ŽILINSKÁ UNIVERZITA V ŽILINE

AUTOREFERÁT
DIZERTAČNEJ PRÁCE

Žilina, máj 2023

Ing. Marek Baláž

Žilinská univerzita v Žiline
Fakulta riadenia a informatiky

Ing. Marek Baláž

Autoreferát dizertačnej práce

UČENIE S POSILŇOVANÍM S VYUŽITÍM HLBOKÝCH NEURÓNOVÝCH SIETÍ

na získanie akademického titulu "**philosophiae doctor**" (**PhD.**)
v študijnom programe doktorandského štúdia
aplikovaná informatika
v študijnom odbore
informatika

Žilina, máj 2023

Dizertačná práca bola vypracovaná v dennej forme doktorandského štúdia na Katedre matematických metód a operačnej analýzy, Fakulte riadenia a informatiky Žilinskej univerzity v Žiline.

Predkladateľ:	*Ing.* ***Marek Baláž***
	Katedra matematických metód a operačnej analýzy
	Fakulta riadenia a informatiky
	Žilinská univerzita v Žiline
Školiteľ:	*doc. Mgr.* ***Ondrej Šuch****, PhD.*
	Matematický ústav Slovenskej akadémie vied
	Banská Bystrica
Školiteľ špecialista:	*Ing.* ***Peter Tarábek****, PhD.*
	Katedra matematických metód a operačnej analýzy
	Fakulta riadenia a informatiky
	Žilinská univerzita v Žiline
Oponenti:	*prof. Ing.* ***Igor Farkaš****, Dr.*
	Fakulta matematiky, fyziky a informatiky
	Univerzita Komenského v Bratislave
	doc. Ing. ***Michal Gregor****, PhD.*
	Ústav konkurencieschopnosti a inovácií
	Žilinská univerzita v Žiline

Autoreferát bol rozoslaný dňa: **30. 6. 2023**

Obhajoba dizertačnej práce sa koná dňa **23. 8. 2023** o **9:30** hod. pred komisiou pre obhajobu dizertačnej práce schválenou odborovou komisiou v študijnom odbore **informatika**, v študijnom programe **aplikovaná informatika**, vymenovanou dekanom Fakulty riadenia a informatiky Žilinskej univerzity v Žiline dňa **29. 6. 2023**.

prof. Ing. ***Karol Matiaško****, PhD.*
predseda pracovnej skupiny odborovej komisie
v študijnom odbore **informatika**
v študijnom programe **aplikovaná informatika**

Fakulta riadenia a informatiky
Žilinská univerzita
Univerzitná 8215/1
010 26 Žilina

Anotácia

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

Počet strán:	*107*	*Počet použitej literatúry:*	*59*
Počet obrázkov:	*12*	*Počet tabuliek:*	*12*

Annotation

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).

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

Number of pages:	*107*	*Number of references:*	*59*
Number of figures:	*12*	*Number of tables:*	*12*

1 Úvod

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 predikcii 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 [17], prehra svetového šampióna v hre GO [24], alebo nedávne úspechy v komplexných hrách akými sú Dota 2 [6] a Starcraft 2 [27].

2 Teoretické východiská a súčasný stav

2.1 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. [26]

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.

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. [26]

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.

$$\pi^* = \arg\max_{\pi} \sum_{t=0}^{T-1} \gamma^t \cdot r_t \tag{1}$$

Cieľom agenta je nájsť takú politiku π^*, ktorá maximalizuje sumu budúcich zrá-

žaných odmien (rovnica 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 $G_t = \sum_{i=t}^{T-1} \gamma^{i-t} \cdot r_i$.

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 ($V(s) = (G_t | s_t = s)$). 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". [26]

2.2 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.

2.2.1 AlphaZero

Agent AlphaZero (AZ) [23] je ďalšou generáciou pôvodného algoritmu AlphaGo [24], 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.

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, 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. 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.

