pre výber ďalšieho ťahu, resp. akcie. Vrcholy stromu predstavujú jednotlivé stavy a hrany povolené akcie, ktoré je možné v stave vykonať. Koreňom stromu je aktuálny stav hry. Strom je postupne budovaný za pomoci modelu prostredia, preto je nevyhnutné, aby model prostredia dokázal spoľahlivo kopírovať tranzitívny model, odmenovú funkciu a terminálové stavy. Pri dosiahnutí uvedených podmienok agent disponuje úplnou informáciou o stave prostredia.

Ako sme vyššie uviedli, v štruktúre MCTS, vrcholy predstavujú stavy, preto je väčšina informácií vo vrchole označovaná notáciou s s. V dôsledku toho sme sa rozhodli spraviť výnimku a pri predikovaných hodnotách stavu uvádzať s miesto o, resp. $s_t = o_t$.

Každý vrchol si uchováva nasledovné informácie:

- stav prostredia s
- pravdepodobnostné rozdelenie povolených akcií $P^{\pi}(s, \cdot)$, získané odstránením ilegálnych akcií z predikovaného rozdelenia $P(s, \cdot|\theta)$
- vektor počtu navštívení povolených akcií $N(s, \cdot)$
- vektor q hodnôt povolených akcií $Q^{\pi}(s, \cdot)$
- Informácia o terminálovom stave

Pre získanie novej akcie je potrebné vykonať tzv. simulácie v strome. Každá simulácia obsahuje 4 nasledovné fázy:

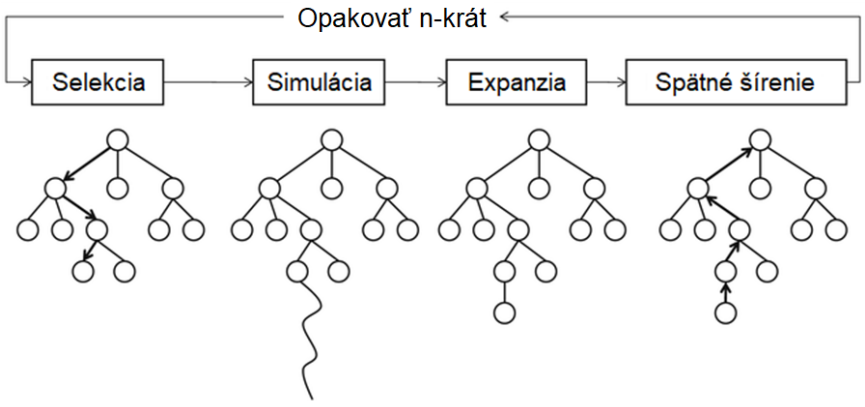
- Fáza selekcie - Strom je prechádzaný od koreňa až ku listu s cieľom nájsť doposiaľ nevykonanú akciu alebo terminálový stav. Akcia sa v každom vrchole vyberá podľa PUCT rovnice (4.1), ktorá definuje akýsi kompromis medzi výberom akcie s

najvyššou hodnotou a prieskumom. Ak sa na konci fáze selekcie nájde terminálový stav, simulácia pokračuje až fázou spätného šírenia.

- Fáza simulácie - Nájdená akcia sa vykoná za pomoci modelu prostredia (v stave rodičovského vrcholu s_{t-1}). Po vykonaní akcie agent obdrží nový stav s_t, ku ktorému predikuje rozdelenie pravdepodobností a hodnotu stavu. Predikované rozdelenie pravdepodobností je nutné upraviť zakázaním nepovolených akcií stavu a opätovne prepočítať.

- Fáza expanzie - Vytvorí sa nový vrchol so stavom s_t a vrcholu sú priradené ďalšie získané informácie. Vrchol je potom vložený do stromu pod rodičovský vrchol (so stavom s_{t-1}) s hranou označenou a_{t-1}.

- Fáza spätného šírenia - je potrebné aktualizovať q hodnoty všetkých vrcholov navštívených počas fázy selekcie podľa rovnice (4.2). Jedná sa o vážený priemer medzi odhadom q hodnôt z $N(s_t, a_t)$ navštívení a G odhadom na základe jedného navštívenia. Počiatočná hodnota G teda predstavuje predikovanú hodnotu stavu $V^\pi(s_t|\theta)$ v neterminálovom stave, alebo terminálovú odmenu, ak sa jedná o terminálový stav. G hodnota mení svoje znamienko podľa toho, či sa jedná o ťah agenta, resp. protihráča. Taktiež je potrebné navýšiť počet navštívení vykonaných akcií $N(s, a)$ o hodnotu 1.



Obrázok 4.1: Fázy MCTS

$$a_t = \underset{a}{\operatorname{argmax}} \left[Q^\pi(s_t, a) + c_{puct} \cdot P^\pi(s_t, a) \cdot \frac{\sqrt{\sum_{b \in A} N(s_t, b)}}{N(s_t, a)} \right] \tag{4.1}$$

$$Q^{\pi}(s_t, a_t) \leftarrow \frac{N(s_t, a_t) \cdot Q^{\pi}(s_t, a_t) + G}{N(s_t, a_t) + 1} \tag{4.2}$$

AZ vykoná 800 simulácií pred výberom akcie. Následne z početností vykonaných akcií v koreni $N(s_{root}, \cdot)$ vytvorí rozdelenie pravdepodobnosti (rovnica 4.3). Vo fáze tréningu sa akcia vyberie náhodne vzhľadom na získané pravdepodobnosti povolených akcií. V testovacej fáze sa vyberá akcia s najvyššou pravdepodobnosťou.

$$\pi(s_t, \cdot) = \frac{N(s_t, \cdot | s_t = s_{root})}{\sum_{b \in A} N(s_t, b)} \tag{4.3}$$

Výpočet chyby pre vrstvu predikujúcu hodnotu stavu je rovnaký ako v prípade algoritmov A2C a PPO (priemerná štvorcová chyba). Rozdiel nastáva pri výpočte chyby pravdepodobnostného rozdelenia, tá sa počíta ako $L_(\theta) = \pi(s, \cdot) \cdot logP(s, \cdot|\theta)$, pričom $\pi(s, \cdot)$ predstavuje pravdepodobnostné rozdelenie získané z MCTS stromu.

Vylepšenie architektúry poskytol open-source projekt LcZero (Leela Chess Zero) [24], ktorý predikuje rozdelenie pravdepodobností pre všetky možné odmeny $V^{\pi}(s, \cdot|\theta)$.

4.2 MuZero

V prípade absencie kópie simulátora, je AZ praktický nefunkčný. Problém rieši agent MuZero (MZ) [25] ktorý využíva hlbokú NN s cieľom aproximovať model prostredia. Jedná sa o novú generáciu AZ. MZ využíva až tri prediktívne funkcie a to funkciu politiky f, prezentačnú funkciu h a dynamickú funkciu g.

Prezentačná funkcia prijíma vstup z prostredia o_t a transformuje ho na stav s_t, resp. transformuje vstup z prostredia na dôležité príznaky interpretujúce stav prostredia v MCTS. Funkciu možno zapísať ako $s_t = h(o_t|\theta_h)$.

Vstupom do funkcie politiky je aktuálny stav, výstupom funkcie je pravdepodobnostné rozdelenie jednotlivých akcií $P(s, \cdot|\theta_f)$ a hodnota stavu $V^{\pi}(s|\theta_f)$. Funkciu možno zapísať ako $P(s, \cdot|\theta_f), V^{\pi}(s|\theta_f) = f(s|\theta_f)$.

Dynamická funkcia sa snaží aproximovať model prostredia s cieľom náhrady simulátora v algoritme MCTS. Na vstupe berie stav s_t a vykonanú akciu a_t. Výstupom funkcie je nasledujúci stav s_{t+1} a odmena za vykonanú akciu $\vec{r}_t$. Odmena je predikovaná ako rozdelenie pravdepodobností možných odmien hry. Funkciu možno zapísať ako $s_{t+1}, \vec{r}_t = g(s_t, a_t|\theta_g)$.

Počas simulácií v MCTS nie je možné zakázať ilegálne akcie vo vrcholoch s výnimkou koreňa. Avšak autori algoritmu ukázali, že MZ sa počas dlhšieho tréningu dokázal naučiť vykonávať prevažne povolené akcie.

MCTS využíva upravenú rovnicu PUCT (rovnica 4.4), ktorá kladie dôraz na dôkladnejšie prehľadávanie. Využíva dve konštanty, ktorých hodnoty v pôvodných experimentoch boli stanovené $c_1 = 1.25$ a $c_2 = 19652$ [25].

$$a_t = \underset{a}{\operatorname{argmax}} \left[Q(s_t, a) + P(s_t, a) \cdot \frac{\sqrt{\sum_b N(s_t, b)}}{1 + N(s_t, a)} \cdot \left(c_1 + \log \left(\frac{\sum_b N(s_t, b) + c_2 + 1}{c_2} \right) \right) \right] \tag{4.4}$$

Účelová funkcia politiky zostáva rovnaká ako pri AZ. Predikované rozdelenie pravdepodobnosti odmeny z dynamickej funkcie je trénované rovnakým spôsobom, ako predikované rozdelenie politiky (cieľová odmena sa transformuje do pravdepodobnostného rozdelenia). Prezentačná funkcia a časť dynamickej funkcie zodpovedajúcej za predikciu budúceho stavu sú trénované nepriamo a to síce s využitím prechádzajúcich gradientov z vyššie uvedenej účelovej funkcie. Dáta sa vyberajú z prioritného zoznamu skúseností. Jedna entita vo vzorke predstavuje trajektóriu dlhú 5 krokov, resp. $\tau = \{o_0, \pi(o_0, \cdot), a_0, r_0, ..., o_4, \pi(o_4, \cdot), a_4, r_4\}$.

Vďaka aproximácii modelu prostredia dynamickou funkciou, bolo možné nasadiť MZ aj v doméne Atari hier, v ktorej algoritmus dosiahol SOTA výsledky [25].

4.3 I2A

Ďalším z modelovo založených algoritmov je I2A (Imagination-Augmented Agent) [26]. Agent využíva predstavivosť za účelom preskúmania možných akcií. Pod pojmom predstavivosť sa myslí schopnosť agenta zahrať určitý počet trajektórii M v aproximovanom, resp. dynamickom modely prostredia. Získanie jednej trajektórie sa nazýva rolovanie. Algoritmus I2A sa skladá z nasledujúcich komponentov:

- Dynamický model prostredia - Model prostredia sa snaží predikovať nasledujúci stav a odmenu zo súčasného stavu a akcie. Model možno zapísať funkciou $s_{t+1}, r_{t+1} = g(s_t, a_t|\theta_g)$
- Funkcia rolovacej politiky - Metóda obsahuje dve funkcie pre politiku, ktoré

predikujú rolovaciu politiku π_{θ_p} a výstupnú politiku π_θ. Rolovacia politika π_{θ_p} sa používa pre rozhodovanie agenta počas rolovania. V prvom kroku rolovania berie funkcia skutočný pozorovaný stav z prostredia o_t a v ďalších krokoch berie len stavy predikované modelom prostredia (s_{t+1}, s_{t+2}, ...). Učenie rolovacej politiky prebieha len v skutočnom stave prostredia s_t pomocou krížovej entropie vzhľadom na skutočne vykonanú akciu a_t získanú z funkcie výstupnej politiky $\pi(s_t|\theta)$.

- Enkodér - Počas rolovania sa využíva tzv. enkodér, ktorý spracováva aktuálnu trajektóriu a kóduje ju do latentného priestoru príznakov. Enkodér pozostáva z RNN. Na vstupe berie predikovaný stav s_{t+1} a odmenu r_t počas každého kroku rolovania. Konečným výstupom enkodéra je kódovaná trajektória v tenzore e_{t+N}.

- Agregátor - Agregátor berie konečné výstupy z M kódovaných trajektórií a vytvára z nich výstupný zoznam $A = e_{0,t+N}, e_{1,t+N}, ..., e_{M-1,t+N}$

- Funkcia výstupnej politiky - Výstup z agregátora spolu so spracovaným aktuálnym stavom prijíma na vstupe funkcia výstupnej politiky π_θ. Funkcia počíta výstupnú politiku π_θ a hodnotu stavu $V^\pi(s|\theta)$. Učenie je rovnaké ako pri A2C agentovi.

4.4 SimPle

Ďalším významným modelovo založeným algoritmom je algoritmus SimPle (Simulated Policy Learning) [27]. Hlavným cieľom algoritmu je správna predikcia nasledujúcej obrazovky hry (v doméne Atari hier). Dynamický model prostredia dostane na vstupe aktuálny pozorovaný stav o_t a akciu a_t, z ktorých sa snaží predikovať obrazovku hry v ďalšom kroku v pixeloch i_{t+1}, spolu s odmenou za vykonanú akciu r_t. Ako sme vyššie uviedli, v doméne Atari hier sa zvyčajne pozorovaný stav skladá z posledný l obrazoviek, resp. $o_t = [i_t, i_{t-1}, ..., i_{t-l}]$.

Počas učenia sa ku dynamickému modelu pripája ďalšia NN, zodpovedná za tvorbu latentného priestoru cez RNN a "Attention" vrstvy. Táto pomocná NN dostáva prijíma na vstupe skutočný (cieľový) obraz hry v novom kroku.

Štruktúra dynamického modelu pozostáva z konvolučných vrstiev s veľkosťou filtra 4×4, ktoré majú za následok redukovať dimenzie vstupu až do tenzoru s rozmermi $256 \times 2 \times 2$. Do tenzoru vstupuje (za pomoci operácie násobenia) informácia o vybranej

akcii a počas učenia aj výstup z pomocnej neurónovej siete. Z latentného priestoru ďalej pokračujú tzv. dekonvolučné vrstvy, ktoré zväčšujú vstupnú informáciu na výstupe. Ku každému výstupu dekonvolučnej vrstvy je taktiež pomocou násobenia pridaná aj informácia o vybranej akcii. Výstupná vrstva predikujúca obraz pozostáva z plne prepojenej vrstvy. Tá predikuje každý pixel ako kategorickú úlohu. Ďalšou kategorickou úlohou je predikcia budúcej odmeny. Predikovaný výstup je tak presnejší, avšak za cenu vyššej výpočtovej náročnosti a počtu parametrov. Výstupom je teda RGB obraz s rozmermi $3 \times 105 \times 80$ a predikovaná odmena.

Za účelom výberu akcie sa vykonáva rolovanie s využitím PPO, ako rozhodovacieho agenta počas simulovaných trajektórií. Tréning PPO prebieha z aktuálnych vzoriek, pričom SimPle vyberá uložené vzorky z buffera.

Hoci algoritmus dosahuje zaujímavé výsledky a v niektorých prípadoch aj state-of-the-art skóre [27], je podobne ako MZ náročný na tréning. Len dynamický model obsahuje 74 miliónov parametrov. Napriek tomu poskytuje zaujímavé princípy v trénovaní dynamického modelu prostredia s využitím latentného priestoru, nakoľko predošlé výskumy učenia dynamického modelu s cieľom predikovania obrazu, nedosahovali tak vysokú presnosť predikcie [28], [29] [30].

4.5 ICM

Vyššie spomenuté modelovo založené algoritmy využívajú dynamický model prostredia pre podporu rozhodovania. Teraz by sme radi spomenuli algoritmy, ktoré využívajú dynamický model prostredia pre podporu prieskumu stavového priestoru prostredia. Podpora prieskumu je žiadúca najmä v prostrediach s riedkymi odmenami, resp. ak agent získa odmenu po vykonaní dlhšej sekvencie akcií.

Príkladom takýchto prostredí je Atari hra "Montezuma's Revenge". Hra obsahuje riedke odmeny, ktoré je možné získať až po vykonaní viacerých úkonov (napr. získať kľúč, odomknúť miestnosť, v miestnosti obísť nepriateľa a až potom zobrať odmenu). V tejto hre mali RL agenti problém s prieskumom prostredia, resp. nedokázali sa dostať ani z počiatočných miestností.

Riešením je podpora agenta na základe interných odmien, resp. odmien, ktoré nepochádzajú z prostredia. V terminológii sa ustálilo značenie r^i pre interné, resp.

vnútorné odmeny a r^e pre odmeny externé, resp. odmeny z prostredia. Externé odmeny teda závisia od odmenovej funkcie prostredia, rovnako ako v predchádzajúcich algoritmoch.

Interné odmeny sa získavajú pomocou učiaceho sa dynamického modelu prostredia $d(o, a|\theta_d)$, s parametrami θ_d. Zvyčajne sa snažia aproximovať extrahované vlastnosti budúceho stavu. Rozdiel medzi skutočnými a predikovanými hodnotami tvorí internú odmenu. Tá sa pripočítava ku externej odmene, ktorú následne obdrží agent. Tým dynamický model podporuje agentovu "zvedavosť" navštevovať stavy, ktoré nepozná, resp. ktoré nevie správne aproximovať.

Na takomto princípe funguje aj algoritmus ICM (Intrinsic Curiosity Module) [31]. Ten sa snaží aproximovať extrahované príznaky stavu s_t, získané z NN agenta, resp. $f(o_t|\theta_F)$. Parametrami θ_F budeme označovať podmnožinu parametrov modelu agenta θ, ktoré sú zodpovedné za extrakciu dôležitých vlastností z pozorovaného stavu. Vnútorná odmena je počítaná rovnicou (4.5), kde η je hyperparametrom algoritmu.

$$r_t^i = \frac{\eta}{2}\|f(o_t|\theta_F) - d(o_t, a_t|\theta_d)\|_2^2 \tag{4.5}$$

Cieľom prediktívneho modelu je teda minimalizácia rozdielu predikcie od skutočne extrahovaných hodnôt (rovnica 4.6).

$$J(\theta_d) = \underset{\theta_D}{\operatorname{argmin}} \frac{1}{|O \cdot A|} \sum_{o \in O} \sum_{a \in A} \|f(o|\theta_F) - d(o, a|\theta_D\|_2^2 \tag{4.6}$$

Pre správne fungovanie v ICM architektúre sa využíva aj ďalší model, tzv. inverzný model. Ten sa snaží predikovať vykonanú akciu a_t z pozorovaných stavov o_t a o_{t+1}, resp. z ich extrahovaných príznakov získaných pomocou parametrov θ_F. Úlohou inverzného modelu je zabezpečiť, aby predikované stavy obsahovali dôležité príznaky. V úlohách s diskrétnymi akciami tvorí účelovú funkciu krížová entropia, pri spojitých akciách sa jedná o L_2 normu.

ICM sa používa ako pomocný modul, ktorý je nutné pripojiť na rôzne typy algoritmov ako napríklad A2C, PPO a pod. V experimentoch bol agent s ICM modulom trénovaný v hrách "Super Mario" a "Sokoban". Problém nastal pri hre "Doom", kde sa na jednom mieste premietal náhodný zhluk informácií, ktorý agent, resp. modul nebol schopný predikovať a tak nastalo uviaznutie.

