VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

HLUBOKÉ NEURONOVÉ SÍTĚ PRO POSILOVANÉ UČENÍ V REALTIMOVÉ STRATEGII

DEEP NEURAL NETWORKS FOR REINFORCEMENT LEARNING IN REAL-TIME STRATEGY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MARCO BARILLA**
AUTHOR

VEDOUCÍ PRÁCE — **MARTIN KOLÁŘ, M.Sc.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Barilla Marco**
Program: Informační technologie
Název: **Hluboké neuronové sítě pro posilované učení v realtimové strategii**
Deep Neural Networks for Reinforcement Learning in Real-Time Strategy
Kategorie: Umělá inteligence

Zadání:

1. Prostudujte základy neuronových sítí a posilovaného učení.
2. Vytvořte si přehled o současných metodách využívají neuronové sítě a posilované učení.
3. Vyberte konkrétní metodu aplikovatelnou na SC2LE (StarCraft II Learning Environment).
4. Nainstalujte a prostudujte SC2LE (StarCraft II Learning Environment), a identifikujte úlohy pro trénování
5. Implementujte navrženou metodu a proveďte experimenty nad vybranými úlohami.
6. Porovnejte dosažené výsledky a diskutujte možnosti budoucího vývoje.
7. Vytvořte stručný plakát prezentující vaši práci, její cíle a výsledky.

Literatura:

- Krizhevsky, A., Sutskever, I. and Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012
- https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0
- https://openai.com/systems/

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Kolář Martin, M.Sc.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Strojové učenie je jedna z najrýchlejšie napredujúcich odvetví dnešnej vedy. Je to podoblasť umelej inteligencie ktorá sa zaoberá problémom, ako pomocou počítačov riešiť komplexné moderné problémy. Vo vývoji tohto odvetvia hrajú dôležitú úlohu hru, pretože predstavujú optimálne prostredie na testovanie nových prístupov a ich porovnávanie so schopnosťami človeka. Jedna z hier ktoré sú v tejto oblasti stredobodom pozronosti je Starcraft 2, vďaka svojej širokej hráčskej základni a svojej komplexnosti. Praktickým cieľom tejto práce je vytvoriť advantage actor ctiric agenta, ktorý ktorý bude schopný operovať v prostredí tejto hry.

Abstract

Machine learning is one of the fastest growing branches of modern science. It is a subfield of artificial intelligence research that is interested the problem of making computers help us solve complex modern problems. Games play an important role in this field because they represent the perfect environment for testing of new approaches and benchmarking against human performance. Starcraft 2 is currently in the spotlight, thanks to its broad playerbase and its complexity. The practical goal of this paper is to create an advantage actor critic agent that is able to operate in the environment of this game.

Kľúčové slová

strojové učenie, učenie posilňovaním, hlboké neurónové siete, A2C, Starcraft 2, pysc2

Keywords

machine learning, reinforcement learning, deep neural networks, A2C, Starcraft 2, pysc2

Citácia

BARILLA, Marco. *Hluboké neuronové sítě pro posilované učení v realtimové strategii.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Martin Kolář, M.Sc.

Hluboké neuronové sítě pro posilované učení v realtimové strategii

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Martina Koláře M.Sc. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Marco Barilla
16. mája 2019

Poďakovanie

Chcel by som poďakovať môjmu vedúcemu práce, pánovi Martinovi Kolářovi M.Sc. za jeho ochotu, pomoc a odborné rady pri zhotovovaní práce.

Obsah

Kapitola 1

Úvod

Koncept umelej inteligencie je veľmi zaujímavá téma ako pre autorov beletrie, tak aj pre odborníkov ktorí v tomto odbore pracujú, pretože obsahuje obrovský potenciál kompletne reformovať viaceré sféry ľudskej spoločnosti.

Už od čias antického Grécka[17] sa ľudia zaoberali myšlienkami o inteligentných umelo vytvorených bytostiach. S technologickým pokrokom v 19. a 20. storočí sa táto téma v spoločnosti začala objavovať ako koncept čoraz častejšie, až kým nástup éry počítačov nepridal tejto problematike reálny rozmer.

Momentálne je výskum v oblasti umelej inteligencie jedna z najhorúcejších odvetví modernej vedy. Svätý grál tohto výskumu je vytvorenie všeobecnej umelej inteligencie, ktorá by bola vo všetkých smeroch porovnateľná s ľudskou. V roku 2017 sa na výskume všeobecnej umelej inteligencie podieľalo vo svete minimálne 40 organizácií[10]. Keďže sa ale zdá byť nereálne všeobecnú inteligenciu naprogramovať deterministicky, výskum sa začal rozvýjať rôznymi smermi.