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

2.2.2 MuZero

V prípade absencie kópie simulátora, je AZ praktický nefunkčný. Problém rieši agent MuZero (MZ) [22] 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 2), ktorá kladie dôraz na dôkladnejšie prehľadávanie. Využíva dve konštanty, ktorých hodnoty v pôvodných experimentoch boli stanovené $c_1 = 1.25$ a $c_2 = 19652$ [22].

$$a_t = \arg\max_a \left[Q(s_t,a) + P(s_t,a) \cdot \frac{\sqrt{\sum_b N(s_t,b)}}{1+N(s_t,a)} \cdot \left(c_1 + \log\left(\frac{\sum_b N(s_t,b) + c_2 + 1}{c_2} \right) \right) \right] \tag{2}$$

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 [22].

3 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.

Cieľom Nášho výskumu je z níženie výkonnosti naučeného agenta 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.

3.1 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, 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 4)

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 [14], 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 3), kde M predstavuje predikovanú hodnotu $M^{\pi}(s,\cdot|\theta)$ inkrementovanú o 1 v každom rodičovskom vrchole v trajektórii od listu ku koreňu.

$$L(s_t,a_t) = \frac{N(s_t,a_t) \times L(s_t,a_t) + M}{N(s_t,a_t)+1} \tag{3}$$

Pre potreby výskumu sme zadefinovali predlžovaciu stratégiu rovnicou (4). 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 [14]. Parameter $\beta > 1$ zvyšuje dôležitosť vykonávania akcií predlžujúcich hru voči q hodnotám.

$$C(s_t, a) = Q(s_t, a) + \beta \times L_{norm}(s_t, a) \tag{4}$$

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.

3.1.1 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 (5). Ak je maximálny počet krokov hry neznámy, resp. môže byť nekonečný, odporúčame opäť použitie počtu krokov, ktorý je vyšší ako priemerná dĺžka hry.

$$\alpha = \frac{poet\ krokov\ od\ zaiatku\ hry}{maximlny\ poet\ krokov\ hry} \tag{5}$$

Výber akcie vo fáze selekcie sa počíta pomocou rovnice (6). 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.

$$a_t = \arg\max_a \left[\alpha \times U(s_t, a) + (1 - \alpha) \times C(s_t, a)\right] \tag{6}$$

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.

3.1.2 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 (7). 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 $\frac{A_N}{A_N+O_N} > \frac{O_N}{A_N+O_N}$.

$$a_t = \arg\max_a \left[\frac{O_N}{A_N + O_N} \times U(s_t, a) + \frac{A_N}{A_N + O_N} \times C(s_t, a) \right] \tag{7}$$

3.2 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.

3.3 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.

3.4 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 dokázala výrazne navýšiť počet výhier v oboch skupinách. V príapde prvej skupiny, agent dokázal remizovať od 341 po 521-krát z celkového počtu 550 hier (pre β od 1.0 po 10.0), pričom celkový počet krokov sa navýšil z 7.8 až na 24.3. V prípade druhej skupiny, agent dokázal remizovať až 106-krát z celkového počtu 123 hier (pre $\beta = 10.0$) a navýšil tak priemerný počet krokov na 23.8 (z pôvodných 12.5 krokov).

Metóda kompromisu na základe hodnotenia akcií taktiež dokázala výrazne zvýšiť počet remíz, resp. priemerný počet krokov. V prípade prvej skupiny dokázala navýšiť počet remíz od 496 po 520-krát (pre β od 1.0 po 5.0). V prípade druhej skupiny bol počet remíz až 122. Parameter N vo všetkých skupinách nehral významnú rolu a tak sa počty remíz menili vzhľadom na parameter β.

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.

4 Monte Carlo prehľadávacie stromy s využitím tenzorov

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 [1] a s využitím listovej, stromovej alebo koreňovej paralelizácie [7].

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 [33], 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 [19].

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) [15].

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 [29]. 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 politi-

kou 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 [8, 31, 12, 21]. Existuje viacero dostupných GitHub repozitárov s implementáciami založenými na rovnakom, resp. podobnom princípe [10, 28, 9, 11, 25, 20].

EfficientZero je vzorkovo efektívnejšia verzia pôvodného MZ [30]. 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.

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.

4.1 Popis metódy

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 $C_R = C_T + C_S \times C_T + 1$.

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).