4.6 RND

Predikovanie extrahovaných vlastností z neustále učiaceho sa modelu agenta je náročné a nestabilné, pretože extrahované vlastnosti sa menia v závislosti od času a učenia. Vhodnejšou alternatívou je predikcia stálych, resp. konštantných hodnôt. Takéto riešenie poskytuje modul RND (Random Network Distillation) [32]. Ten motivuje agenta taktiež na princípe vnútornej odmeny, avšak oproti ICM sa líši v spôsobe jej predikcie.

Princíp RND pozostáva z dvoch sietí - cieľovej NN a učiacej sa NN. Cieľová sieť má náhodne inicializované váhy θ_g, ktoré sa počas tréningu nemenia. Extrahuje náhodné vlastnosti (vzhľadom na náhodne inicializované parametre) z aktuálne pozorovaného stavu $y_t = g(o_t|\theta_g)$.

Učiaca sieť $d(o|\theta_d)$ s parametrami θ_d sa snaží aproximovať výstup cieľovej siete. Cieľom je, podobne ako u ICM algoritmu, nájsť také parametre θ_d, ktoré minimalizujú rozdiel predikovaných hodnôt vzhľadom na výstupy cieľovej siete s nemennými parametrami θ_g (rovnica 4.7).

$$J(\theta_d) = \underset{\theta_D}{\operatorname{argmin}} \frac{1}{|O \cdot A|} \sum_{o \in O} \sum_{a \in A} \|g(o|\theta_g) - d(o|\theta_D\|_2^2 \tag{4.7}$$

Koncept bol testovaný na datasete MNIST, kde popri klasifikačnej sieti sa trénoval aj RND modul. Na konci tréningu boli RND modulu podsunuté nové dáta, pri ktorých sa merala chyba predikcie medzi cieľovou a učiacou sa sieťou cez MSE. Cieľom teda nebolo dosiahnuť vyššiu presnosť predikcie klasifikačného modelu, ale overenie teoretických predpokladov v praxi.

Keďže pixely v pozorovaných stavoch v rôznych prostrediach nadobúdajú rozdielne hodnoty, tak každý pixel je štandardizovaný. Preto je potrebné uchovať si štatistiku pre každú jednu entitu pozorovaného vstupu. Vnútorné odmeny sú normalizované a orezané do intervalu $[-5, 5]$.

Modul RND sa používa v kombinácii s PPO. V pôvodnom článku rozšírili NN agenta o ďalší výstup, predikujúci hodnotu stavu počítanú v interných odmenách $V_i^{\pi}(o_t|\theta_{v_i})$. Je teda dôležité oddeliť predikciu interných a externých odmien. Interné odmeny ovplyvňujú politiku agenta tak, že participujú na výpočte výhodnosti (rovnica 4.8). Jednotlivé výhodnosti sú ešte násobené koeficientmi β_e, resp. β_i.

$$
\begin{aligned}
A^{\pi}(o_t, a) = \beta_e \cdot [r_t^e + \gamma_e \cdot V_e^{\pi}(o_{t+1}|\theta_{v_e})] - V_e^{\pi}(o_t|\theta_{v_e}) \\
+\beta_i \cdot \left[r_t^i + \gamma_i \cdot V_i^{\pi}(o_{t+1}|\theta_{v_i})\right] - V_i^{\pi}(o_t|\theta_{v_i})
\end{aligned}
\tag{4.8}
$$

PPO v kombinácii s RND modulom dosiahol state-of-the-art v hrách ako "Montezuma's Revenge", "Gravitar" a "Venture" [32]. Skóre prekonal až algoritmus Go-Explore, ktorý je kombináciou učenia dynamického modelu a prieskumu za pomoci predstavivosti [33]. Nevýhodou Go-Explore algoritmu je nutnosť transformácie a uchovávania navštívených stavov za účelom kvalitnejšieho rozhodovania, čo si samozrejme vyžaduje obrovskú pamäťovú náročnosť. Zároveň nutnosť uchovávania si navštívených stavov je náročná v prostrediach s veľkým stavovým priestorom.

Kapitola 5

Ciele

Algoritmy učenia posilňovaním predstavené v predchádzajúcich kapitolách dosahujú zaujímavé výsledky, avšak potrebujú veľký počet tréningových dát - najmä algoritmy politiky gradientu napriek svojím vhodným konvergenčným vlastnostiam. Potreba veľkého počtu dát je problém predovšetkým pri absencii simulátora prostredia, resp. ak sa agent musí učiť z reálnych dát, napr. v zdravotníctve z dát popisujúcich liečbu pacientov. V takomto prípade môže byť získavanie nových dát drahé, časovo náročné a v niektorých prípadoch aj rizikové.

Z hľadiska dátovej efektivity sa ako najúspešnejšie javia práve modelovo založené algoritmy využívajúce model prostredia pre podporu rozhodovania (napr. MZ [34]). Takýto agenti majú viaceré problémy, napr. v prostrediach s riedkymi odmenami nedokážu efektívne preskúmavať stavový priestor, alebo v prostrediách so stochastickými prechodmi, ktoré môžu narúšať rozhodovací proces napr. v procese plánovania.

Napriek uvedeným problémom, stále vidíme v agentoch využívajúcich model prostredia pre podporu rozhodovania obrovský potenciál, práve preto sa ciele dizertačnej práce zameriavajú na modelovo založené algoritmy z oblasti učenia posilňovaním.

Ako sme už spomenuli, agenti využívajúci model prostredia pre podporu rozhodovania vo všeobecnosti nevedia efektívne preskúmavať stavový priestor a majú problémy v prostrediach s riedkymi odmenami. Na druhej strane agenti s modelom pre podporu prieskumu stavového priestoru nedosahujú v prostrediach s hustými odmenami také skóre ako agenti s modelom prostredia určeným pre podporu rozhodovania. Preto sa v našom prvom cieli dizertačnej práce chceme venovať práve tejto problematike.

Naším prvým cieľom dizertačnej práce je vytvorenie novej metódy mode-

lovo založeného agenta, ktorý bude využívať model prostredia pre podporu prieskumu stavového priestoru a zároveň aj pre podporu rozhodovania.

V takomto prípade musí agent pri podpore rozhodovania brať do úvahy aj vnútornú odmenu, resp. chybu predikcie. Tým vzniká nutnosť vytvorenia novej výstupnej vrstvy modelu prostredia, ktorá bude predikovať chybu predikcie nového stavu (ako pri RND), ktorá môže byť využitá ako vnútorná motivácia. Takéto využitie modelu prostredia na dva účely má potenciál zvýšiť úroveň politiky agenta.

Ďalší cieľ práce sa týka opäť modelovo založených algoritmov a to v súvislosti so schopnosťou agenta adaptovať sa na súperovú (slabšiu) úroveň. Výskumy adaptívnosti agenta sa realizovali najmä s využitím ďalšieho prediktívneho modelu, ktorý hodnotí agentove a súperove akcie. V našom prístupe využijeme modifikáciu MCTS metódy, resp. pôvodnej PUCT rovnice.

Druhým cieľom práce je vytvorenie metódy pre modelovo založený algoritmus pri tvorbe agenta so schopnosťou adaptovať sa na úroveň súpera.

5.1 Metodika dizertačnej práce

1. Analýza metód využívaných v učení posilňovaním.
2. Identifikácia vhodných domén a existujúcich simulátorov
3. Návrh a implementácia agenta so schopnosťou adaptovať sa na súpera.
4. Návrh a implementácia agenta s podporou prieskumu stavového priestoru.
5. Overenie správnosti a efektivity vytvorených riešení.
6. Dokumentácia dosiahnutých výsledkov.

Kapitola 6

Adaptívnosť modelovo založeného algoritmu

Nedávne úspechy modelovo založených algoritmov, ako napríklad porážka svetového šampióna v hre GO algoritmom AlphaGO či porážka šachového programu Stockfish algoritmom AZ sú významnými míľnikmi strojového učenia a umelej inteligencie, avšak "bežným" hráčom nepriniesli žiadny významný prínos. Predpokladáme, že bežný hráč by nemal záujem hrať s umelou inteligenciou, s ktorou by všetky hry jednoznačne prehral. Z tohto dôvodu vznikol nápad prispôsobiť úroveň náročnosti naučeného agenta súperovej (horšej) stratégii.

V oblasti RL výskumov spojených s adaptívnosťou, v hrách s nulovým súčtom, identifikujeme dva hlavné prístupy:

V prvom prístupe hrá agent proti čoraz silnejším oponentom s cieľom adaptovať sa na aktuálny level súpera a poraziť ho. Po výhre agenta sa zvyčajne zmení súper a agent hrá opäť proti novému, silnejšiemu súperovi [35]. V komplexných prostrediach s veľkým počtom rôznych stratégií, je vhodné, ak agent hrá proti viacerým súperom s rôznymi stratégiami [36]. Cieľ je opäť rovnaký a to adaptovať sa na rôznych oponentov. Je zrejmé, že v prvom prístupe je adaptívnosť použitá na silnejších súperov s cieľom zlepšiť politiku agenta.

V druhom, resp. opačnom prístupe, agent hrá proti slabším súperom s cieľom vyrovnať obťažnosť hry. Vo výskumoch agent môže hrať proti slabším NN [37] alebo proti iným vyhodnocovacím funkciám [38]. V oboch spomenutých výskumoch je nutné na konci tréningu agenta naučiť ďalší prediktívny model, ktorý vyhodnocuje a porovnáva

úroveň agenta a oponenta, pričom zasahuje do politiky výberu akcií. Experimenty oboch metód boli vykonávané v pomerne jednoduchých prostrediach s malým počtom akcií a absenciou významných budúcich následkov ako môžeme objaviť v stolných hrách (chybný ťah môže zvrátiť priebeh hry).

Náš výskum je zameraný na druhý prístup, t.j. cieľom naučeného agenta je zámerne znížiť svoju výkonnosť na úroveň protihráča, aby hra trvala dlhšie, resp. vyrovnanejšie. Vychádzame z predpokladu, v ktorom má bežný hráč najväčší úžitok z hry, ak hrá so súperom na približne rovnakej úrovni. Inými slovami, hráč môže mať nízky úžitok z hry vtedy, ak všetky partie výrazne prehrá (súper je príliš silný) alebo vyhrá (súper je príliš slabý).

Takto by bolo umožnené priemerným hráčom hrať zaujímavé partie s možnosťou učenia sa nových stratégií a prístupov od adaptívneho agenta, bez nutnosti nastavovania obťažnosti.

Zníženie úrovne natrénovaného agenta docielime prostredníctvom zámerného predlžovania hry týmto agentom. Zámerné predĺženie je súčasťou rozhodovacieho procesu agenta, čím je mu znemožnený výber len najlepších akcií. Predpokladáme, že ak agent bude hrať proti skúsenejšiemu oponentovi, tak bude vyberať sofistikovanejšie akcie s cieľom predĺžiť hru. Na druhej strane ak bude agent hrať proti začiatočníkovi, môže robiť výraznejšie chyby, aby docielil predĺženie hry.

Vzhľadom na výpočtovú náročnosť tréningu MZ, sme sa rozhodli použiť algoritmus AZ, pri ktorom netreba trénovať dynamický model prostredia. Taktiež kvôli výpočtovej zložitosti sme zvolili hru TicTacToe.

6.1 Popis metódy

Pred výberom každej akcie, AZ vykonáva proces plánovania, ktorý pozostáva z MCTS algoritmu, hlbokej NN a modelu prostredia. Výber akcie AZ závisí od výsledného rozdelenia pravdepodobnosti získaného z MCTS. Proces budovania stromu závisí od PUCT rovnice (4.1), ktorá vyberá akcie počas fáze selekcie. Vo výskume sme sa zamerali na modifikáciu rovnice PUCT.

V rámci terminológie by sme radi zaviedli dva pojmy:

- Originálna stratégia - jedná sa o pôvodnú stratégiu výberu akcie pomocou PUCT

rovnice

- Predlžovacia stratégia - navrhnutá stratégia výberu akcie za účelom umelo predĺžiť hru (rovnica 6.2)

Pred tréningom AZ je nutné pridať novú výstupnú vrstvu $M^\pi(s, \cdot|\theta)$, ktorá má za úlohu predikovať počet krokov do konca hry podľa originálnej stratégie π. Vrstva predikuje vektor pravdepodobnostného rozdelenia počtu krokov do konca hry. Inšpirovali sme sa projektom LcZero [24], ktorý výstupnú vrstvu využíva za účelom zrýchlenia tréningu. V nami navrhnutej modifikácii využívame výstup novej vrstvy ako dôležitú podporu pri rozhodovaní.

S novou výstupnou vrstvou je potrebné rozšíriť aj dátovú štruktúru uchovávajúcu dáta vrcholov o priemerný počet krokov z vrcholu, resp. stavu do konca hry $L(s, a)$, pre všetky povolené akcie. Tento vektor sa bude upravovať vo fáze spätného šírenia, podobne ako q hodnoty (rovnica 6.1), kde M predstavuje predikovanú hodnotu $M^\pi(s, \cdot|\theta)$ inkrementovanú o 1 v každom rodičovskom vrchole v trajektórii od listu ku koreňu.

$$L(s_t, a_t) = \frac{N(s_t, a_t) \times L(s_t, a_t) + M}{N(s_t, a_t) + 1} \tag{6.1}$$

Pre potreby výskumu sme zadefinovali predlžovaciu stratégiu rovnicou (6.2). Stratégia výberu akcie zahŕňa očakávanú q hodnotu a normalizovaný očakávaný počet krokov do konca hry v intervale $[0, 1]$, vynásobený parametrom metódy β. Q hodnotu sme do rovnice aplikovali za účelom zamedziť agentovi vykonávať výrazne nevýhodné akcie, ktoré môžu rýchlo viesť ku jeho porážke. Očakávaný počet krokov do konca hry sme normalizovali maximálnym počtom krokov do konca hry (inak by boli L hodnoty na začiatku hry príliš vysoké a na konci príliš nízke). V prípade, ak je maximálny počet krokov neznámy, vývojári LcZero uprednostňujú použiť vyšší počet krokov, ako priemerná dĺžka hry [24]. Parameter $\beta > 1$ zvyšuje dôležitosť vykonávania akcií predlžujúcich hru voči q hodnotám.

$$C(s_t, a) = Q(s_t, a) + \beta \times L_{norm}(s_t, a) \tag{6.2}$$

Avšak aj po pridaní q hodnôt do predlžovacej stratégie nastávali hry (trajektórie), v ktorých často absentoval výber najlepších akcií ako aj prehľadávania iných možností v MCTS (agent s cieľom predĺžiť hru sa dopúšťal výrazných chýb alebo preferoval len

jednu akciu v danom stave). Preto sa nami navrhnuté metódy adaptívnosti agenta zaoberajú pomerom, resp. kompromisom medzi originálnou a predlžujúcou stratégiou. Navrhli a testovali sme dve metódy - kompromis na základe počtu krokov do konca hry a kompromis na základe hodnotenia akcií.

6.1.1 Kompromis na základe počtu krokov do konca hry

Prvá metóda kompromisu je akýmsi preklápaním rozhodovacieho procesu z predlžovacej stratégie do originálnej. Výber akcie závisí od parametra α, počítaného rovnicou (6.3). Ak je maximálny počet krokov hry neznámy, resp. môže byť nekonečný, odporúčame opäť použitie počtu krokov, ktorý je vyšší ako priemerná dĺžka hry.

$$\alpha = \frac{počet\,krokov\,od\,začiatku\,hry}{maximálny\,počet\,krokov\,hry} \tag{6.3}$$

Výber akcie vo fáze selekcie sa počíta pomocou rovnice (6.4). Na začiatku hry je hodnota parametra α nízka, preto agent umelo predlžuje hru. Zvyšovaním počtu krokov sa zvyšuje hodnota α a tak agent začína preferovať originálnu stratégiu. Napríklad v šachu alebo GO môže byť zvýšenie náročnosti dôležité, pretože umelé predlžovanie hry môže pôsobiť pre ľudského súpera únavne.

$$a_t = \underset{a}{\operatorname{argmax}} \left[\alpha \times U(s_t, a) + (1 - \alpha) \times C(s_t, a)\right] \tag{6.4}$$

Hoci kompromis na základe počtu krokov do konca hry môže pôsobiť pomerne jednoducho, jeho potenciál vidíme napr. v šachu. Na začiatku hry má oponent výhodu a tak pomocou dôvtipných stratégií môže agenta obrať o dôležité figúrky. V priebehu času začne agent viac uprednostňovať originálnu stratégiu a tým využívať komplexnejšie stratégie vedúce ku získaniu odmien, resp. ku výhre. Avšak už disponuje menším počtom figúrok a tak v polovici hry začína pre bežného hráča zaujímavá výzva, v zmysle poraziť (oslabeného) neustále sa zlepšujúceho hráča.

V metóde sa hlavný význam adaptívnosti skrýva prevažne v prvej polovici hry a to síce v schopnosti agenta reagovať na súperove ťahy s cieľom zámerného predlžovania hry, resp. agent hľadá také akcie, aby hru predĺžil a zároveň neprehral v krátkom čase.

6.1.2 Kompromis na základe hodnotenia akcií

Druhý spôsob kompromisu zohľadňuje vykonané akcie agenta a súpera pomocou ohodnotenia. Každý vykonaný ťah v hre (agentov aj súperov) je ohodnotený pomocou originálnej stratégie výberu akcií. Pôvodné hodnotenie vychádzalo z q hodnôt, ktoré sa napokon neosvedčili ako vhodný ukazovateľ, pretože dokázali spoľahlivo predikovať q hodnoty len vo frekventovane navštevovaných stavoch.

Preto sme si ako hodnotiacu veličinu zvolili získanú pravdepodobnosť vykonanej akcie podľa MCTS, resp. sumu posledných N pravdepodobností vykonaných akcií. Ak $N = 1$, tak berieme do úvahy len pravdepodobnosť poslednej vykonanej akcie. Na druhej strane, ak $N > 1$, tak berieme do úvahy sumu pravdepodobností posledných N vykonaných akcií. U agenta definujeme sumu posledných N pravdepodobností vykonaných akcií ako parameter A_N a u súpera O_N. Metóda kompromisu je definovaná rovnicou (6.5). Parametre ohodnotení, vydelené ich spoločnou sumou, slúžia ako váhy jednotlivých stratégií. Ak agent vykonal posledných N akcií výrazne horších oproti súperovi, tak zlomok $\frac{O_N}{A_N+O_N}$ dosahuje vyššiu hodnotu a v ďalšom ťahu bude agent preferovať originálna stratégia U. V opačnom prípade, ak agent zahral posledných N výrazne lepších akcií ako súper, tak v ďalšom ťahu bude nútený zhoršiť rozhodovanie, pretože $\frac{A_N}{A_N+O_N} > \frac{O_N}{A_N+O_N}$.

$$a_t = \underset{a}{\operatorname{argmax}} \left[\frac{O_N}{A_N + O_N} \times U(s_t, a) + \frac{A_N}{A_N + O_N} \times C(s_t, a) \right] \tag{6.5}$$

6.2 Experimenty

Experimenty navrhnutých metód kompromisu vyžadovali implementáciu vybraného prostredia a agenta AZ, trénovanie širokej škály súperov s odlišným nastavením parametrov a architektúrou NN, vzájomné porovnanie natrénovaných agentov a testovanie navrhnutých metód najsilnejším agentom voči ostatným.