Jeden z týchto smerov je strojové učenie. Je to podoblasť umelej inteligencie, ktorá sa zaoberá problematikou vytvárania matematických modelov, pomocou ktorých dokážeme zo vstupných dát získať určité predpovede. Strojové učenie má dnes široké uplatnenie vo viacerých oblastiach, napríklad vo vede, rôznych odvetviach komerčnej sféry alebo aj v medicíne.

Dôležitú úlohu vo vývoji strojového učenia mali historicky hry, ktoré slúžia ako veľmi vhodné prostredie na benchmarking rôznych prístupov a tiež prípadné súperenie s ľudským oponentom.

Spoločnosti Deepmind a Blizzard v roku 2017 vytvorili prostredie[4] na strojové učenie vo veľmi populárnej strategickej hre Starcraft 2, čo v tejto oblasti predstavuje dôležitý míľnik. Spolu s prostredím PySC2, vydali článok sc2le[23] ktorý slúži ako dokumentácia k tomuto prosterdiu a tiež popisuje experimenty ktoré tu boli vykonané.

Náplňou tejto práce bude popísať mnou implementovaný A2C (synchronous advantage actor critic) algoritmus a zdokumentovanie experimentovania v PySC2 prostredí.

Čo sa týka obsahu teoretickej časti, najprv bude bližšie popísaná teória problematiky strojového učenia, v rámci ktorej prejdeme konkrétnejšie na posilované učenie. Ďalej si v rámci posilovaného učenia priblížime metódy založené na funkcii hodnoty (value based methods) a tiež metódy založené na politike (policy based methods). Následne niečo o actor critic algoritmoch, čo je množina algoritmov do ktorej patrí v tejto práci implementovaný algoritmus A2C.

V poslednej časti teórie bude nasledovať bližší popis najprv strategickej hry Starcraft 2 a potom tiež aj prostredia PySC2 pre strojové učenie, v ktorom sa budú experimenty odohrávať.

V experimentálnej časti sú detaily ohľadom konkrétnej implementácie algoritmu a jeho návrhu, tiež dokumentácia jednotlivých experimentov a ich zasadenie do kontextu.

Na záver bude uvedené zhodnotenie a celkové zhrnutie výsledkov, ako úspechov, tak aj nedostatkov a ďalších potencionálnych možností vývoja.

Kapitola 2

Strojové učenie

Z ľudského pohľadu je učenie schopnosť získavať nové a modifikovať už nadobudnuté poznatky alebo skúsenosti. Túto schopnosť majú všetky zvieratá s komplexnejšou nervovou sústavou a dokonca existujú dôkazy aj o tom, že do určitej miery sú schopné sa učiť aj rastliny. Učenie teda predstavuje schopnosť dynamicky sa správaním prispôsobiť prostrediu, čo pre organizmy prináša evolučné výhody. Konktétne aj u ľudí, schopnosť efektívne sa učiť je jeden zo základných pilierov ľudskej inteligencie a nášho úspechu ako živočíšneho druhu. Nie je teda nič zvláštne na tom, že sa výskumníci snažili tento fenomén zreplikovať aj v oblasti počítačov.

2.1 Zrod strojového učenia

Odbor ako taký vznikol pôvodne ako jeden z nástrojov na výskum umelej inteligencie. Korene strojového učenia siahajú ešte pred rok 1959, kedy tento termín prvý krát použil v odborných kruhoch Arthur Samuel[20]. Neskôr pojem strojové učenie fromálne definoval Tom M. Mitchell. Zpočiatku sa tešilo relatívnemu záujmu vedeckých kruhov, avšak nástupom 70. rokov a obdobia skepticizmu voči umelej inteligencii, takzvaného "AI winter", upadol tento odbor na nejaký čas do pozadia[8].

Dôležitý miľnik však predstavovalo znovuobjavenie spätnej propagácie (backpropagation) v 80. rokoch. Vplyvom ďalšieho postupného zdokonaľovanía algoritmov a rapídnemu nárastu na výkone hardvéru, sa od prelomu tisícročí strojové učenie teší popularite ako vo výskumnej sfére, tak aj nasadzovaniu v praxi.

Obdobie najrapídnejšieho rozvoja je však posledných 10 rokov. Hlavný dôvod je graduálny nárast na výkone hardvéru, najmä čo sa týka grafických akcelerátorov, čo umožnilo industriálne nasadenie hlbokého učenia (deep learning), ktoré nebolo v minulosti kôli výpočtovej náročnosti efektívne použiteľné.

2.2 Praktické využitie

Podstata strojového učenia spočíva v schopnosti generalizovať na základe skúseností. To znamená že algorimus dokáže poskytnúť požadovaný výstup aj zo vstupov, na ktorých konkrétne trénovaný nebol. Vďaka tejto vlastnosti sú dnes rôzne verzie strojového učenia široko aplikované v rôznych sférach spoločnosti.