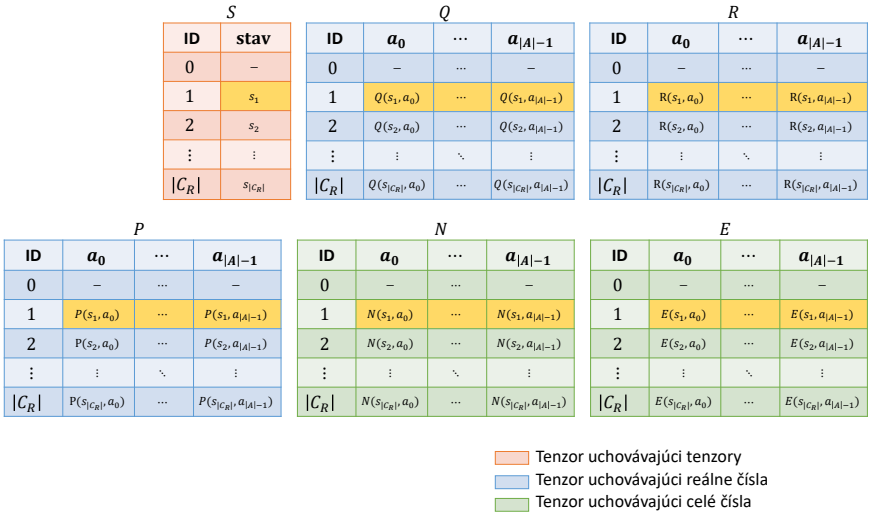
Úplná dátová štruktúra pozostáva zo šiestich tenzorov (obr. 1). Ako sme už spomenuli, prvý index každého tenzoru predstavuje ID (riadky) vrcholov. V tenzore S, index i vráti i-ty stav (vrchol s $ID = i$), pričom ostatné tenzory vrátia i-ty vektor. Napr. tenzor Q s indexom i vracia vektor q hodnôt $Q(i)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

Vďaka dátovej štruktúre je možné vykonávať všetky fázy MCTS simulácie pomocou vektorových operácií. Vzhľadom na obmädzený počet strán autoreferátu, pseudokódy sú uvedené a detailne popísané až v dizertačnej práci, resp. v publikovanom článku.

4.2 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 [29]. Implementácia vytvára nový proces pre každé bežiace prostredie. Každý proces okrem prostredia obsahuje aj kópiu NN. Implementácia využíva



Obr. 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.

knižnicu Ray, ktorá spravuje bežiace procesy. Implementácia bola použitá a citovaná vo viacerých výskumoch [8, 31, 12, 21] a na jej princípe existuje rada ďalších dostupných implementácií [10, 28, 9, 11, 25, 20]. 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.

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.

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.

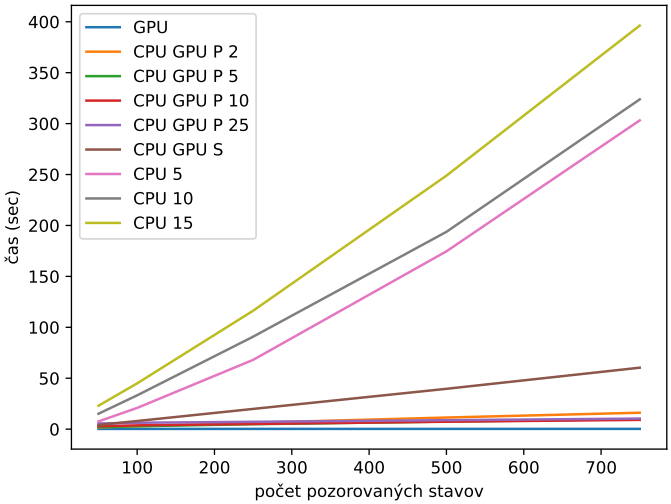
4.2.1 Výsledky

V prípade scenáru náhodnej akcie (obrázok 2), 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 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. 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 (obrázok 3) 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,



Obr. 2: časová náročnosť jednotlivých MCTS implementácií v závislosti od počtu pozorovaných stavov s aplikáciou scenáru náhodnej akcie bez využitia NN