6.2.1 Prostredie

Vzhľadom na náročnosť tréningu sme sa rozhodli implementovať prostredie TicTacToe s parametrami:

- Veľkosť hernej plochy 5×5

- Potrebný počet rovnakých symbolov v rade (na výhru) 4

Za pomoci tenzorov a knižnice PyTorch sme celé prostredie implementovali na GPU. Kontrola výhry sa vykonáva pomocou dvojrozmerných matíc - filtrov. Remíza je kontrolovaná pomocou sčítania obsadených políčok. Takáto implementácia umožňuje paralelné vykonanie akcií vo viacerých hrách naraz. V našom prípade sme tak boli schopný zrýchliť vykonávanie MCTS za pomoci GPU prostredia.

Funkcia odmeny prostredia vráti odmenu na konci hry, $+1$ za výhru, 0 za remízu a -1 za prehru.

S cieľom podporiť prehľadávanie stavového priestoru, sme sa rozhodli 50% nových hier začínať s náhodne zabraným políčkom.

6.2.2 Modifikácia implementácie

Pôvodný AZ agent pozostáva z dvoch fáz vykonávaných paralelne - zber dát a učenie algoritmu. Experimenty v pôvodnom článku o AZ boli hardvérovo podporené vyšším počtom výpočtových zariadení [23]. V našom prípade sme mali k dispozícií len jedno zariadenie, resp. jednu grafickú kartu, čo viedlo k neefektívnemu vykonávaniu oboch fáz súčasne. Neustále požiadavky na predikciu z viacerých procesov zberajúcich dáta v kombinácii s učením modelu znížili výkon GPU takmer 10-násobne. Dôsledkom vzniknutého problému sme uprednostnili sekvenčné vykonávanie fáz, resp. po fáze zberu dát (hraní hier proti sebe samému na viacerých procesoch paralelne) nasleduje fáza učenia.

Pri sekvenčnom vykonávaní jednotlivých fáz, vznikol problém s vysokou nestabilitou tréningu. Tento problém sme vyriešili pridaním tzv. arény, pri ktorej sme sa inšpirovali algoritmom AlphaGO. Fáza arény nasleduje po fáze učenia a porovnáva aktuálne naučenú sieť s doposiaľ najlepšou NN na základe odohraných vzájomných partií. Vo vzájomných dueloch sa nevyberajú akcie s najvyššou q hodnotou, ale náhodne vybrané akcie vzhľadom na získané rozdelenia pravdepodobností z MCTS. Ak aktuálne naučená NN porazí doposiaľ najlepšiu v pomere aspoň 65%, tak sa doposiaľ najlepšia NN aktualizuje parametrami aktuálnej NN. Doposiaľ najlepšia NN sa používa na zber dát s cieľom zabezpečiť neklesajúcu úroveň partií.

S nasadením predošlých modifikácií sa objavil problém súvisiaci s predikciou pravdepodobnostného rozdelenia akcií $P(s, \cdot|\theta)$. Počas tréningu, vo väčšine stavov totiž

dochádzalo ku presileniu práve jednej z povolených akcií, čím akcia dosahovala pravdepodobnosť vykonania 100%. Tým pádom PUCT metóda stratila vo väčšine krokov význam a zozbierané dáta obsahovali podobné trajektórie. Ako riešenie sa nám osvedčilo vo fáze selekcie navštíviť každú akciu v aktuálnom koreni MCTS, čím sa posilnil efekt PUCT metódy a tak konečné rozdelenie pravdepodobností z MCTS poskytovalo viac možností s vyššími pravdepodobnosťami.

Poslednou významnou modifikáciou bolo osemnásobné rozšírenie zozbieraných dát pomocou rotácií a zrkadlenia rotácií pozorovaného stavu a cieľovej politiky z MCTS.

6.2.3 Agent

Počas trénovania agentov sme vykonali desiatky rôznych tréningov. Úspešne sa nám podarilo naučiť až 26 rôznych agentov, avšak z informatívneho hľadiska považujeme za dôležité opísať parametre agenta s najvyššou dosiahnutou úrovňou, ktorého parametre sú:

- Koeficient rýchlosti učenia - počiatočná hodnota 0.01, koncová hodnota 0.0001
- Dirichletov šum - počiatočná hodnota 0.3, koncová hodnota 0.03 (z pôvodného článku [23])
- L_2 norma 0.0001
- Počet iterácií 1000
- Počet procesov 8
- Počet hier na jednom procese počas iterácie 200
- Veľkosť vzorky 15000
- Počet vzoriek počas iterácie 200
- Veľkosť zoznamu skúseností 1000000
- Počet simulácií MCTS 200

Tréning najlepšieho agenta trval na nami dostupnom výpočtovom zariadení cez 4 dni. Použitý hardvér pozostával z grafickej karty NVIDIA GeForce RTX 2080 Ti a 16

jadrového procesoru Intel(R) Xeon(R) CPU E5-2643 0 @ 3.30GHz. V neurónových sieťach sme používali filtre s rozmerom 3×3 a aktivačné funkcie ReLU. Architektúra NN najlepšieho agenta sa skladá z:

- Konvolučná vrstva s počtom filtrov 96
- 3× reziduálny blok s počtom filtrov 96
- Výstupná hlava politiky:
  - Konvolučná vrstva s počtom filtrov 32
  - Plne prepojená vrstva s počtom neurónov 256
  - Plne prepojená vrstva s počtom neurónov 25
- Výstupná hlava hodnoty stavu:
  - Konvolučná vrstva s počtom filtrov 32
  - Plne prepojená vrstva s počtom neurónov 128
  - Plne prepojená vrstva s počtom neurónov 3
- Výstupná hlava predikujúca počet krokov do konca hry:
  - Konvolučná vrstva s počtom filtrov 32
  - Plne prepojená vrstva s počtom neurónov 256
  - Plne prepojená vrstva s počtom neurónov 25

Z architektúry siete vyplýva, že sme zmenili úlohy predikcie hodnoty stavu $V^{\pi}(s, \cdot|\theta)$ a počtu krokov do konca hry $L^{\pi}(s, \cdot|\theta)$ na úlohy klasifikácie (trénované pomocou krížovej entropie).

6.2.4 Výsledky

Najlepšie hrajúci agent hral 2 hry (v prvej hre začínal prvý a v druhej ako druhý hráč) proti 460 slabším agentom pozostávajúcich z 25 plne natrénovaných agentov a ich slabšie natrénovaných verzií zo skorších iterácií tréningov. Pokiaľ hral najlepší agent s originálnou stratégiou, tak neprehral ani jednu hru.

Zaujímavosťou bolo zopár agentov, ktorí prvú hru (začínali ako druhí) prehrali a druhú remizovali. Preto pre lepšie porovnanie metódy, sme sa rozhodli rozdeliť odohraných 920 partií do nasledujúcich skupín:

- 1. skupina - agent vyhral hru do 10 krokov (550 partií)
- 2. skupina - agent vyhral hru nad 10 krokov (123 partií)
- 3. skupina - agent remizoval hru (247 partií)

Metóda kompromisu na základe počtu krokov do konca hry

V metóde kompromisu na základe počtu krokov do konca hry sme testovali hodnoty parametru β. Prvý riadok tabuliek obsahuje dáta z partií podľa pôvodnej stratégie, resp. bez kompromisu.

Tabuľka 6.1: Výsledky 1. skupiny s využitím metódy kompromisu na základe počtu krokov do konca hry

Beta	Výhry	Prehry	Remízy	Priemerný počet krokov výhry / prehry / všetky hry
-	550	0	0	7.8 / - / 7.8
1.0	209	0	341	12.8 / - / 20.3
2.0	82	0	468	16.0 / - / 23.7
3.0	56	0	494	15.0 / - / 24.0
5.0	30	0	520	12.2 / - / 24.3
10.0	29	0	521	12.6 / - / 24.3
15.0	27	17	506	13.7 / 12.9 / 24.0
20.0	27	17	506	13.7 / 12.9 / 24.0
25.0	27	17	506	13.7 / 12.9 / 24.0

Z výsledkov prvej skupiny (tab. 6.1) možno vidieť, že nasadenie metódy kompromisu významne zmenilo výsledky partií už pri $\beta = 1$. Výrazne sa zvýšil aj priemerný počet krokov. V tabuľke možno vidieť úbytok počtu výhier vzhľadom na zvyšovanie hodnoty parametru β, čo pri vysokých hodnotách vedie aj ku nárastu prehier.

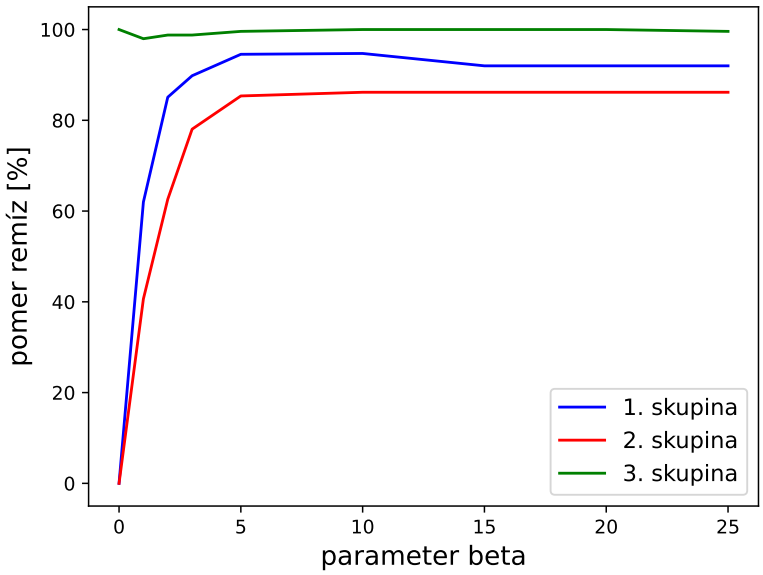
Tabuľka 6.2: Výsledky 2. skupiny s využitím metódy kompromisu na základe počtu krokov do konca hry

Beta	Výhry	Prehry	Remízy	Priemerný počet krokov výhry / prehry / všetky hry
-	123	0	0	12.5 / - / 12.5
1.0	60	13	50	12.4 / 18.0 / 18.1
2.0	46	0	77	14.5 / - / 21.0
3.0	27	0	96	18.1 / - / 23.5
5.0	18	0	105	16.8 / - / 23.8
10.0	17	0	106	16.6 / - / 23.8
15.0	17	0	106	16.6 / - / 23.8
20.0	17	0	106	16.6 / - / 23.8
25.0	16	1	106	19.0 / 9.0 / 24.0

Tabuľka 6.3: Výsledky 3. skupiny s využitím metódy kompromisu na základe počtu krokov do konca hry

Beta	Výhry	Prehry	Remízy	Priemerný počet krokov výhry / prehry / všetky hry
-	0	0	247	- / - / 25.0
1.0	5	0	242	16.4 / - / 24.8
2.0	3	0	244	14.7 / - / 24.9
3.0	3	0	244	16.0 / - / 24.9
5.0	1	0	246	20.0 / - / 25.0
10.0	0	0	247	- / - / 25.0
15.0	0	0	247	- / - / 25.0
20.0	0	0	247	- / - / 25.0
25.0	0	1	246	- / 13.0 / 25.0

Zaujímavý jav vznikol v druhej skupine (tab. 6.2), kde agent prehral 13 hier pri $\beta = 1$, avšak pri vyšších hodnotách ($1 < \beta < 25$) bol počet prehier nulový. Vysvetľujeme si to ako výraznú nezhodu medzi originálnou a predlžovanou stratégiou. Podobne ako v



Obrázok 6.1: Pomer remíz voči všetkým hrám pre všetky skupiny s využitím metódy kompromisu na základe počtu krokov do konca hry

prvej skupine, počet výhier klesal, resp. počet remíz stúpal so zvyšovaním parametru β.

V tretej skupine (tab. 6.3) tvorenej z remizovaných partií narástol počet výhier pri nízkych hodnotách β. Príčinou bol nepresný odhad predlžovanej stratégie v niektorých stavoch. Prekvapením bol nárast počtu prehier až od $\beta = 25$, ktorý sme očakávali podstatne skôr.

Počet remíz v závislosti od zvyšovania hodnoty parametru β je zobrazený v grafe (obr. 6.1). Z výsledkov vyplýva, že aplikácia metódy kompromisu úspešne zvýšila počet remíz (prípadne prehier), resp. predĺžila počet krokov výherných partií u väčšiny experimentov.

Metóda kompromisu na základe hodnotenia akcií

V metóde kompromisu na základe hodnotenia akcií sme skúmali rôzne hodnoty parametrov β a N. Vzhľadom na väčší počet kombinácií sme sa rozhodli preskúmať 3 hodnoty parametru β a 4 hodnoty parametru N.

Výsledky väčšiny hier prvej skupiny (tab. 6.4) sa zmenili z výhier na remízy. Zvyšovanie hodnoty parametru β má podobný efekt ako pri predchádzajúcich experimentoch - nárast počtu remíz a predĺženie hier.

Vo výsledkoch $\beta > 3$ možno pozorovať, že parameter N nemá významný vplyv

na pomer výhier, remíz a prehier. Jav je zapríčinený nepresným hodnotením akcií pomocou originálnej stratégie. Agent sa totiž počas tréningu vyhýba nevýhodným akciám pomocou znižovania ich pravdepodobností a tak nevie objektívne posúdiť, ktorá z nevýhodných akcií v konkrétnom stave je lepšia, resp. horšia. Preto pomer hodnotení je presný prevažne u preferovaných akcií.

V druhej skupine (tab. 6.5) agent remizoval takmer všetky hry už pri $\beta = 3$. Zmena hodnoty parametru N nepriniesla takmer žiaden efekt.

V tretej skupine (tab. 6.6) zostali výsledky hier nezmenené a tak sa zachoval dosiahnutý počet remíz.

Konštatujeme že, použitie samotnej metódy prinieslo priaznivé výsledky v podobe navýšenia počtu remíz a krokov.

Tabuľka 6.4: Výsledky 1. skupiny s využitím metódy kompromisu na základe hodnotenia akcií

Beta	N	Výhry	Prehry	Remízy	Priemerný počet krokov výhry / prehry / všetky hry
-	-	550	0	0	7.8 / - / 7.8
3.0	1	54	0	496	16.0 / - / 24.1
3.0	4	40	0	510	13.6 / - / 24.2
3.0	8	40	0	510	13.6 / - / 24.2
3.0	25	41	0	509	13.8 / - / 24.2
5.0	1	30	0	520	11.6 / - / 24.3
5.0	4	30	0	520	11.6 / - / 24.3
5.0	8	30	0	520	11.6 / - / 24.3
5.0	25	30	0	520	11.6 / - / 24.3
10.0	1	30	15	505	11.6 / 13.0 / 23.9
10.0	4	30	15	505	11.6 / 13.0 / 23.9
10.0	8	30	15	505	11.6 / 13.0 / 23.9
10.0	25	30	15	505	11.6 / 13.0 / 23.9

Tabuľka 6.5: Výsledky 2. skupiny s využitím metódy kompromisu na základe hodnotenia akcií

Beta	N	Výhry	Prehry	Remízy	Priemerný počet krokov výhry / prehry / všetky hry
-	-	123	0	0	12.5 / - / 12.5
3.0	1	2	0	121	17.0 / - / 24.9
3.0	4	1	0	122	10.0 / - / 24.9
3.0	8	1	0	122	10.0 / - / 24.9
3.0	25	1	0	122	10.0 / - / 24.9
5.0	1	1	0	122	10.0 / - / 24.9
5.0	4	1	0	122	10.0 / - / 24.9
5.0	8	1	0	122	10.0 / - / 24.9
5.0	25	1	0	122	10.0 / - / 24.9
10.0	1	1	0	122	10.0 / - / 24.9
10.0	4	1	0	122	10.0 / - / 24.9
10.0	8	1	0	122	10.0 / - / 24.9
10.0	25	1	0	122	10.0 / - / 24.9

6.3 Záver kapitoly

Ako sme na začiatku kapitoly uviedli, naším cieľom bol návrh metód, ktoré adaptujú úroveň natrénovaného agenta na nižšiu úroveň súpera a tak zabezpečia vyrovnanejšiu hru. V našom prípade sme merali schopnosť adaptácie prostredníctvom počtu remíz a predĺženia počtu krokov.

V prípade metódy kompromisu na základe počtu krokov do konca hry možno pozorovať výrazný vplyv parametru β, ktorého navyšovaním narastá aj počet remíz (a prípadných prehier), resp. klesá počet výhier natrénovaného agenta. Už samotné využitie predlžovacej stratégie pri hodnote $\beta = 1$ dokáže v prípade prvej aj druhej skupiny znížiť počet výhier pod 50% a taktiež predĺžiť počet vykonaných krokov v hre.

V prípade metódy kompromisu na základe hodnotenia akcií môžeme opäť pozorovať výrazný vplyv parametru β na výsledný počet remíz, resp. počet krokov v hre. Na druhej strane hodnota parametru N nemá (okrem pár prípadov) výrazný vplyv na

Tabuľka 6.6: Výsledky 3. skupiny s využitím metódy kompromisu na základe hodnotenia akcií

Beta	N	Výhry	Prehry	Remízy	Priemerný počet krokov výhry / prehry / všetky hry
-	-	0	0	247	- / - / 25.0
3.0	1	0	0	247	- / - / 25.0
3.0	4	0	0	247	- / - / 25.0
3.0	8	0	0	247	- / - / 25.0
3.0	25	0	0	247	- / - / 25.0
5.0	1	0	0	247	- / - / 25.0
5.0	4	0	0	247	- / - / 25.0
5.0	8	0	0	247	- / - / 25.0
5.0	25	0	0	247	- / - / 25.0
10.0	1	0	0	247	- / - / 25.0
10.0	4	0	0	247	- / - / 25.0
10.0	8	0	0	247	- / - / 25.0
10.0	25	0	0	247	- / - / 25.0

výsledky. To znamená, že počas hodnotenia akcií nezáleží, či metóda hodnotí len posledný ťah alebo viac ťahov.

V oboch metódach kompromisu hrá významnú rolu práve nami navrhnutá predlžovacia stratégia, ktorá úspešne prispieva ku navýšeniu počtu remíz, resp. počtu krokov. Z dosiahnutých výsledkov teda vyplýva, že nami navrhované metódy dokázali predĺžiť počty krokov v pozorovaných hrách a navýšiť celkové počty remíz v jednotlivých výkonnostných skupinách a zároveň nespôsobili zníženie výkonu agenta v hrách, v ktorých remizoval. Zachovanie počtu remíz podporuje náš počiatočný predpoklad, že agent prispôsobí vyberanie predlžujúcich akcií vzhľadom na úroveň súpera.

Výsledky experimentov vykonané v prostredí TicTacToe sa samozrejme môžu líšiť od experimentov vykonaných v iných, komplexnejších prostrediach s potenciálne nekonečným počtom krokov. Preto by overenie výsledkov v inom prostredí mohlo byť predmetom ďalšieho výskumu (ako aj ďalšia modifikácia predlžovacej stratégie).

Kapitola 7

Monte Carlo prehľadávacie stromy s využitím tenzorov

Ako uvádzame v predchádzajúcich kapitolách, MCTS je dôležitou súčasťou algoritmov učenia posilňovaním odvodených z AlphaGO. Síce MCTS výrazne zvyšuje dosiahnutú úroveň agentov v konkrétnych prostrediach, stále sa jedná o neefektívny algoritmus z hľadiska výpočtovej náročnosti. Rýchlosť vykonania MCTS simulácií je žiadúca najmä kvôli ich opätovnému vykonávaniu pri výbere akcie.

Za posledné dve dekády boli navrhnuté viaceré postupy, ako zrýchliť vykonávanie MCTS simulácií. Napríklad paralelizácia MCTS bežiaceho na procesore (CPU) je efektívna v zariadeniach so zdieľanou pamäťou [39] a s využitím listovej, stromovej alebo koreňovej paralelizácie [40].

Ďalšie návrhy delegujú vykonávanie operácií predovšetkým na grafickú kartu (GPU), ktorá dosahuje násobne vyššie rýchlosti v spracovaní vektorových dát. Jeden z prvých prístupov využíval plnú implementáciu MCTS na GPU [41], čo neprinieslo výraznú efektivitu v dôsledku sekvenčného spracovania atomických operácií rôzneho typu, ktoré rýchlejšie zvláda realizovať CPU. Ďalšia implementácia sa zameriavala na blokovú paralelizáciu jednotlivých častí stromu, v ktorom každý proces spracovával pridelený blok. Autori v článku následne uviedli aj vylepšenú CPU-GPU implementáciu [42].

Z dôvodu paralelizácie bola nutná aj modifikácia pôvodnej UCT metódy, v ktorej sa počas prehľadávania rozdeľuje fáza selekcie medzi viaceré procesy. Jednou z takýchto modifikácií je Balance Unobserved in UCT (BU-UCT) [43].

Hoci vyššie uvedené metódy paralelizácie dosahujú zaujímavé výsledky a zrýchľujú

vykonávanie MCTS simulácií, zameriavajú sa len na spracovanie jedného veľkého stromu. Avšak počas zberu dát v učení posilňovaním, agent využíva väčší počet malých nezávislých stromov za účelom výberu akcií v každom spustenom prostredí - pre každý pozorovaný stav sa vytvára samostatný MCTS. Napríklad počas tréningu MZ v doméne Atari hier sa v každom strome vykonávalo 50 simulácií (v prípade stolných hier sa vykonávalo 800 simulácií).

Implementácie riešiace paralelizáciu viacerých MCTS sú založené na CPU alebo kombinovaných CPU a GPU prístupoch. Wernerova open-source implementácia MZ využíva paralelizáciu procesov počas zbierania dát [34]. Každý proces má uloženú kópiu NN, pričom zbiera dáta len z jedného prostredia, t.j. vykonáva sa práve jedna MCTS metóda na proces. V paralelizácií procesov sa využíva knižnica Ray (s politikou priradenia jednej GPU práve jednému procesu). Preto v zariadeniach s menším počtom GPU a väčším počtom CPU jadier, je vhodnejšie využiť vykonávanie operácií výhradne na CPU (predovšetkým pre menšie NN). Uvedená implementácia bola použitá vo viacerých prácach [44, 45, 46, 47]. Existuje viacero dostupných GitHub repozitárov s implementáciami založenými na rovnakom, resp. podobnom princípe [48, 49, 50, 51, 52, 53].

EfficientZero je vzorkovo efektívnejšia verzia pôvodného MZ [54]. V implementácii si každý proces taktiež uchováva NN, avšak proces zbiera dáta z viacerých prostredí. V každom procese je MCTS metóda aplikovaná na pozorované stavy súčasne. Počas MCTS simulácie, fáza selekcie je vykonávaná postupne pre spracovávané stavy resp. stromy. Vybrané kombinácie akcií a stavov sú poslané ako vzorka vstupov do NN. Zo vstupných dát sa predikujú výstupy pomocou dynamickej a predikčnej funkcie. Fázy simulácie a spätného šírenia sú opäť vykonávané sekvenčne.

Odhliadnuc od pamäťovej náročnosti, za najväčší problém paralelizácie MCTS v učení posilňovaním považujeme spracovanie fáz selekcie a spätného šírenia na CPU. Tieto fázy totiž obsahujú atomické operácie spojené s prechádzaním stromu a neustálym prepočítavaním numerických hodnôt.

V prípade MCTS v kombinácii s agentom MZ sa nám nepodarilo z on-line dostupných implementácií nájsť efektívnejšie riešenie, ako:

1. Predspracovanie pozorovaných stavov pomocou prezentačnej funkcie

2. Rozdelenie výstupov z prezentačnej funkcie na menšie vzorky, ktoré budú ďalej

spracovávať paralelné procesy (jednu vzorku spracúva jeden proces)

3. Proces inicializuje strom pre každý pozorovaný stav získanej vzorky
4. Počas jednotlivých MCTS simulácií, procesy začnú vykonávať fázy selekcie postupne (sekvenčne) pre každý strom.
5. Každý proces pošle získané dáta hlavnému procesu.
6. Hlavný proces vykoná dopredný beh NN nad všetkými získanými dátami súčasne.
7. Predikované dáta rozdelí a pošle späť paralelným procesom.
8. Paralelne procesy postupne pridajú získané hodnoty do stromov a sekvenčne aplikujú fázu spätného šírenia informácií.
9. Po skončení simulácií pošle každý paralelný proces výsledné údaje hlavnému procesu.

Spomenuté implementácie dokonca používajú menej efektívne spracovanie dát (ako uvedené riešenie), resp. distribúciu úkonov medzi procesy. Dokonca MCTS implementácie algoritmov AZ a AlphaGO sú limitované aj prostredím bežiacom na CPU. V prípade EfficientZero, uchovávanie NN v každom procese je pamäťovo neefektívne a predlžuje čas predikcie, keďže GPU je vyťažená predikciami z viacerých procesov súčasne.

V rámci výskumu sa nám podarilo úspešne navrhnúť a implementovať heuristiku MCTS pomocou tenzorov, resp. multi-dimenzionálnych polí. Najväčšou výhodou je skutočnosť, že vykonávanie vektorových, resp. tenzorových operácií na grafickej karte je oproti CPU násobne rýchlejšie. V našej implementácii sa vykonávajú jednotlivé fázy MCTS simulácií všetkých stromov súčasne (za pomoci tenzorových operácií), t.j. naša metóda MCTS spracúva paralelne určitý počet stromov, odpovedajúci počtu aktuálne spustených prostredí.

7.1 Popis metódy

Implementácia je navrhnutá k algoritmu MZ z dôvodu jeho dosiahnutých state-of-the-art výsledkov a možnosti využitia NN ako náhrady prostredia. Ten avšak nezohľadňuje terminálové stavy (nevie v ktorých stavoch nastáva koniec hry), preto sme neskôr

vytvorili jednoduchú modifikáciu, aby bolo možné implementáciu používať aj v iných algoritmoch, ako napr. AZ (najlepšie v kombinácii s GPU prostredím), či v rôznych ďalších kombináciách agentov učenia posilňovaním.

Implementácia využíva metódy z knižnice PyTorch, avšak v prípade potreby je možné použiť aj ďalšie populárne knižnice, ktoré podporujú operácie s tenzormi na GPU (napr. TensorFlow).

7.1.1 Dátová štruktúra a notácie

Dostupné online implementácie využívajú stromovú štruktúru, ktorá pozostáva z objektov reprezentujúcich vrcholy stromu a smerníkov predstavujúcich hrany [34, 48, 49, 50, 51, 52, 53, 54]. Na základe predchádzajúcich skúseností konštatujeme, že v kritických častiach algoritmov je vhodné uprednostniť používanie jednoduchých dátových typov pred prácou s objektmi, ktorá si vyžaduje náročnejšiu údržbu. V našom prípade sme nahradili stromovú objektovú štruktúru tenzorovou, pričom všetky údaje sú uložené v tenzoroch na GPU.

Každý vrchol MCTS stromu si musí uchovávať hodnoty rôznych atribútov, napr. stav, q hodnoty, zoznam potomkov a pod. V našej implementácii je každý typ atribútov uchovaný v inom tenzore - tabuľke.

Počas návrhu sme sa inšpirovali Q tabuľkou (maticou), ktorá uchováva Q hodnoty všetkých kombinácií stavov a akcií. Jej riadky predstavujú jednotlivé stavy prostredia a stĺpce množinu dostupných akcií. Q tabuľka má rozmer $|S| \times |A|$, t.j. počet všetkých stavov a počet všetkých akcií.

Na základe pamäťových nárokov, nie je možné používať Q tabuľku v komplexných prostrediach s vysokým počtom stavov. Avšak v našom prípade nie je potrebné uchovávať všetky stavy prostredia v pamäti. Pri spracovaní vzorky C_T pozorovaných stavov (z C_T bežiacich prostredí) s počtom simulácií C_S, je potrebné alokovať pamäť pre $C_T + C_S \times C_T$ stavov. Prvých C_T stavov patrí koreňom stromov. Ďalších $C_S \times C_T$ stavov bude preskúmaných počas MCTS simulácií (C_T stavov počas jednej simulácie). V našom prístupe považujeme prvý riadok tabuľky za prázdny z implementačných dôvodov (vysvetlené neskôr), takže celkový počet riadkov jedného tenzoru je $C_R = C_T + C_S \times C_T + 1$.

Indexovanie v RL sa líši oproti štandardnej matematickej notácii s dolným indexom.

Indexy sú zapísané v zátvorkách za tenzorom, napr. $Q(s, a)$ - prvý index odkazuje na stav, druhý na konkrétnu akciu.

Všetky stavy sa uchovávajú v tenzore S. Prvý rozmer tenzoru je počet stavov, resp. riadkov C_R, ďalšie rozmery sú totožné s rozmerom jedného stavu. Počas inicializácie metódy sú do tenzoru S vložené koreňové stavy. Počas MCTS simulácií sa pridávajú preskúmavané stavy. Každý stav, resp. vrchol má priradené unikátne identifikačné číslo (ID) odpovedajúce číslu riadku v tenzore (index). Identifikačné čísla (indexy) sú totožné vo všetkých tenzoroch štruktúry. Korene majú indexy od 1 do C_T (vrátane). Indexy nových stavov, resp. vrcholov z prvej MCTS simulácie sú v intervale od $C_T + 1$ do $2 \times C_T$, z druhej od $2 \times C_T + 1$ do $3 \times C_T$, atď.

Q hodnoty sú uložené v tenzore Q, podobnému Q tabuľke. Prvý index (index riadku) predstavuje ID vrcholov. Rozmer tenzoru je $C_R \times |A|$ - druhý index ukazuje na akcie. Na začiatku MCTS metódy sú všetky hodnoty inicializované na nulu. Jednotlivé q hodnoty sa upravujú počas fázy spätného šírenia informácií.

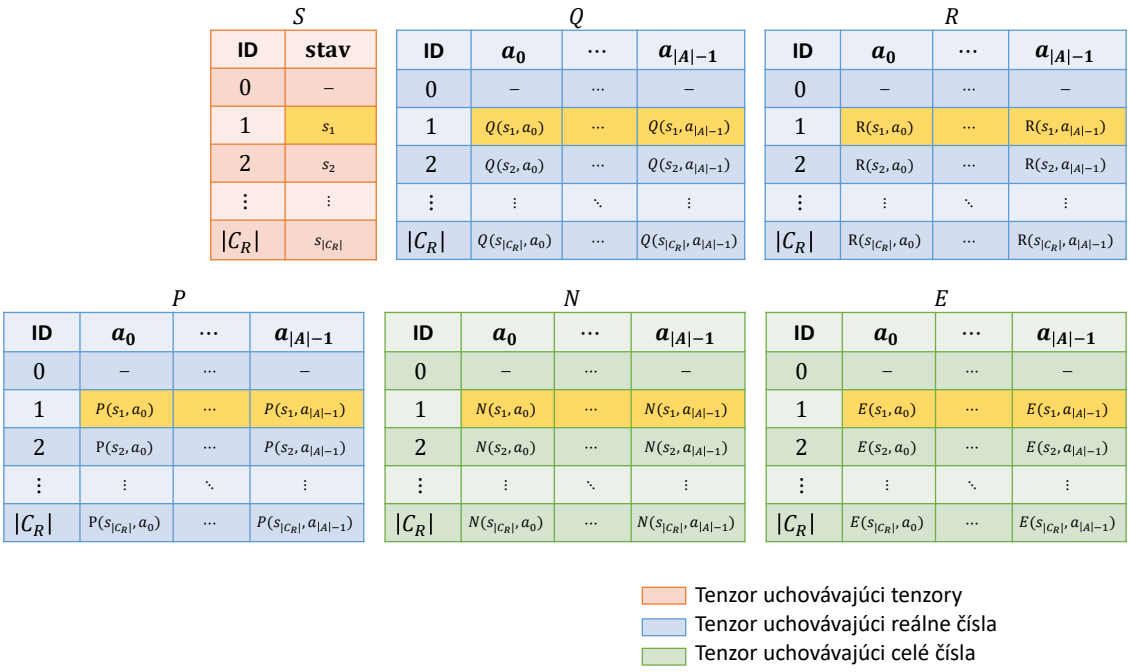
Na rovnakom princípe funguje aj tenzor R (rozmeru $C_R \times |A|$) uchovávajúci predikované odmeny pomocou dynamickej funkcie. Odmeny sú vkladané počas fázy expanzie.

Predikované rozdelenia pravdepodobností sú uložené v tenzore P (s rovnakým rozmerom $C_R \times |A|$).

Na rozdiel od doposiaľ spomenutých tenzorov uchovávajúcich desatinné hodnoty, tenzor N pozostáva z celých čísel. Ukladá počty navštívení akcií, resp. potomkov vrcholu. Keďže je dôležité pamätať si počet navštívenia každej akcie v prehľadávaných vrcholoch, rozmer tenzoru je opäť rovnaký $C_R \times |A|$.

Posledný tenzor E (s rozmerom $C_R \times |A|$) uchováva identifikačné čísla potomkov vrcholov. Ak existuje hrana z rodičovského vrcholu $ID = i$ po navštívení akcie a do potomka s $ID = j$, potom $E(i, a) = j$. Ak neexistuje hrana z vrcholu i po navštívení akcie a, potom hodnota $E(i, a) = 0$ (z tohto dôvodu je nultý riadok v tenzoroch prázdny).

Úplná dátová štruktúra pozostáva zo šiestich tenzorov (obr. 7.1). Ako sme už spomenuli, prvý index každého tenzoru predstavuje ID (riadky) vrcholov. V tenzore S, index i vráti i-ty stav (vrchol s $ID = i$), pričom ostatné tenzory vrátia i-ty vektor. Napr. tenzor Q s indexom i vracia vektor q hodnôt $Q(i) = [Q(i, 0), Q(i, 1), \cdots, Q(i, |A| - 1)]$. Na druhej strane, index i v kombinácii s akciou a ukazuje na skalárnu q hodnotu $Q(i, a)$.



Obrázok 7.1: Dátová štruktúra — údaje každého vrcholu sú uložené v šiestich tenzoroch. Index riadku predstavuje ID vrcholu. Napr. zvýraznený riadok predstavuje hodnoty vrcholu pre ID = 1.

V našej implementácii často využívame vektorové, resp. tenzorové indexovanie. Napr. kombinácia vrcholu $ID = i$ a vektoru akcií $\vec{a} = [a_0, a_{15}, a_{17}]$ (a_0, a_{15} a a_{17} predstavujú celočíselné hodnoty) môže byť použitá ako $Q(i, \vec{a}) = [Q(i, a_0), Q(i, a_{15}), Q(i, a_{17})]$. Kombinácia dvoch indexových vektorov $\vec{i} = [i_2, i_5, i_1, i_{25}]$ a $\vec{a} = [a_{10}, a_5, i_{17}, a_{10}]$ aplikovaná na tenzore Q vráti $Q(\vec{i}, \vec{a}) = [Q(i_2, a_{10}), Q(i_5, a_5), Q(i_1, a_{17}), Q(i_{25}, a_{10})]$

7.1.2 Inicializácia a predspracovanie

Pred vykonávaním MCTS simulácií je potrebné inicializovať dátovú štruktúru a ďalšie pomocné lokálne premenné (alg. 11, riadky 1-9). Vektor koreňových indexov $\vec{i}_R = [1, 2, 3, \cdots, C_T - 1, C_T]$ sa generuje ako celočíselná postupnosť z intervalu $[1, C_T]$ (alg. 11, riadok 10).

Tenzor koreňových stavov je počítaný zo vzorky pozorovaní pomocou prezentačnej funkcie h. Tenzor pravdepodobností a vektor hodnôt sú získané predikciou funkcie politiky f. Ku predikovaným pravdepodobnostiam je pripočítaný Dirichletov šum pre podporu prehľadávania stavového priestoru. Získané stavy a rozdelenia pravdepodob-

ností sa vkladajú do tenzorov dátovej štruktúry S a P pomocou vektoru koreňových indexov (alg. 11, riadky 14-15).

Algoritmus 11 Inicializácia a predspracovanie

Require: vzorka pozorovaných stavov o

Require: počet simulácií C_S

Require: počet akcií $|A|$

1: $C_T \leftarrow |o|$

2: $C_N \leftarrow C_S + 1$

3: $C_R \leftarrow 1 + C_T \times C_N$

4: $S \leftarrow$ nulový tenzor s rozmerom $C_R \times \mathit{rozmery\ stavu}$

5: $P \leftarrow$ nulový tenzor s rozmerom $C_R \times |A|$

6: $Q \leftarrow$ nulový tenzor s rozmerom $C_R \times |A|$

7: $R \leftarrow$ nulový tenzor s rozmerom $C_R \times |A|$

8: $N \leftarrow$ celočíselný nulový tenzor s rozmerom $C_R \times |A|$

9: $E \leftarrow$ celočíselný nulový tenzor s rozmerom $C_R \times |A|$

10: $\vec{i}_R \leftarrow$ postupnosť z intervalu $[1, C_T]$

11: $s \leftarrow h(o\ |\theta_h)$

12: $p, \vec{v} \leftarrow f(s\ |\theta_f)$

13: $p \leftarrow p +$ Dirichletov šum

14: $S(\vec{i}_R) \leftarrow s$

15: $P(\vec{i}_R) \leftarrow p$

7.1.3 Fáza selekcie

Počas MCTS simulácií je potrebné uchovávať v pamäti prechádzané trajektórie každého stromu, získané počas fázy selekcie (kvôli poslednej fáze - spätného šírenia). Každá trajektória (s dĺžkou L) je tvorená prechádzanými vrcholmi a akciami, t.j. $\tau = [(ID_0, a_0), (ID_1, a_1), (ID_2, a_2), \cdots, (ID_{L-1}, a_{L-1})]$. Identifikačné čísla prechádzaných vrcholov a indexy akcií sú vkladané do pomocných tenzorov I a A. V tenzore I sa nachádzajú ID vrcholov a v tenzore A indexy vybraných akcií pomocou PUCT metódy. Oba tenzory majú rozmer $C_N \times C_T$. C_N je počet simulácií zvýšený o 1 (pretože prvé uložené ID patria koreňom). Počet stĺpcov je C_T, t.j. každý strom uchováva trajektóriu

v jednom stĺpci (index stĺpcu zodpovedá poradovému indexu stromu). Vektor $\vec{l}$ (dĺžky C_T) predstavuje počty krokov prechádzaných trajektórií. Uchovávané informácie sú dôležité najmä v nasledujúcich fázach.