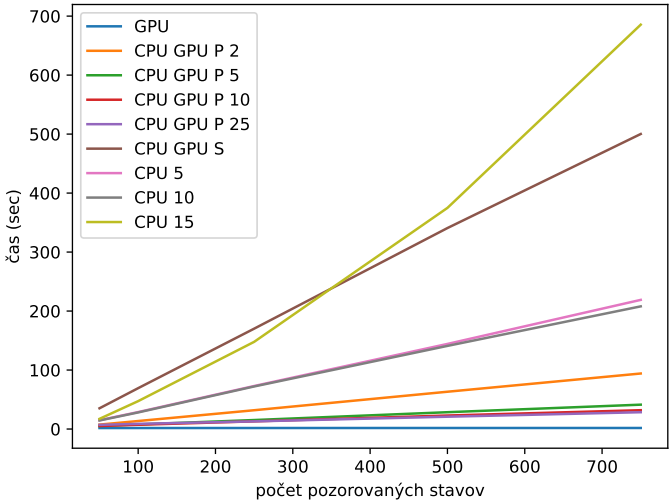
získaných počas fáze selekcie. Zvyšovanie dĺžky každej trajektórie je zapríčinené výberom konštantnej akcie (vždy sa prechádza rovnaká trajektória).

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 $C_T = 750$

Z druhého scenáru a vyššie uvedených násobkov rýchlosti opäť vyplýva, že navyšovaním počtu pozorovaných stavov sa násobne zvyšuje efektivita GPU imple-

mentácie voči ostatným. Samotný čas vykonania GPU implementácie pre $C_T = 750$ je len o 5% vyšší ako nameraný čas pre $C_T = 50$.



Obr. 3: časová náročnosť jednotlivých MCTS implementácií v závislosti od počtu pozorovaných stavov s aplikáciou scenáru konštantnej akcie bez využitia NN

5 MuZero s podporou prieskumu stavového priestoru

Modelovo založené algoritmy využívajú model prostredia buď pre podporu rozhodovania [23, 22, 16], alebo pre podporu prieskumu stavového priestoru [18, 32]. V poslednej kapitole sa zaoberáme návrhom modelovo založeného agenta, ktorý kombinuje oba prístupy.

V poslednej 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].

5.1 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.

$$\vec{q} = \frac{\beta_e Q_e^\pi(s,\cdot) + \beta_i Q_i^\pi(s,\cdot)}{\beta_e + \beta_i} \tag{8}$$

Pri aplikácii PUCT rovnice počítame q hodnoty ako kombináciu interných a externých hodnôt (rovnica 8). Samozrejme výsledný vektor q hodnôt je dôležité normalizovať do intervalu $[0,1]$.

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 [13] 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.

Kompletná modifikácia spolu so pseudokódom je opäť detailne popísaná v dizertačnej práci.

5.2 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, pričom čelí nástrahám ako pád, kolízia s nepriateľom a vstup do ohňa.

5.2.1 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, pričom modifikovaný MZ dosiahol skóre 19.93. Nižšie skóre je pravdepodobne zapríčinené vnútornou motiváciou, ktorá už naučeného agenta stále motivuje v prieskume stavového priestoru.

5.2.2 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.

6 Záver a zhrnutie prínosov

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 [4, 5].

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 [2].

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 [3]. 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.

Literatúra

[1] Ali, S., Plaat, A.e.a.: 'Parallel monte carlo tree search from multi-core to many-core processors', 2015

[2] Baláž, M., Tarábek, P.: 'Alphazero with real-time opponent skill adaptation', *2021 International Conference on Information and Digital Technologies (IDT)*, pp. 194–199, IEEE, 2021

[3] Baláž, M., Tarábek, P.: 'Tensor implementation of monte-carlo tree search for model-based reinforcement learning', *Applied Sciences*, vol. 13, no. 3, p. 1406, 2023

[4] Baláž Marek, T.P.: 'Double dqn architectures performance in pong', *Journal of Information, Control and Management Systems*, pp. 3–11, 2019

[5] Baláž Marek, T.P.: 'Effect of local rewards in games pong and lunar lander', *Mathematics in science and technologies conference*, pp. 19–24, 2020