Ako sme uviedli, trajektórie začínajú z koreňových vrcholov, resp. každá trajektória začína koreňom konkrétneho stromu. Preto je potrebné vložiť ID koreňových vrcholov na nultý riadok tenzoru I, ešte pred hlavným cyklom selekcie (alg. 12, riadok 4). Počítadlo krokov je nastavené na hodnotu 0.

Počas fázy selekcie je veľmi dôležitý vektor $\vec{i_N}$, ktorý sa skladá z indexov aktívnych trajektórií. Inými slovami, vektor $\vec{i_N}$ si uchováva indexy stromov, ktoré pokračujú vo fáze selekcie. Na začiatku je vektor naplnený vektorom koreňových indexov, pretože každý strom sa zúčastňuje fázy selekcie.

Cyklus selekcie začína podmienkou $\vec{i_N} \neq \emptyset$. Podmienka kontroluje, či vektor aktívnych trajektórií je prázdny. Ak je vektor prázdny, nemá zmysel v cykle pokračovať a tak fáza selekcie končí.

Na začiatku cyklu sú ID naposledy pridaných vrcholov aktívnych trajektórií vložené do vektoru $\vec{i}$ pomocou aktuálneho kroku a vektoru indexov aktívnych trajektórií $\vec{i_N}$ (alg. 12, riadok 8).

Aplikovanie PUCT metódy (rovnica 4.4) pre aktuálne vrcholy je taktiež zabezpečené vykonaním tenzorových operácií sčítania, násobenia, delenia atď. Metóda vracia vektor vybraných akcií v konkrétnych vrcholoch (jedna akcia na jeden vrchol).

Indexy posledných vrcholov aktívnych trajektórií sú aktualizované (alg. 12, riadok 11).

Zo získaných akcií $\vec{a}$ metódou PUCT sa kontrolujú potomkovia vrcholov v tenzore E. Ako sme uviedli v popise tenzoru E, ak ešte kombinácia (ID, a) nebola navštívená, tenzor vráti hodnotu 0. Získané hodnoty z tenzoru E sú vložené na nový riadok tenzoru I.

Pre zjednodušenie, majme štyri stromy $C_T = 4$, avšak iba tri aktívne trajektórie $\vec{i_N} = [0, 1, 3]$, aktuálne vrcholy $\vec{i} = [5, 7, 8]$, vybrané akcie $\vec{a} = [3, 1, 0]$ a $E(\vec{i}, \vec{a}) = [E(5, 3) = 15, E(7, 1) = 0, E(8, 0) = 21] = [15, 0, 21]$. Ak $I(krok, \vec{i_N}) = [15, 0, 21]$, potom aktualizovaný nový riadok bude vyzerať ako $I(krok) = [15, 0, 0, 21]$. Prvá nula indikuje koniec aktívnej trajektórie s indexom 1 pretože neexistuje potomok z vrcholu $ID = 7$ a akcie 1. Druhá nula je nezmenená hodnota patriaca neaktívnej trajektórii.

Posledný príkaz vráti indexy nenulových hodnôt aktuálneho riadku z tenzoru I. Ak je vektor prázdny, fáza selekcie je ukončená a nasleduje ďalšia fáza MCTS simulácie.

Algoritmus 12 Fáza selekcie

1: $I \leftarrow$ celočíselný nulový tenzor s rozmerom $C_N \times C_T$
2: $A \leftarrow$ celočíselný nulový tenzor s rozmerom $C_N \times C_T$
3: $\vec{l} \leftarrow$ celočíselný nulový vektor s dĺžkou C_T
4: $I(0) \leftarrow \vec{i}_R$
5: $krok \leftarrow 0$
6: $\vec{i_N} \leftarrow \vec{i}_R$
7: **while** $\vec{i_N} \neq \emptyset$ **do**
8: $\quad \vec{i} \leftarrow I(krok, \vec{i_N})$
9: $\quad \vec{a} \leftarrow$ aplikuj PUCT metódu na vrcholy $\vec{i}$
10: $\quad A(krok, \vec{i_N}) \leftarrow \vec{a}$
11: $\quad \vec{l}(\vec{i_N}) \leftarrow krok$
12: $\quad krok \leftarrow krok + 1$
13: $\quad I(krok, \vec{i_N}) \leftarrow E(\vec{i}, \vec{a})$
14: $\quad \vec{i_N} \leftarrow$ vráť indexy nenulových hodnôt z vektoru $I(krok)$
15: **end while**

7.1.4 Fáza simulácie a expanzie

V našej implementácii, resp. v jej popise sme spojili fázy simulácie a expanzie, pretože obe obsahujú len zopár navzájom na seba nadväzujúcich riadkov kódu.

Na základe vektoru $\vec{l}$ sú získané kombinácie posledných vrcholov a akcií zo všetkých trajektórií (alg. 13, riadky 1-3). Tenzor nových stavov a vektor odmien sa počítajú za pomoci dynamickej funkcie. Z nových stavov sú predikované pravdepodobnostné rozdelenia a hodnoty stavov pomocou prediktívnej funkcie.

ID nových vrcholov, resp. nových stavov sa počítajú z vektoru koreňových indexov $\vec{i_R}$. Následne sa aktualizujú tenzory R a E.

Na konci fázy expanzie, predikované tenzory stavov s a pravdepodobnostných rozdelení p sú vkladané ako nové vrcholy do dátovej štruktúry, resp. do tenzorov S a P (alg. 13, riadky 9-10).

Algoritmus 13 Fázy simulácie a expanzie

Require: aktuálne číslo MCTS simulácie C_I

1: $\vec{k} \leftarrow$ generuj sekvenciu celých čísel v poradí, z intervalu $[0, C_T - 1]$

2: $\vec{i} \leftarrow I(\vec{l}, \vec{k})$

3: $\vec{a} \leftarrow A(\vec{l},\ \vec{k})$

4: $s, \vec{r} \leftarrow d(S(\vec{i}\ ), \vec{a}|\theta_d)$

5: $p, \vec{v} \leftarrow f(s\ |\theta_f)$

6: $\vec{i_{new}} \leftarrow \vec{i_R} + C_T \times (C_I + 1)$

7: $E(\vec{i}, \vec{a}) \leftarrow \vec{i_{new}}$

8: $R(\vec{i}, \vec{a}) \leftarrow \vec{r}$

9: $S(\vec{i_{new}}) \leftarrow s$

10: $P(\vec{i_{new}}) \leftarrow p$

7.1.5 Fáza spätného šírenia

Počas fázy spätného šírenia sú získané trajektórie prechádzané reverzne s cieľom aktualizovať q hodnoty a počty navštívení. Pomocná lokálna premenná *krok* uchováva dĺžku najdlhšej trajektórie získanej počas fázy selekcie. Preto index fázy začína od $krok - 1$.

Na začiatku cyklu sa získajú indexy aktívnych trajektórií v aktuálnom kroku. Pomocou indexov sú získané ID vrcholov a zvolené akcie z tenzorov I a A (alg. 14, riadky 3-4).

Sumy zrážaných odmien aktívnych trajektórií, získané z $\vec{v}(\vec{i_N})$, sa aktualizujú pomocou Bellmanovej rovnice. Následne sa aktualizujú aj atribúty vybraných vrcholov, resp. ich q hodnoty a počty navštívení (tenzory Q a N).

Cyklus končí upravením atribútov koreňových vrcholov.

7.1.6 Spracovanie výstupu

Po vykonaní MCTS simulácií v cykle sa počítajú výsledné hodnoty pravdepodobností vykonania akcií z jednotlivých stromov. Jedná sa o vydelenie počtov navštívení akcií z koreňa ich súčtom, v prípade MZ počtom simulácií. Hodnota stavu sa získa násobením získaných pravdepodobností akcií s ich počítanými q hodnotami.

V niektorých implementáciách MCTS sa rieši aj výber akcie. Ten môže prebiehať

Algoritmus 14 Fáza spätného šírenia

1: **for** $krok \leftarrow$ dekrementuj z $krok - 1$ do 0 **do**
2: $\quad \vec{i_N} \leftarrow$ vráť indexy nenulových hodnôt z vektoru $I(krok)$
3: $\quad \vec{i} \leftarrow I(krok, \vec{i_N})$
4: $\quad \vec{a} \leftarrow A(krok, \vec{i_N})$
5: $\quad \vec{v}(\vec{i_N}) \leftarrow R(\vec{i}, \vec{a}) + \gamma \times \vec{v}(\vec{i_N})$
6: $\quad Q(\vec{i}, \vec{a}) \leftarrow \frac{Q(\vec{i},\vec{a}) \times N(\vec{i},\vec{a}) + \vec{v}(\vec{i_N})}{N(i,\vec{a})+1}$
7: $\quad N(\vec{i}, \vec{a}) \leftarrow N(\vec{i}, \vec{a}) + 1$
8: **end for**

ako náhodný výber akcie vzhľadom na vypočítané rozdelenie pravdepodobnosti akcií, alebo výberom akcie s maximálnou pravdepodobnosťou (záleží od nutnosti situácie). Knižnica PyTorch umožňuje vykonávanie oboch metód na GPU.

7.1.7 Rozšírenie o terminálové stavy

Hoci použitie navrhnutej implementácie je prezentované na algoritme MZ (v ktorom absentujú terminálové stavy), s nasledujúcimi úpravami sa dá navrhnutá metóda aplikovať aj na modely prostredia podporujúce terminálové stavy (napr. AZ):

- Pridanie nového tenzoru T (tvoreného jednotkami) do pamäťovej štruktúry, ktorý si uchováva informáciu o terminálových kombináciách (s, a). Tento tenzor je nutné aktualizovať pri dosiahnutí nových, resp. nepreskúmaných stavov.
- Inicializácia jednotkového vektora $\vec{t}$ s dĺžkou C_T v alg. (12) za pôvodným riadkom 3. Vektor uchováva informáciu, či konkrétna trajektória nedosiahla terminálový stav. Ak bol na konci i-tej trajektórie obsiahnutý terminálový stav, tak $\vec{t}(i) = 0$.
- V alg. (12) za pôvodným riadkom 11, je potrebné aktualizovať informácie o terminálových stavoch nasledovne $\vec{t}(\vec{i_N}) \leftarrow T(\vec{i}, \vec{a})$.
- V alg. (12) je potrebné upraviť riadok 13 nasledovne $I(krok, \vec{i_N}) \leftarrow E(\vec{i}, \vec{a}) \times T(\vec{i}, \vec{a})$
- V alg. (13) je potrebné spracovať len tie trajektórie, ktorých index vo vektore $\vec{t}$ je nenulový. V tomto prípade je dôležité najskôr inicializovať nulový vektor $\vec{v}$ a následne do neho vložiť predikované hodnoty stavov.

7.2 Experimenty

V experimentoch sme pozorovali čas vykonania MCTS metód na vzorkách dát rôznych veľkostí. Nami navrhnutá implementácia bola porovnávaná s tromi rozdielnymi prístupmi. Každá implementácia mala k dispozícii grafickú kartu NVIDIA GeForce RTX 2080 Ti a 16 jadrový procesor Intel(R) Xeon(R) CPU E5-2643 0 @ 3.30GHz. Zoznam testovaných implementácií:

- CPU - použili sme Wernerovú open-source implementáciu z GitHub repozitáru [34]. Implementácia vytvára nový proces pre každé bežiace prostredie. Každý proces okrem prostredia obsahuje aj kópiu NN. Implementácia využíva knižnicu Ray, ktorá spravuje bežiace procesy. Implementácia bola použitá a citovaná vo viacerých výskumoch [44, 45, 46, 47] a na jej princípe existuje rada ďalších dostupných implementácií [48, 49, 50, 51, 52, 53]. V našich experimentoch sme testovali jej využitie len v testoch bez NN, pretože uchovanie vysokého počtu kópií NN na GPU je pamäťovo náročné.

- CPU-GPU S - v implementácii sú vykonávané fázy selekcie, expanzie a spätného šírenia informácií sekvenčne. MCTS metóda sekvenčne vykoná fázy selekcie pre stromy všetkých spustených prostredí v cykle. Zozbierané dáta spolu putujú do NN (na GPU). Získané údaje sú sekvenčne spracované metódami expanzie a spätného šírenia.

- CPU-GPU P - na začiatku metódy MCTS, implementácia rozdelí skupiny stromov do viacerých procesov, v ktorých sa fázy selekcie, expanzie a spätného šírenia vykonávajú paralelne. Po fáze selekcie sa pošlú potrebné dáta hlavnému procesu, ktorý vykoná predikciu pomocou NN. Predikované dáta sa pošlú späť jednotlivým procesom.

- GPU - navrhnutá, vyššie popísaná GPU tenzorová implementácia.

Implementácie CPU-GPU S, CPU-GPU P a GPU sme implementovali a vyladili. Implementáciu CPU sme čiastočne modifikovali s cieľom umožniť vykonávanie replikácií.

Experimenty porovnávajú implementácie v doméne Atari hier. V doméne sa nachádza veľké množstvo hier s odlišným cieľom hry, funkciou odmeny a počtom akcií. Avšak naším cieľom nie je natrénovať MZ agenta na konkrétnej hre, ale porovnanie časov

vykonania MCTS implementácií. Preto sme experimenty zovšeobecnili a neorientovali sa len na jednu hru.

Spracovávaná vzorka pozorovaných stavov je generovaná ako náhodný tenzor obsahujúci C_T menších tenzorov odzrkadľujúcich pozorovaný stav. Každý pozorovaný stav má rozmer $128 \times 96 \times 96$, ten je rovnaký, ako vstup v pôvodných MZ experimentoch [25]. V pôvodných experimentoch pozorovaný stav pozostával z posledných 32 RGB obrázkov hry a 32 vykonaných akcií. Počet akcií v každom experimente je 18, čo je maximálny možný počet akcií u Atari hier.

V prípade CPU-GPU P sme vykonali experimenty s využitím 2, 5, 10 a 25 procesov, na rozdiel od CPU implementácie, kde sa nám osvedčilo použitie 5, 10 a 15 procesov (nižšie a vyššie počty procesov predlžujú čas vykonávania).

Veľkosti vzoriek C_T pozorovaných stavov, ktoré sme použili v experimentoch, sú 50, 100, 250, 500 a 750 (vyšší počet pozorovaných stavov v kombinácií s NN presiahol pamäť použitej GPU a tak by bolo nutné využiť GPU s vyššou pamäťou, prípadne viac grafických kariet).

Z hľadiska využitia NN sme vykonávali dva druhy experimentov - s využitím NN a bez využitia NN. Experimenty bez využitia NN odzrkadľujú čas potrebný na vykonanie MCTS metódy bez času potrebného na predikciu prostredníctvom NN.

V poslednej sade experimentov sme sa orientovali na meranie trvania najrýchlejších implementácií s rôznym počtom MCTS simulácií C_S na vzorke $C_T = 100$. Počty jednotlivých MCTS simulácií sme si zvolili 50, 100, 200 a 400. Takúto sadu experimentov sme opäť vykonávali bez využitia NN.

Výsledok každého experimentu je priemerná hodnota počítaného času v sekundách zo 100 vykonaných replikácií spolu s polovičnou hodnotou vypočítaného intervalu spoľahlivosti.

7.2.1 Scenáre

Sústredili sme sa na 2 scenáre, ktoré odzrkadľujú krajné prípady rozhodovania:

1. náhodná akcia - v scenári pozorujeme rozhodovanie doposiaľ nenatrénovanej NN. Výber akcií počas fáze selekcie predstavuje v dlhodobom horizonte práve rovnomerné rozdelenie. V tomto prípade inklinuje MCTS k rastu do šírky.

2. konštantná akcia - v scenári pozorujeme možné rozhodovanie pretrénovanej siete. Počas fáze selekcie sa stále preferuje výber práve jednej (konštantnej) akcie. V scenári inklinuje MCTS k rastu do hĺbky, čím môžeme preskúmať maximálne dĺžky trajektórií.

V prípade experimentov s využitím NN, predikované rozdelenie pravdepodobnosti je prepísané funkciou generujúcou rozdelenie aktuálneho scenáru.

Okrem generovaného rozdelenia pravdepodobnosti, predikcia hodnoty stavu a odmeny sa vždy prepíše na hodnotu 0, aby sa predišlo odklonu od vybraného scenára.

7.2.2 Architektúra neurónovej siete

Ako sme uviedli vyššie, NN sa skladá z troch modulov, resp. funkcií - prezentačná, dynamická a funkcia politiky. Keďže MCTS metódu sme prispôsobili MZ, rozhodli sme sa použiť modifikovanú architektúru MZ z originálneho článku. Všetky filtre majú rozmer 3×3, aktivačnú funkciu používame relu.

Prezentačná funkcia je identickou kópiou pôvodnej MZ funkcie. Pozostáva z:

- 1 konvolučná vrstva s odsadením 2, počtom kernelov 128, výstup má rozmer $128 \times 48 \times 48$
- 2 reziduálne bloky s počtom kernelov 128
- 1 konvolučná vrstva s odsadením 2, počtom kernelov 256, výstup má rozmer $256 \times 24 \times 24$
- 3 reziduálne bloky s počtom kernelov 256
- podvzorkovanie s využitím priemeru, s odsadením 2, výstup má rozmer $256 \times 12 \times 12$
- 3 reziduálne bloky s počtom kernelov 256
- podvzorkovanie s využitím priemeru, s odsadením 2, výstup má rozmer $256 \times 6 \times 6$

Výstup prezentačnej funkcie, resp. rozmer každého stavu, je $256 \times 6 \times 6$.

Dynamická funkcia dostáva na vstupe kombináciu stavu a akcie, ktorá je zakódovaná v tenzore $1 \times 6 \times 6$ a pripojená ku tenzoru stavu (vstupný tenzor má rozmer $257 \times 6 \times 6$). Dynamická funkcia poskytuje dva výstupy - nový stav a odmenu.

V pôvodnom článku neboli spomenuté informácie o plne prepojených skrytých vrstvách, preto sme museli použiť vlastný návrh. Ten vyplýva z predchádzajúcich experimentov počas implementácie PPO algoritmu.

Dynamická funkcia sa skladá z:

- 1 konvolučná vrstva s počtom kernelov 256
- 8 reziduálnych blokov s počtom kernelov 256 (výstup z posledného bloku je novým stavom)
- transformácia tenzoru na vektor s dĺžkou 9216
- plne prepojená vrstva s počtom neurónov 512
- relu
- plne prepojená vrstva s počtom neurónov 1 (predikovaná odmena)

Funkcia politiky taktiež poskytuje dva výstupy - rozdelenie pravdepodobnosti a hodnotu stavu. Pripája sa na výstup prezentačnej alebo dynamickej vrstvy. Rovnako aj architektúra funkcie politiky sa nenachádza v pôvodnom článku. Náš návrh bol opäť ovplyvnený predchádzajúcimi skúsenosťami.

Spoločná časť funkcie politiky predstavuje jednu plne prepojenú vrstvu s počtom neurónov 512 a aktivačnou funkciou relu. Tu sa výstupné hlavy rozdeľujú. Každá hlava obsahuje jednu plne prepojenú vrstvu s počtom neurónov 512 a opäť aktivačnou funkciou relu. Výstupná plne prepojená vrstva má 18 neurónov v hlave predikujúcej pravdepodobnostné rozdelenie a 1 neurón v hlave predikujúcej hodnotu stavu.

7.2.3 Výsledky bez využitia NN ako modelu prostredia

Výsledky z tabuliek (7.1) a (7.2) boli získané s využitím $C_S = 50$ MCTS simulácií. V oboch tabuľkách možno pozorovať efektivitu nami navrhnutej implementácie (GPU) voči ostatným.

V prípade scenáru náhodnej akcie (tabuľka 7.1), čas spracovania $C_T = 750$ pozorovaných stavov je približne 6.5 násobne nižší ako čas spracovania 50 pozorovaných stavov druhou najefektívnejšou metódou v danom riadku (CPU GPU P 2). V prípade

Tabuľka 7.1: Výsledky porovnania MCTS implementácií s aplikáciou scenáru náhodnej akcie bez využitia NN

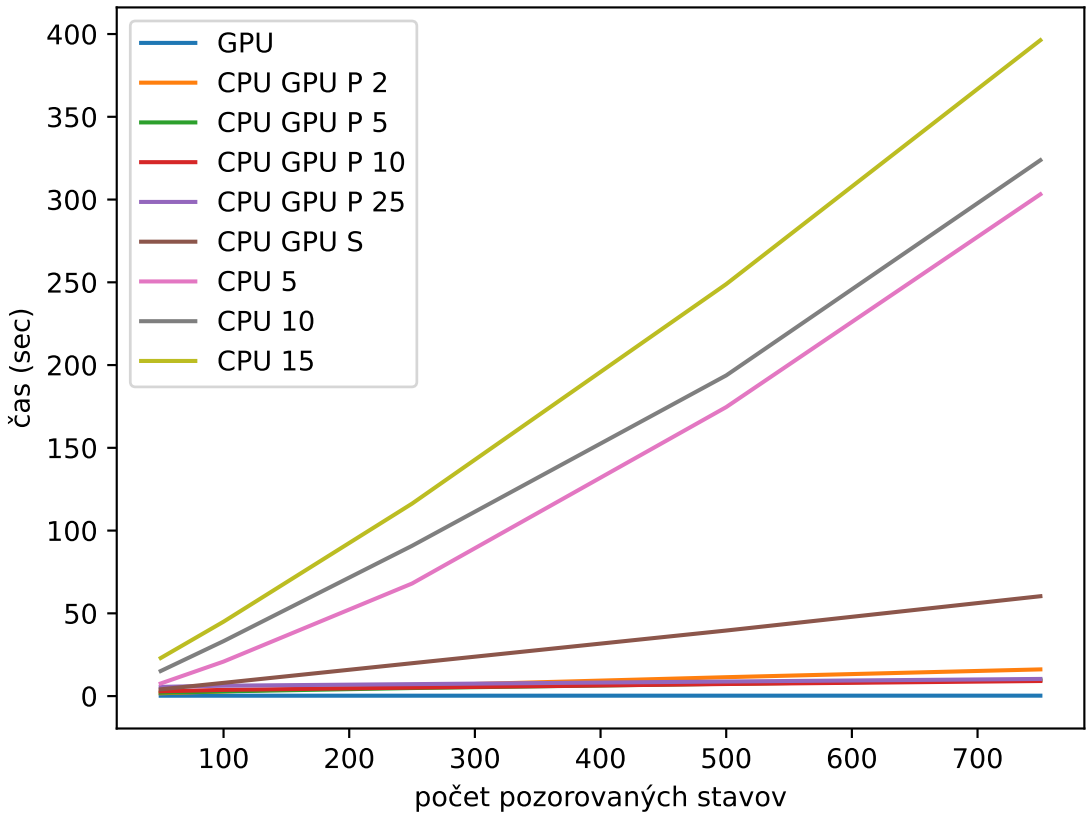
C_T	GPU	CPU GPU P				CPU GPU S	CPU		
		2	5	10	25		5	10	15
50	0.225 ±0.001	1.721 ±0.013	2.045 ±0.01	2.891 ±0.017	5.576 ±0.038	4.09 ±0.003	7.62 ±0.112	15.147 ±0.223	22.999 ±0.183
100	0.234 ±0.001	2.855 ±0.009	2.832 ±0.011	3.749 ±0.016	6.282 ±0.032	8.001 ±0.005	20.849 ±0.036	33.206 ±0.139	44.956 ±0.635
250	0.243 ±0.001	6.27 ±0.016	4.869 ±0.022	5.103 ±0.014	7.224 ±0.035	19.865 ±0.008	68.015 ±0.158	90.801 ±0.427	116.284 ±0.755
500	0.253 ±0.001	11.421 ±0.022	7.447 ±0.029	7.299 ±0.026	8.811 ±0.031	39.61 ±0.015	174.611 ±0.361	193.741 ±0.239	248.905 ±0.738
750	0.262 ±0.001	16.131 ±0.045	9.791 ±0.041	9.118 ±0.022	10.368 ±0.038	60.344 ±0.015	303.172 ±0.536	323.785 ±1.458	396.275 ±0.724

porovnania rýchlostí na rovnakom počte pozorovaných stavov, tak GPU implementácia oproti druhej najrýchlejšej implementácii je:

- 7.64 násobne rýchlejšia (oproti CPU GPU 2) pre $C_T = 50$
- 12.10 násobne rýchlejšia (oproti CPU GPU 5) pre $C_T = 100$
- 20.03 násobne rýchlejšia (oproti CPU GPU 5) pre $C_T = 250$
- 28.84 násobne rýchlejšia (oproti CPU GPU 10) pre $C_T = 500$
- 34.80 násobne rýchlejšia (oproti CPU GPU 10) pre $C_T = 750$

Z vyššie uvedených výsledkov vyplýva, že so zvyšovaním počtu pozorovaných stavov sa násobne zvyšuje efektivita nami navrhnutej implementácie oproti ostatným. Časový nárast použitých prístupov je zobrazený na obrázku (7.2). Samotný čas vykonania GPU implementácie pre $C_T = 750$ je len 16% vyšší ako nameraný čas pre $C_T = 50$.

V scenári s konštantnou akciou (tabuľka 7.2) je možné pozorovať výrazný nárast času takmer u všetkých implementácií oproti výsledkom z prvého scenára. Nárast je spôsobený opakovaným zvyšovaním dĺžky prechádzaných trajektórií v MCTS, získaných



Obrázok 7.2: časová náročnosť jednotlivých MCTS implementácií v závislosti od počtu pozorovaných stavov s aplikáciou scenáru náhodnej akcie bez využitia NN

počas fáze selekcie. Zvyšovanie dĺžky každej trajektórie je zapríčinené výberom konštantnej akcie (vždy sa prechádza rovnaká trajektória).

Hoci je nami navrhnutá implementácia stále najrýchlejšia a opäť platí, že čas spracovania 750 pozorovaných stavov je násobne nižší ako čas spracovania 50 pozorovaných stavov druhou najefektívnejšou metódou, tak násobky rýchlostí sú v jednotlivých riadkoch nižšie. Teda v prípade scenáru konštantnej akcie, GPU implementácia je:

- 3.02 násobne rýchlejšia (oproti CPU GPU 10) pre $C_T = 50$
- 4.24 násobne rýchlejšia (oproti CPU GPU 10) pre $C_T = 100$
- 7.49 násobne rýchlejšia (oproti CPU GPU 25) pre $C_T = 250$
- 11.82 násobne rýchlejšia (oproti CPU GPU 25) pre $C_T = 500$
- 15.95 násobne rýchlejšia (oproti CPU GPU 25) pre $C_T = 750$

Tabuľka 7.2: Výsledky porovnania MCTS implementácií s aplikáciou scenáru konštantnej akcie bez využitia NN

C_T	GPU	CPU GPU P				CPU GPU S	CPU		
		2	5	10	25		5	10	15
50	1.686 ±0.008	7.519 ±0.012	5.194 ±0.028	5.095 ±0.019	7.096 ±0.028	35.175 ±0.064	14.941 ±0.119	14.49 ±0.143	17.266 ±0.127
100	1.704 ±0.006	13.575 ±0.012	7.825 ±0.03	7.24 ±0.018	8.812 ±0.033	69.282 ±0.116	28.769 ±0.044	28.142 ±0.035	47.295 ±0.12
250	1.732 ±0.005	31.876 ±0.027	15.257 ±0.031	13.037 ±0.028	12.975 ±0.033	170.135 ±0.163	72.939 ±0.067	72.173 ±0.152	147.788 ±0.395
500	1.766 ±0.004	63.233 ±0.041	28.529 ±0.033	22.916 ±0.044	20.886 ±0.037	340.716 ±0.474	144.271 ±0.205	140.942 ±0.088	374.888 ±0.753
750	1.787 ±0.003	94.143 ±0.087	41.317 ±0.042	31.937 ±0.031	28.512 ±0.046	500.181 ±0.479	218.951 ±0.395	207.96 ±0.137	685.497 ±1.611

Z druhého scenáru a vyššie uvedených násobkov rýchlosti opäť vyplýva, že navyšovaním počtu pozorovaných stavov sa násobne zvyšuje efektivita GPU implementácie voči ostatným. Samotný čas vykonania GPU implementácie pre $C_T = 750$ je len o 5% vyšší ako nameraný čas pre $C_T = 50$.

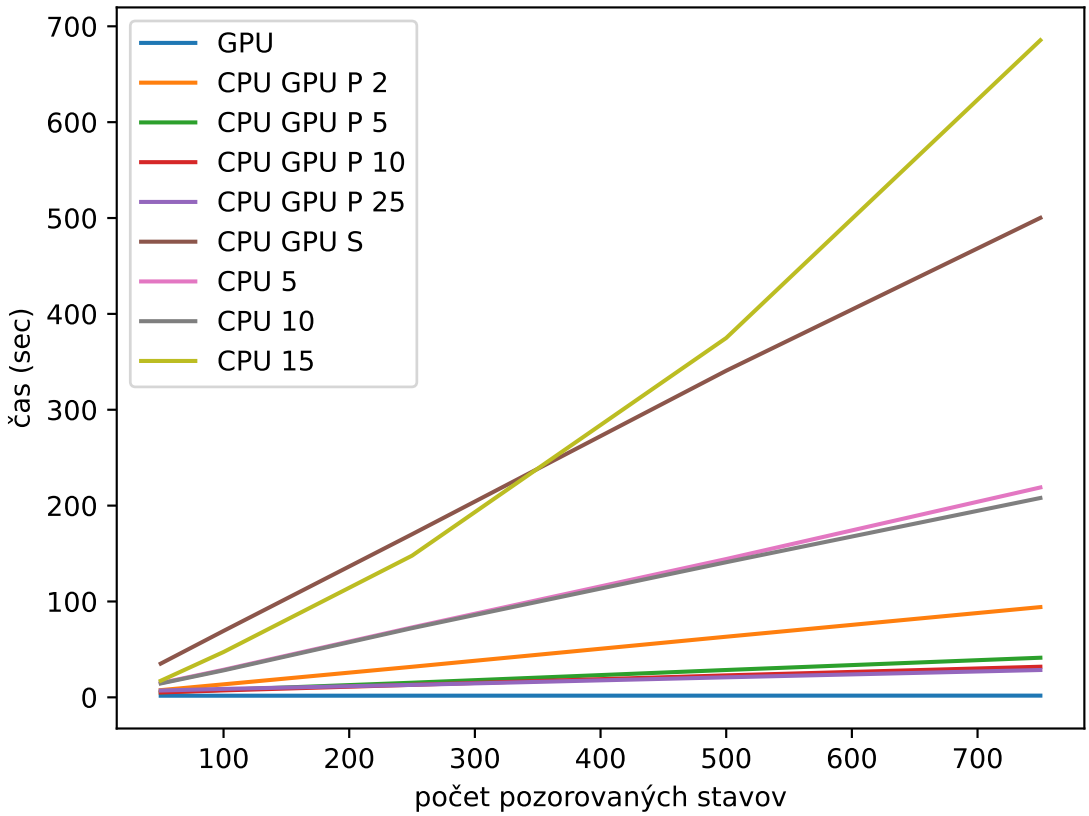
Časový nárast v závislosti od počtu pozorovaných stavov možno pozorovať na obrázku (7.3).

Zaujímavým pozorovaním (nezávisle od hlavného cieľa), je zrýchlenie Wernerovej (CPU) implementácie pri spracovaní vysokého počtu pozorovaných stavov v scenári konštantnej akcie oproti scenáru náhodnej akcie (napr. v prípade CPU 10 pre $C_T = 750$ je zrýchlenie 1.55 násobné).

7.2.4 Výsledky s využitím NN

Vo výsledkoch s využitím NN s počtom MCTS simulácií $C_S = 50$ (tabuľky 7.3 a 7.4) možno pozorovať časový nárast vykonania MCTS metódy, spôsobený behom NN.

Výsledky Wernerovej implementácie s využitím NN na CPU sú príliš výpočtovo náročné (pre $C_T = 50$ je priemerný čas vykonania 338 sekúnd), preto sme upustili od



Obrázok 7.3: časová náročnosť jednotlivých MCTS implementácií v závislosti od počtu pozorovaných stavov s aplikáciou scenáru konštantnej akcie bez využitia NN

ich testovania.

Keďže čas predikcie na vzorke pozorovaných stavov je navýšený o čas vykonania NN, tak GPU implementácia pochopiteľne opäť dosahuje prvenstvo.

Výsledné časy vykonania s využitím NN predstavujú významný časový nárast oproti výsledkom bez využitia NN, napr. v prípade scenáru náhodnej akcie pre $C_T = 750$ sa celkový čas navýšil o 2.204 sekundy.

7.2.5 Výsledky s rozdielnym počtom MCTS simulácií

Z výsledkov pre oba scenáre (tabuľky 7.5 a 7.6) vyplýva, že nami navrhnutá GPU implementácia dosahuje z hľadiska časovej náročnosti najlepšie výsledky.

Avšak v prípade nárastu meraného času medzi $C_S = 50$ a $C_S = 400$ dosahuje nami navrhnutá metóda najvyššie násobky, napr. v scenári konštantnej akcie sa jedná o

Tabuľka 7.3: Výsledky porovnania MCTS implementácií s aplikáciou scenáru náhodnej akcie s využitím NN

C_T	GPU	CPU GPU P				CPU GPU S
		2	5	10	25	
50	0.383 ±0.001	1.962 ±0.007	2.269 ±0.008	3.285 ±0.016	5.634 ±0.047	6.091 ±0.004
100	0.539 ±0.001	3.053 ±0.006	3.091 ±0.008	3.919 ±0.015	6.442 ±0.037	10.001 ±0.005
250	0.946 ±0.001	6.909 ±0.012	5.574 ±0.014	5.757 ±0.021	7.868 ±0.026	21.864 ±0.009
500	1.714 ±0.002	12.607 ±0.019	9.144 ±0.044	8.787 ±0.023	10.254 ±0.03	41.61 ±0.018
750	2.466 ±0.003	18.185 ±0.024	11.857 ±0.055	11.47 ±0.039	12.571 ±0.031	62.345 ±0.016

55.921 násobný nárast (druhý najvyšší nárast dosiahnutý GPU CPU P 5 je 36.448 násobný). Je teda pravdepodobné, že v prípade vysokého počtu MCTS simulácií môže nami navrhovaná metóda zaostávať.

Zaujímavým pozorovaním je aj zvýšenie efektivity, resp. zníženie meraného času v prípade CPU GPU P 25 implementácie, ktorá bola najpomalšia v scenári s náhodnou akciou, avšak v prípade scenára konštantnej akcie dokázala časovo prekonať rovnakú implementáciu s nižším počtom procesov. Domnievame sa, že tento jav je spôsobený takým časovo náročným vykonávaním MCTS simulácií, ktoré sa oplatí paralelizovať väčším počtom procesov aj za cenu vyšších časových nárokov pre ich spravovanie.

7.3 Záver kapitoly

V kapitole sme popísali navrhnutú implementáciu MCTS metódy implementovanej pomocou tenzorov pre algoritmus MZ. Implementácia je vhodná v prípadoch, v ktorých je potrebné vykonávanie rozhodnutí prostredníctvom MCTS vo viacerých prostrediach paralelne, napr. v oblasti učenia posilňovaním.

Experimenty ukázali časovú efektivitu nami navrhnutej implementácie na GPU

Tabuľka 7.4: Výsledky porovnania MCTS implementácií s aplikáciou scenáru konštantnej akcie s využitím NN

C_T	GPU	CPU GPU P				CPU GPU S
		2	5	10	25	
50	1.909 ±0.008	7.942 ±0.011	5.332 ±0.023	5.296 ±0.019	7.309 ±0.028	33.371 ±0.05
100	2.074 ±0.007	14.182 ±0.022	8.426 ±0.028	7.849 ±0.023	9.201 ±0.026	67.45 ±0.105
250	2.507 ±0.005	32.975 ±0.028	16.207 ±0.028	14.054 ±0.035	14.231 ±0.033	168.301 ±0.142
500	3.232 ±0.004	65.08 ±0.053	29.884 ±0.035	24.283 ±0.035	22.527 ±0.048	338.874 ±0.423
750	3.984 ±0.004	97.403 ±0.092	43.776 ±0.041	34.896 ±0.04	30.752 ±0.046	501.425 ±0.351

oproti iným používaným prístupom. Pre porovnanie sme uviedli aj experimenty s využitím NN používanej v doméne Atari hier.

Radi by sme opätovne spomenuli výsledky experimentov bez využitia NN, kde navrhnutá GPU implementácia dosahuje najnižší časový nárast medzi $C_T = 50$ a $C_T = 750$ a to síce len 1.16 násobok v prípade scenára s náhodou akciou, resp. 1.06 násobok v prípade scenára s konštantnou akciou.

V experimentoch s využitím NN sa výrazne navýšili časy vykonania GPU implementácie v prípade oboch scenárov. Avšak časový nárast spôsobený vykonaním NN ovplyvňuje všetky testované implementácie, keďže v každej implementácii sa predikcia pomocou NN vykonáva rovnako.