[6] Berner, C., Brockman, G., et al.: 'Dota 2 with large scale deep reinforcement learning', *arXiv preprint arXiv:1912.06680*, 2019

[7] Chaslo, e.a.: 'Parallel monte-carlo tree search', 2008

[8] Gabirondo-López, J., Egaña, J., et al.: 'Towards autonomous defense of sdn networks using muzero based intelligent agents', *IEEE Access*, vol. 9, pp. 107184–107199, 2021

[9] Gras, J.: 'Muzero', https://github.com/johan-gras/MuZero, 2019

[10] Koul, A.: 'muzero-pytorch', https://github.com/koulanurag/muzero-pytorch, 2020

[11] Krishnamurthy, Y.: 'simple-muzero', https://github.com/yamsgithub/simple-muzero, 2020

[12] Lapan, M.: *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

[13] Li, B., Lu, T., et al.: 'Curiosity-driven exploration for off-policy reinforcement learning methods', *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 1109–1114, IEEE, 2019

[14] Linscott, G., et. al.: 'Leela chess zero', *Available: https://lczero.org/*, 2018

[15] Liu, A., Liang, Yitao, e.a.: 'On effective parallelization of monte carlo tree search.', 2020

[16] Lukasz Kaiser, Mohammad Babaeizadeh, P.M.e.a.: 'Model-based reinforcement learning for atari', *arXiv preprint arXiv:1903.00374*, 2019

[17] Mnih, V., Kavukcuoglu, K., et al.: 'Playing atari with deep reinforcement learning', *arXiv preprint arXiv:1312.5602*, 2013

[18] Pathak, D., Agrawal, P., et al.: 'Curiosity-driven exploration by self-supervised prediction', *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 16–17, 2017

[19] Rocki, K., Suda, R.: 'Large-scale parallel monte carlo tree search on gpu', 2011

[20] Schaposnik, F.: 'muzero', https://github.com/fidel-schaposnik/muzero, 2021

[21] Scholz, J., Weber, C., et al.: 'Improving model-based reinforcement learning with internal state representations through self-supervision', *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2021

[22] Schrittwieser, J., Antonoglou, I., et al.: 'Mastering atari, go, chess and shogi by planning with a learned model', *Nature*, vol. 588, no. 7839, pp. 604–609, 2020

[23] Silver, D., Hubert, T., et al.: 'Mastering chess and shogi by self-play with a general reinforcement learning algorithm', *arXiv preprint arXiv:1712.01815*, 2017

[24] Silver D., Huang A., M.C.e.a.: 'Mastering the game of go with deep neural networks and tree search', *Nature 529 (2016)*, pp. 484–489, 2016, doi:doi.org/10.1038/nature16961

[25] Sivaraj, M.: 'muzero', https://github.com/madhusivaraj/muzero, 2020

[26] Sutton, R.S., Barto, A.G.: *Reinforcement learning: An introduction*, MIT press, 2018

[27] Vinyals, O., B.I.C.W.e.a.: 'Grandmaster level in starcraft ii using multi-agent reinforcement learning', *Nature 575 (2019)*, p. 350–354, 2019, doi:doi.org/10.1038/s41586-019-1724-z

[28] Voskuil, K.: 'muzero', https://github.com/kaesve/muzero, 2021

[29] Werner Duvaud, A.H.: 'Muzero general: Open reimplementation of muzero', https://github.com/werner-duvaud/muzero-general, 2019

[30] Ye, W., Liu, S., et al.: 'Mastering atari games with limited data', *Advances in Neural Information Processing Systems*, vol. 34, 2021

[31] Yilmaz, E., Sanni, O., et al.: 'Deep reinforcement learning approach to air traffic optimization using the muzero algorithm', *AIAA AVIATION 2021 FORUM*, p. 2377, 2021

[32] Yuri Burda, Harrison Edwards, A.S.O.K.: 'Exploration by random network distillation', 2018

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

7 Zoznam vlastných publikácií autora

- Marek Baláž a Peter Tarábek. "*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.
- Marek Baláž a Peter Tarábek. "*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.
- Marek Baláž a Peter Tarábek. "*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.
- Marek Baláž a Peter Tarábek. "*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].