Vzhľadom na výsledky experimentov s rozdielnym počtom MCTS simulácií konštatujeme, že navrhnutá GPU implementácia dosahuje najvyššie rozdiely medzi $C_S = 50$ a $C_S = 400$. To znamená, že časová efektivita GPU implementácie klesá s navyšovaním C_S.

Z dosiahnutých výsledkov konštatujeme, že je vhodné využívať čo najvyšší počet spustených prostredí za účelom dosiahnutia najlepšej efektivity. Samozrejme ten závisí od viacerých faktorov ako napr. pamäťová náročnosť na CPU (dáta z prostredia uložené

Tabuľka 7.5: Výsledky porovnania rôznych počtov MCTS simulácií s aplikáciou scenáru náhodnej akcie bez využita NN

C_S	GPU	CPU GPU P		
		5	10	25
50	0.234 ±0.001	2.832 ±0.011	3.749 ±0.016	6.282 ±0.032
100	0.481 ±0.006	4.906 ±0.015	5.454 ±0.019	9.945 ±0.032
200	1.068 ±0.001	9.908 ±0.028	10.225 ±0.023	18.109 ±0.034
400	2.294 ±0.002	20.422 ±0.134	19.844 ±0.031	34.562 ±0.082

v RAM) a GPU (NN spolu s dátami z MCTS), schopnosti paralelizácie vykonávania jednotlivých prostredí a pod.

Tabuľka 7.6: Výsledky porovnania rôznych počtov MCTS simulácií s aplikáciou scenáru konštantnej akcie bez využitia NN

C_S	GPU	CPU GPU P		
		5	10	25
50	1.704 ±0.006	7.825 ±0.03	7.24 ±0.018	8.812 ±0.033
100	6.371 ±0.029	22.36 ±0.072	19.344 ±0.03	19.734 ±0.037
200	24.281 ±0.135	75.867 ±0.095	61.968 ±0.066	60.144 ±0.099
400	95.291 ±0.571	285.212 ±0.216	217.657 ±0.139	193.77 ±0.1

Kapitola 8

MuZero s podporou prieskumu stavového priestoru

Modelovo založené algoritmy využívajú model prostredia buď pre podporu rozhodovania [23, 25, 27], alebo pre podporu prieskumu stavového priestoru [31, 32]. V poslednej kapitole sa zaoberáme návrhom modelovo založeného agenta, ktorý kombinuje oba prístupy.

V časti analýzy venovanej modelovo založeným algoritmom (Kapitola 4) sme sa okrem už vyššie popísaných algoritmov zaoberali aj predikciou pozorovaného stavu pomocou NN. Hoci v tejto oblasti dominuje agent SimPle [27], na predikciu pozorovaného stavu v doméne Atari hier existuje rada ďalších výskumov [28, 29, 30]. V týchto výskumoch bola NN trénovaná pomocou učenia učiteľom s datasetom uchovávajúcim pätice ako v zozname skúseností, resp. $(o_t, a_t, r_t, o_{t+1}, T_t)$.

Ak disponujeme v doméne Atari hier plne pozorovaným stavom, tak eliminujeme možnosť stochastických prechodov, čím v ľubovoľnej kombinácii pozorovaného stavu a akcie dostaneme práve jeden nasledujúci pozorovaný stav. V takomto prípade by chyba predikcie učiacej sa NN (modelu prostredia predikujúceho nový pozorovaný stav spolu s odmenou a terminálnou informáciou) predstavovala vnútornú (internú) odmenu, resp. motiváciu agenta prehľadávať stavový priestor. Rovnaká NN by tak mohla plniť dynamickú funkciu agenta MZ, čím by sme zabezpečili tvorbu MCTS.

Na základe vykonaných experimentov sme zistili, že v jednoduchých hrách (Pong, Breakout) sa takýto dynamický model dokáže naučiť takmer presne predikovať odmenu a terminálny stav. Problém nastáva pri predikcii pozorovaného stavu. Väčšina oblastí

pozorovaného stavu je predikovaná správne, avšak správnosť predikcie malých, resp. pohyblivých komponentov sa znižuje (napr. loptička). Počas algoritmu MCTS sa tak chyba predikcie navyšuje, nakoľko stavy nových vrcholov sú predikované z predchádzajúcich predikovaných stavov. Kvôli tomuto problému sme boli nútení začať s výskumom opäť od začiatku.

V nasledujúcej časti výskumu sme sa zaoberali vylepšením algoritmu MZ v podobe pridania schopnosti vnútornej motivácie do algoritmu. Schopnosť vnútornej motivácie sme riešili pomocou už odskúšaného algoritmu RND [32].

8.1 Popis riešenia

Ako sme už uviedli v popise algoritmu RND, algoritmus obsahuje dve NN - učiacu a cieľovú. Učiaca sieť sa snaží priblížiť k predikciám výstupov cieľovej siete. Obe siete vykonávajú predikcie nad rovnakým (aktuálne) pozorovaným stavom. Avšak v MCTS sa pozorované stavy objavujú len na začiatku algoritmu, konkrétne pred ich spracovaním prezentačnou funkciou (ďalej MCTS obsahuje len vnútorné stavy). Preto sme RND nepoužívali počas budovania MCTS, ale až pri trénovaní algoritmu zo zoznamu skúseností (vysvetlené neskôr).

Proces učenia nami modifikovaného algoritmu prebieha ako cyklus iterácií. Každá iterácia pozostáva z vykonania kroku v každom prostredí a následného tréningu MZ a RND sietí práve z jednej vzorky získanej zo zoznamu skúseností.

8.1.1 Modifikácia MCTS

Potrebu zahrnúť internú odmenu do rozhodovacieho procesu (do MCTS) sme riešili rozšírením prediktívnych funkcií, atribútov vrcholu a modifikáciou výpočtu q hodnôt využívaných v PUCT rovnici.

Výstup dynamickej funkcie sme rozšírili o internú odmenu $\vec{r}_i(s_t, a_t)$ (opäť predikovanú ako pravdepodobnostné rozdelenie). V takomto prípade sa nejedná o predikciu konštantnej hodnoty, nakoľko pri prieskume stavového priestoru má interná odmena, resp. chyba predikcie potenciál klesať. Výstupy z dynamickej funkcie môžeme zapísať ako $s_{t+1}, \vec{r}_e(s_t, a_t), \vec{r}_i(s_t, a_t) = g(s_t, a_t|\theta_g)$.

Výstup funkcie politiky sme rozšírili o predikciu internej hodnoty stavu, resp.

hodnoty stavu pozostávajúcej len z interných odmien $V_i^\pi(s,\cdot)$ (opäť sa jedná o vektor pravdepodobnostného rozdelenia). Rozšírený výstup funkcie možno zapísať ako $P^\pi(s,\cdot), V_e^\pi(s,\cdot), V_i^\pi(s,\cdot) = f(s|\theta_f)$.

Nakoľko samostatne predikujeme hodnoty interných a externých odmien a taktiež interných a externých hodnôt stavov, tak potrebujeme rozdeliť aj ich uchovanie vo vrcholoch MCTS. Preto každý vrchol bude uchovávať vektor externých odmien $R_e(s,\cdot)$ a vektor interných odmien $R_i(s,\cdot)$ samostatne. Taktiež sme rozdelili aj vektor q hodnôt $Q^\pi(s,\cdot)$ na externé $Q_e^\pi(s,\cdot)$ a interné $Q_i^\pi(s,\cdot)$ hodnoty, čím môžeme využívať rozdielne hodnoty γ koeficientov.

$$\vec{q} = \frac{\beta_e Q_e^\pi(s,\cdot) + \beta_i Q_i^\pi(s,\cdot)}{\beta_e + \beta_i} \tag{8.1}$$

Pri aplikácii PUCT rovnice počítame q hodnoty ako kombináciu interných a externých hodnôt (rovnica 8.1). Samozrejme výsledný vektor q hodnôt je dôležité normalizovať do intervalu $[0,1]$.

8.1.2 Modifikácia učenia neurónových sietí

Agent MZ využíva na trénovanie svojich prediktívnych funkcií zoznam skúseností (off-policy tréning). Na druhej strane algoritmus RND bol trénovaný na aktuálne zozbieraných vzorkách dát (on-policy tréning). Boyao Li [55] zlúčil off-policy RL agenta s on-policy ICM algoritmom prostredníctvom jednoduchej zmeny tréningu ICM na off-policy a následným sčítaním externých a interných odmien pri počítaní q hodnôt.

Hoci v implementácii trénujeme NN na vzorke dát (trajektórií) zo zoznamu skúseností, pre zjednodušenie budeme popisovať algoritmus učiaceho procesu (alg. 15) len na jednej vybranej trajektórii. Dáta zo zoznamu skúseností budeme značiť horným indexom X^D.

Prediktívne funkcie agenta MZ (s výnimkou RND sietí) predikujú vektory veličín. Avšak niektoré časti si vyžadujú transformáciu predikovaného vektoru na skalárnu veličinu (napr. počítanie cieľových hodnôt stavov), ktorú budeme značiť ako $phi(x)$. Inverzná transformácia je rozloženie skalárnej veličiny na vektor (transformácia cieľovej hodnoty na rozdelenie), ktorú budeme značiť ako $phi^{-1}(x)$. Transformácie predstavujú operácie používané v MZ článku [25].

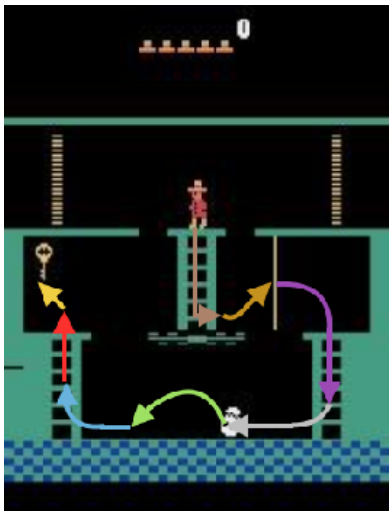
V algoritme sú všetky externé cieľové hodnoty ako aj pravdepodobnostné rozdelenia uložené v zozname skúseností a priebežne sa aktualizujú po zozbieraní nových dát, resp.

pred procesom učenia.

8.2 Experimenty

Experimenty sme vykonávali v dvoch prostrediach z domény Atari hier - Pong a Montezuma's Revenge. Pong považujeme za najjednoduchšie prostredie zo širokej škály Atari hier. Využívali sme ho prevažne na testovanie a ladenie modifikovaného algoritmu.

Prostredie Montezuma's Revenge je známe pre svoju komplexnosť a riedke odmeny. Hra predstavuje akýsi labyrint pozostávajúci z 24 rozličných miestností. Hráč sa snaží prechádzať jednotlivé miestnosti a hľadať v nich predmety akými sú napr. meč, fakľa alebo kľúče, ktoré mu otvoria dvere do ďalších miestností. V miestnostiach sú ukryté rôzne nástrahy, ktoré hráčovi zoberú život a vrátia ho na počiatočnú pozíciu. Len samotné získanie prvého kľúča, resp. prvých bodov si vyžaduje, aby hráč vykonal sekvenciu deviatich úkonov s rozličnou dobou trvania (obrázok 8.1), pričom čelí nástrahám ako pád, kolízia s nepriateľom a vstup do ohňa.



Obrázok 8.1: Cesta potrebná pre získanie prvých bodov v hre Montezuma's Revenge

Montezuma's Revenge je jedno z najnáročnejších prostredí v doméne Atari hier. RL agenti bez podpory prieskumu stavového priestoru tu spravidla nezískajú žiadne body. Rovnaký počet bodov (nula) nahral aj pôvodný MZ po 100 miliónoch krokov.

Z dôvodu komplexnosti druhého prostredia, ako aj časovej a výpočtovej náročnosti, sme sa rozhodli vykonať experimenty s nižším počtom krokov, v ktorých sme cielili len na získanie prvých bodov (tréning jednoduchšieho agenta - PPO v kombinácii s

Algoritmus 15 Proces učenia

Require: Zoznam skúseností D

Require: Začiatok trajektórie t

Require: Maximálna dĺžka trajektórie k

1: Vypočítaj koniec trajektórie $K \leftarrow t + k$

2: Získaj vnútorný stav $s_t \leftarrow h(o_t|\theta_h)$

3: $P^\pi(s_t, \cdot), V_e^\pi(s_t, \cdot), V_i^\pi(s_t, \cdot) \leftarrow f(s_t|\theta_f)$

4: Počítaj chybu politiky $L_P \leftarrow \pi^D(s_t)\ logP^\pi(s_t, \cdot)$

5: Počítaj chybu extern. hodnoty stavu $L_{VE} \leftarrow phi^{-1}(V_E^D(s_t))\ log\ \vec{V}_e^\pi(s_t)$

6: Inicializuj chybu intern. hodnoty stavu $L_{VI} \leftarrow 0$

7: Inicializuj chybu externej odmeny $L_{RE} \leftarrow 0$

8: Inicializuj chybu internej odmeny $L_{RI} \leftarrow 0$

9: Inicializuj chybu učiacej siete $L_{RND} \leftarrow 0$

10: **while** $t + 1 < K$ a zároveň o_{t+1} nie je terminálový stav **do**

11: $s_{t+1}, \vec{r}_e(s_t, a_t), \vec{r}_i(s_t, a_t) \leftarrow g(s_t, a_t^D|\theta_g)$

12: Aktualizuj chybu $L_{RE} \leftarrow L_{RE} + \frac{phi^{-1}(R_E^D(s_t,a_t))\ log\ \vec{r}_e(s_t,a_t)}{k}$

13: $t \leftarrow t + 1$

14: Predikuj vektor učiacou NN $\vec{u}(o_t) \leftarrow x(o_t|\theta_x)$

15: Predikuj vektor cieľovou NN $\vec{v}(o_t) \leftarrow y(o_t|\theta_y)$

16: Aktualizuj chybu učiacej NN $L_{RND} \leftarrow L_{RND} + \frac{MSE(\vec{u}(o_t),\vec{v}(o_t))}{k}$

17: Počítaj internú odmenu $R_I(s_{t-1}, a_{t-1}) \leftarrow \frac{MSE(\vec{u}(o_t),\vec{v}(o_t))}{2}$

18: Aktualizuj chybu $L_{RI} \leftarrow L_{RI} + \frac{phi^{-1}(R_I(s_{t-1},a_{t-1}))\ log\ \vec{r}_i(s_{t-1},a_{t-1})}{k}$

19: $P^\pi(s_t, \cdot), V_e^\pi(s_t, \cdot), V_i^\pi(s_t, \cdot) \leftarrow f(s_t|\theta_f)$

20: Počítaj cieľovú hodnotu intern. stavu $V_I(s_{t-1}) \leftarrow R_I(s_{t-1}, a_{t-1})\ \gamma_i\ phi(V_i^\pi(s_t, \cdot))$

21: Aktualizuj chybu $L_{VI} \leftarrow L_{VI} + \frac{phi^{-1}(V_I(s_{t-1}))\ log\ V_i^\pi(s_{t-1},\cdot)}{k}$

22: Aktualizuj chybu $L_P \leftarrow L_P + \frac{\pi^D(s_t)\ logP^\pi(s_t,\cdot)}{k}$

23: Aktualizuj chybu $L_{VE} \leftarrow L_{VE} + \frac{phi^{-1}(V_E^D(s_t))\ log\ V_e^\pi(s_t,\cdot)}{k}$

24: **end while**

25: Aplikuj optimalizačný mechanizmus na základe dosiahnutých chýb L_P, L_{VE}, L_{VI}, L_{RE}, L_{RI} a L_{RND}

RND nám trval 7 dní). Maximálny počet interakcií s prostredím sme preto nastavili na 1.2 milióna. Tento počet je dostačujúci na úspešné natrénovanie modifikovaného MZ algoritmu v hre Pong a taktiež na získanie prvých bodov v hre Montezuma's Revenge.

Experimenty sme opäť vykonávali na grafickej karte NVIDIA GeForce RTX 2080 Ti a 16 jadrovom procesore Intel(R) Xeon(R) CPU E5-2643 0 @ 3.30GHz

8.2.1 Achitektúra neurónových sietí

Pri návrhu architektúry NN sme sa inšpirovali použitými sieťami v článku MZ a vlastnými skúsenosťami z iných RL agentov. V druhom prostredí sme testovali dve alternatívy, ktoré sa líšili v počte kernelov použitých v reziduálnych blokov - 96 a 128. Opäť ako pri predchádzajúcich experimentoch, veľkosť kernelu vo všetkých vrstvách je 3. Po každej konvolučnej a plne prepojenej vrstve nasleduje aktivačná funkcia relu.

Prezentačná funkcia sa skladá z:

- 1 konvolučná vrstva s odsadením 1, veľkosťou kroku 2 a počtom kernelov 64, výstup má rozmer 48×48
- 1 konvolučná vrstva s odsadením 1, veľkosťou kroku 2 a počtom kernelov 96, výstup má rozmer 24×24
- 2 reziduálne bloky s počtom kernelov 96 (alebo 128)
- podvzorkovanie s využitím maxima, s odsadením 2, výstup má rozmer 12×12
- 2 reziduálne bloky s počtom kernelov 96 (alebo 128)
- podvzorkovanie s využitím priemeru, s odsadením 2, výstup má rozmer 6×6

Výstup z prezentačnej funkcie, resp. rozmer skrytého stavu je $96 \times 6 \times 6$, resp. $128 \times 6 \times 6$. Pôvodný MZ využíval skrytý stav s rozmerom $256 \times 12 \times 12$.

Dynamická vrstva pozostáva z piatich reziduálnych blokov s počtom kernelov 96, resp. 128. V prípade výstupných vrstiev dynamickej funkcie a funkcie politiky sa medzi každou výstupnou vrstvou nachádza plne prepojená vrstva s počtom neurónov 512.

Architektúru NN modulu RND sme navrhli na základe predchádzajúcich experimentov s agentom PPO.

Cieľová NN modulu RND pozostáva z:

- 1 konvolučná vrstva s veľkosťou kernelu 8×8 odsadením 2, veľkosťou kroku 4, počtom kernelov 32, výstup má rozmer $32 \times 24 \times 24$
- Aktivačná funkcia ELU
- 1 konvolučná vrstva s veľkosťou kernelu 3×3 odsadením 1, veľkosťou kroku 2, počtom kernelov 64, výstup má rozmer $64 \times 12 \times 12$
- Aktivačná funkcia ELU
- 1 konvolučná vrstva s veľkosťou kernelu 3×3 odsadením 1, veľkosťou kroku 1, počtom kernelov 64, výstup má rozmer $64 \times 12 \times 12$
- Aktivačná funkcia ELU
- Plne prepojená vrstva s 512 neurónmi

Učiaca sa NN modulu RND má rovnakú architektúru ako cieľová NN, avšak je ešte rozšírená o dve plne prepojené vrstvy s 512 neurónmi.

8.2.2 Parametre experimentov

Pri vykonaní experimentov v doméne Atari hier sme použili rovnaké parametre pre obe prostredia, resp. agenta:

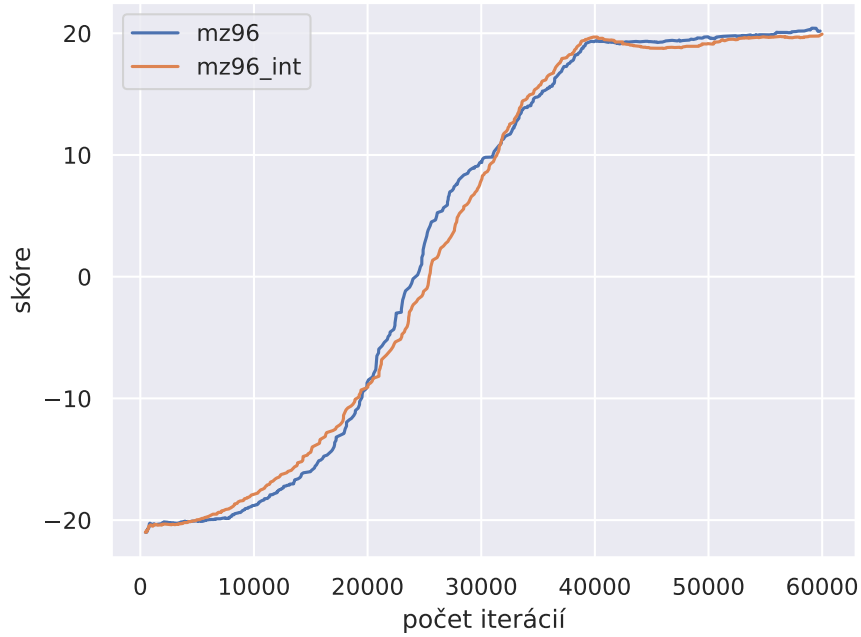
- Počet interakcií interakcií s prostredím 1200000
- Koeficient rýchlosti učenia MZ 0.0005
- Koeficient rýchlosti učenia RND 0.00025
- Počet výstupov NN predikujúcich externú hodnotu stavu 7 (interval $[-3, 3]$)
- Počet výstupov NN predikujúcich internú hodnotu stavu 7 (interval $[-3, 3]$)
- Počet výstupov NN predikujúcich externú odmenu 3 (interval $[-1, 1]$)
- Počet výstupov NN predikujúcich internú odmenu 3 (interval $[-1, 1]$)
- Počet krokov pri počítaní cieľových hodnôt $TD(\lambda) = 10$ krokov
- Počet MCTS simulácií 50

- Počet prostredí 20
- Veľkosť zoznamu skúseností 1000000 záznamov
- Veľkosť vzorky v prostredí Pong 1024, resp. 500 v prostredí Montezuma's Revenge

8.2.3 Výsledky v prostredí Pong

Prostredie Pong sa vyznačuje hustými odmenami s jednoduchým ovládaním, takže je pre väčšinu RL algoritmov vhodným nástrojom pre overenie funkčnosti samotného algoritmu. V našom prípade sme porovnávali modifikovaný MZ oproti našej implementácii pôvodného MZ vo verzii 96 kernelov.

Priemerné skóre je merané ako pohyblivý priemer meraný z posledných 100 epizód (hier). Nami implementovaný pôvodný MZ agent dosiahol najvyššie priemerné skóre 20.43 bodov (obr. 8.3, značenie mz96), pričom modifikovaný MZ dosiahol skóre 19.93 (obr. 8.3, značenie mz96_int). Nižšie skóre je pravdepodobne zapríčinené vnútornou motiváciou, ktorá už naučeného agenta stále motivuje v prieskume stavového priestoru.

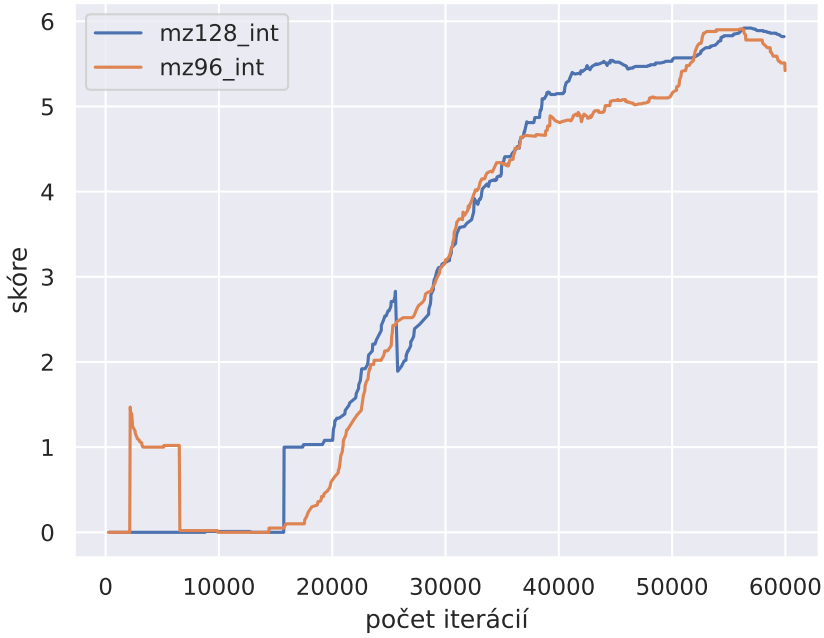


Obrázok 8.2: Dosiahnuté skóre v prostredí Pong

8.2.4 Výsledky v prostredí Montezuma's Revenge

Ako sme uviedli v popise prostredia, Montezuma's Revenge predstavuje komplexné prostredie s riedkymi odmenami. Naším cieľom bolo získať prvú odmenu, nakoľko pôvodnému agentovi MZ sa to nepodarilo, čím získal 0 bodov. Za získanie prvej odmeny získava agent 100 bodov. Takže priemerné skóre 1 predstavuje získanie odmeny jedenkrát za 100 epizód.

Modifikovaný MZ s obomi architektúrami NN (96 a 128 kernelov) dosiahol priemerné skóre takmer 6 bodov, pričom skóre pokračovalo v raste aj na konci experimentov. Výrazné rozdiely v rámci architektúr sme pri získaní prvej odmeny nezaznamenali.



Obrázok 8.3: Dosiahnuté skóre v prostredí Montezuma's Revenge

8.3 Záver kapitoly

V rámci kapitoly sa nám podarilo navrhnúť modifikovaný algoritmus MZ využívajúci vnútornú motiváciu modelu RND, ktorej informácie dokáže nepriamo využiť (predikciou interných odmien a hodnôt stavov) pri budovaní MCTS, resp. v procese rozhodovania.

Experimenty v prostredí Pong ukázali, že modifikovaný algoritmus je schopný učenia sa v prostredí s hustými odmenami, pričom vnútorná motivácia výrazne neznižuje jeho

efektivitu oproti pôvodnému MZ.

Dosiahnutie pozitívneho priemerného skóre v prostredí Montezuma's Revenge odzrkadľuje účinnosť pridanej vnútornej motivácie, resp. modifikovaného agenta oproti pôvodnému MZ. Konštatujeme, že vnútorná motivácia je dostatočne silná, aby podporila agenta v prieskume stavového priestoru.

Sme si vedomí absencie ďalších a komplexnejších experimentov na väčšom počte Atari hier, ktoré by mohli byť predmetom ďalšieho výskumu.

Záver

Výskum, resp. výstupy dizertačnej práce, môžeme rozdeliť do štyroch oblastí a to analýza súčasného stavu, adaptívnosť modelovo založeného algoritmu, návrh a implementácia MCTS s využitím tenzorov a modifikácia MZ s podporou prieskumu stavového priestoru. Všetky oblasti majú svoj spoločný prienik v hlbokom učení a algoritmoch učenia posilňovaním.

V rámci analýzy súčasného stavu sme implementovali väčšinu spomenutých agentov s cieľom porozumieť dôležitým súvislostiam počas jednoduchých experimentov. Niektoré prvotné výsledky základných experimentov a overení sme publikovali v univerzitných časopisoch, resp. na konferenciách [56, 57].

Vo výskume venovanému adaptívnosti modelovo založeného algoritmu sa nám podarilo navrhnúť, implementovať a úspešne otestovať adaptívne stratégie, ktoré sa snažia obmedziť, resp. prispôsobiť výkonnosť natrénovaného agenta slabšiemu protivníkovi. Návrh sme založili na predpoklade, že v určitých prostrediach ako napr. TicTacToe a šach, môžeme adaptívnosť agenta modelovať ako snahu hrať dlhšiu hru, resp. remizovať. Adaptívne stratégie môžu byť základom nových výskumov venovaných tejto problematike. Výsledky sme publikovali na medzinárodnej konferencii [58].

Návrh a implementáciu MCTS pomocou tenzorov považujeme za dôležitý prínos v oblasti RL výskumu, pretože dokáže výrazne urýchliť vykonávanie experimentov. Ako je možné pozorovať v experimentoch, nami navrhnutá implementácia dokázala výrazne prekonať doposiaľ používané implementácie. Výsledky experimentov spolu so zdrojovým kódom sme zverejnili v časopise Applied Sciences [59]. Zdrojový kód je taktiež dostupný na https://github.com/marrekb/MuZero.

V poslednej časti dizertačnej práce sme sa venovali modifikácii MZ agenta s podporou prieskumu stavového priestoru. Prvotné experimenty ukázali, že agent je schopný sa natrénovať v prostredí s hustými odmenami (čím sme zistili, že interné odmeny mu

neprekážajú v tréningu). V komplexnom prostredí s riedkymi odmenami (Montezuma's Revenge) sa modifikovanému agentovi podarilo získať prvú odmenu, čím nahral vyššie priemerné skóre (takmer 6 bodov) oproti pôvodnému MZ (s priemerným skóre 0 bodov). Náš prístup môže opäť pomôcť ďalším výskumom zameraným na modifikáciu skupiny AlphaGO algoritmov ako aj na rozširovanie MCTS algoritmu o podporu prieskumu.

Veríme, že navrhnuté prístupy, dostupné implementácie, výsledky experimentov ako aj ďalšie nadobudnuté poznatky v dizertačnej práci a publikáciách prispejú výskumom v populárnej oblasti učenia posilňovaním s využitím hlbokých neurónových sietí.

Zoznam použitej literatúry

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[2] Maddison C. et al. Silver D., Huang A. Mastering the game of go with deep neural networks and tree search. *Nature 529 (2016)*, pages 484–489, 2016.

[3] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[4] Babuschkin I. Czarnecki W.M. et al Vinyals, O. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature 575 (2019)*, page 350–354, 2019.

[5] Enda Howley Seyed Sajad Mousavi, Michael Schukat. Traffic light control using deep policy-gradient and value-function-based reinforcement learning. *IET Intelligent Transport Systems, Volume 11*, pages 417–423, 2017.

[6] P. Stone N. Kohl. Policy gradient reinforcement learning for fast quadrupedal locomotion. *IEEE International Conference on Robotics and Automation, 2004*, pages 1–6, 2004.

[7] Timothy Lillicrap Sergey Levine Shixiang Gu, Ethan Holly. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. *ArXiv.org*, pages 1–9, 2016.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[9] Jimmy Ba Diederik P. Kingma. Adam: A method for stochastic optimization. *ArXiv.org*, pages 1–15, 2014.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[13] Xiao Wang, Xiaohua Xie, and Jianhuang Lai. Convolutional lstm based video object detection. In Jian-Huang Lai, Cheng-Lin Liu, Xilin Chen, Jie Zhou, Tieniu Tan, Nanning Zheng, and Hongbin Zha, editors, *Pattern Recognition and Computer Vision*, pages 99–109, Cham, 2018. Springer International Publishing.

[14] Rahul Dey and Fathi M Salemt. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pages 1597–1600. IEEE, 2017.

[15] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[16] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *arXiv preprint arXiv:1509.06461*, 2015.

[17] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003, 2016.

[18] Emma Brunskill and et. al. Cs234: Reinforcement learning winter 2019, 2019.

[19] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous

methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[20] Sergey Levine Michael Jordan Pieter Abbeel John Schulman, Philipp Moritz. High-dimensional continuous control using generalized advantage estimation. 2015.

[21] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[22] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[23] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[24] Gary Linscott and et. al. Leela chess zero. *Available: https://lczero.org/*, 2018.

[25] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[26] Théophane Weber, Sébastien Racanière, David P Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomenech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. Imagination-augmented agents for deep reinforcement learning. *arXiv preprint arXiv:1707.06203*, 2017.

[27] Piotr Milos et. al. Lukasz Kaiser, Mohammad Babaeizadeh. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

[28] Honglak Lee Richard Lewis Satinder Singh Junhyuk Oh, Xiaoxiao Guo. Action-conditional video prediction using deep networks in atari games. *arXiv preprint arXiv:1507.08750*, 2015.

[29] Katja Hofmann Felix Leibfried, Nate Kushman. A deep learning approach for joint video frame and reward prediction in atari games. *arXiv preprint arXiv:1611.07078*, 2016.

[30] Elias Wang, Atli Kosson, and Tong Mu. Deep action conditional neural network for frame prediction in atari games. Technical report, Technical Report, Stanford University, 2017.

[31] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 16–17, 2017.

[32] Amos Storkey Oleg Klimov Yuri Burda, Harrison Edwards. Exploration by random network distillation. 2018.

[33] Joel Lehman Kenneth O. Stanley Jeff Clune Adrien Ecoffet, Joost Huizinga. Go-explore: a new approach for hard-exploration problems. 2019.

[34] Aurèle Hainaut Werner Duvaud. Muzero general: Open reimplementation of muzero. `https://github.com/werner-duvaud/muzero-general`, 2019.

[35] Michiel Van Der Ree and Marco Wiering. Reinforcement learning in the game of othello: Learning against a fixed opponent and learning from self-play. In *2013 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL)*, pages 108–115. IEEE, 2013.

[36] Luís Filipe Teófilo, Nuno Passos, Luís Paulo Reis, and Henrique Lopes Cardoso. Adapting strategies to opponent models in incomplete information games: a reinforcement learning approach for poker. In *International Conference on Autonomous and Intelligent Systems*, pages 220–227. Springer, 2012.

[37] He He, Jordan Boyd-Graber, Kevin Kwok, and Hal Daumé III. Opponent modeling in deep reinforcement learning. In *International conference on machine learning*, pages 1804–1813. PMLR, 2016.

[38] Gustavo Andrade, Geber Ramalho, Hugo Santana, and Vincent Corruble. Extending reinforcement learning to provide dynamic game balancing. In *Proceedings of*

the Workshop on Reasoning, Representation, and Learning in Computer Games, 19th International Joint Conference on Artificial Intelligence (IJCAI), pages 7–12, 2005.

[39] S Ali and Aske et al. Plaat. Parallel monte carlo tree search from multi-core to many-core processors. 2015.

[40] et al. Chaslo. Parallel monte-carlo tree search. 2008.

[41] J. Zhou. Parallel go on cuda with monte carlo tree search. In *Master's thesis, University of Cincinnati, OhioLINK Electronic Theses and Dissertations Center*, 2013.

[42] Kamil Rocki and Reiji Suda. Large-scale parallel monte carlo tree search on gpu. 2011.

[43] Anji Liu and et al. Liang, Yitao. On effective parallelization of monte carlo tree search. 2020.

[44] Jon Gabirondo-López, Jon Egaña, Jose Miguel-Alonso, and Raul Orduna Urrutia. Towards autonomous defense of sdn networks using muzero based intelligent agents. *IEEE Access*, 9:107184–107199, 2021.

[45] Emre Yilmaz, Olatunde Sanni, Mark Kotwicz Herniczek, and Brian German. Deep reinforcement learning approach to air traffic optimization using the muzero algorithm. In *AIAA AVIATION 2021 FORUM*, page 2377, 2021.

[46] Maxim Lapan. *Deep Reinforcement Learning. Das umfassende Praxis-Handbuch: Moderne Algorithmen für Chatbots, Robotik, diskrete Optimierung und Web-Automatisierung inkl. Multiagenten-Methoden.* MITP-Verlags GmbH & Co. KG, 2020.

[47] Julien Scholz, Cornelius Weber, Muhammad Burhan Hafez, and Stefan Wermter. Improving model-based reinforcement learning with internal state representations through self-supervision. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021.

[48] Anurag Koul. muzero-pytorch. https://github.com/koulanurag/muzero-pytorch, 2020.

[49] Ken Voskuil. muzero. https://github.com/kaesve/muzero, 2021.

[50] Johan Gras. Muzero. https://github.com/johan-gras/MuZero, 2019.

[51] Yamuna Krishnamurthy. simple-muzero. https://github.com/yamsgithub/simple-muzero, 2020.

[52] Madhu Sivaraj. muzero. https://github.com/madhusivaraj/muzero, 2020.

[53] Fidel Schaposnik. muzero. https://github.com/fidel-schaposnik/muzero, 2021.

[54] Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering atari games with limited data. *Advances in Neural Information Processing Systems*, 34, 2021.

[55] Boyao Li, Tao Lu, Jiayi Li, Ning Lu, Yinghao Cai, and Shuo Wang. Curiosity-driven exploration for off-policy reinforcement learning methods. In *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1109–1114. IEEE, 2019.

[56] Tarábek Peter Baláž Marek. Double dqn architectures performance in pong. *Journal of Information, Control and Management Systems*, pages 3–11, 2019.

[57] Tarábek Peter Baláž Marek. Effect of local rewards in games pong and lunar lander. *Mathematics in science and technologies conference*, pages 19–24, 2020.

[58] Marek Baláž and Peter Tarábek. Alphazero with real-time opponent skill adaptation. In *2021 International Conference on Information and Digital Technologies (IDT)*, pages 194–199. IEEE, 2021.

[59] Marek Baláž and Peter Tarábek. Tensor implementation of monte-carlo tree search for model-based reinforcement learning. *Applied Sciences*, 13(3):1406, 2023.

Zoznam publikácií

Double DQN architectures performance in Pong - Journal of Information, Control and Management Systems [print] = JICMS. - ISSN 1336-1716. - Roč. 17, č. 1 (2019), s. 3-11 [print].

Baláž Marek (50%), Tarábek Peter (50%)

Effect of local rewards in games Pong and Lunar Lander - Mathematics in science and technologies [print] : proceedings of the MIST conference 2020. - 1. vyd. - [S.l.]: [s.n.], 2020. - ISBN 9798648566026. - s. 19-24 [print].

Baláž Marek (50%), Tarábek Peter (50%)

AlphaZero with real-time opponent skill adaptation - Mathematics Information and digital technologies 2021 [electronic] : proceedings of the international conference. - 1. vyd. - Danvers: Institute of Electrical and Electronics Engineers, 2021. - ISBN 978-1-6654-3692-2. - s. 194-199.

Zaradené v: SCOPUS

Baláž Marek (50%), Tarábek Peter (50%)

Tensor implementation of Monte-Carlo tree search for model-based reinforcement learning - Applied sciences [electronic]. - ISSN 2076-3417 (online). - Roč. 13, č. 3 (2023), s. [1-20] [online].

Zaradené v: Current Content Connect ; SCOPUS ; Web of Science Core Collection

Spôsob prístupu: https://www.mdpi.com/2076-3417/13/3/1406

Baláž Marek (50%), Tarábek Peter (50%)