Jedna z najväčších oblastí je počítačové videnie. Zaoberá sa rozoznávaním objektov, pohybu v scéne, prípadne rekonštrukciou obrazu. Uplatnenie nachádza napríklad v bezpeč-

nostných systémoch na dynamické rozpoznávanie tváre, analýzu dopravy alebo na detekciu anomálií na snímkoch v medicíne.

Ďalej napríklad oblasť rozpoznávania zvuku. Využívjú ju osobní asisitenti ako Alexa alebo Siri, na zachytenie ľudkej reči a dekódovanie jej významu.

V praxi nájde využitie nájde množstvo ďalších odvetví strojového učenia, od precízneho ovládania robotických manipulátorov, cez algoritmy na analýzu finančných trhov až po autonómne riadiace vozidlá.

Obr. 2.1: **Tesla autopilot** zdroj: Tesla, Inc.

Kapitola 3

Učenie posilňovaním

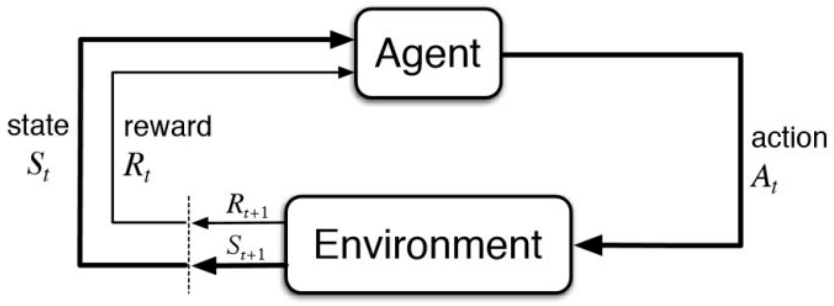
Strojové učenie sa delí podľa prístupu na 3 oblasti:

- Učenie s učiteľom
- Učenie bez učiteľa
- Učenie posilňovaním

Bližšie nás bude zaujímať učenie posilňovaním, pretože to tejto kategórie spadá A2C algoritmus implementovaný v tejto práci.

Učenie posilňovaním (po anglicky reinforcement learning, ďalej RL) je čo sa princípu fungovania týka, inšpirované spôsobom ako sa reálne učia zvieratá. Zvieratá majú tendenciu opakovať určité vzory správania ak im to prináša pozitívne výsledky, napríklad potravu, alebo vyhnutie sa predátorovi. Naopak, vzory správania ktoré im neprinášajú nič, respektíve majú na zviera negatívny dopad, sa väčšinou snažia eliminovať.

No a na podobnom princípe fungujú aj RL algorimy. Sú špecifické tým, že sa učia samostatne, iba na základe interakcie s prostredím[21]. Všeobecný princíp fungovania je že algoritmus v danom stave volí akciu, za ktorú môže pri prechode do nasledujúceho stavu získať odmenu.



Obr. 3.1: **Model interakcie medzi RL agentom a prostredím** zdroj: [21]

3.1 Markovov rozhodovací proces

Markovov rozhodovací proces nám poskytuje matematicky vymedzený rámec na modelovanie rozhodovania v stochastických prostrediach. Predstavuje teda teoretický základ ktorým možme formálne popísať princíp RL[6].

Tvorí ho usoriadaná štvorica <S,A,P,R>:

- S predstavuje množinu všetkých stavov s
- A predstavuje množinu všetkých stavov a
- $P(S, A, S) \rightarrow [0, 1]$ kde $P(s'|a, s)$ je pravdepodobnosť že sa dostaneme do $s' \in S$ pri počiatočnom stave $s \in S$ a zvolení akcie $a \in A$
- $R(S, A, S) \rightarrow \mathbb{R}$ kde $R(s, a, s')$ je funkcia odmeny keď akciou $a \in A$ prejdeme do stavu $s' \in S$ zo stavu $s \in S$

V tomto modeli platí Markovovká vlastnosť, ktorej význam je že stav $s \in S$ úplne popisuje stav prostredia a ďalší stav a odmena sú závyslé iba od zvolenej akcie v danom stave a prechodovej funkcie.

Za týchto podmienok je cieľom RL agenta nájsť optimálnu stratégiu π tak, aby maximalizoval kumulatívnu odmenu. Kumulatívne odmeny (returns) sú definované ako:

$$\sum_{t=1}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1})$$

kde γ je diskontný faktor pre ktorý platí: $1 \geq \gamma > 0$ a jeho hodnota určuje, nakoľko sú relevantné odmeny z predchádzajúcich prechodov.

3.2 Metódy založené na funkcii hodnoty

Úloha RL algoritmov je založená na hľadaní takej optimálnej stratégie $\pi(s)$[9]:

$$\pi^* = argmax_\pi V^\pi(s)$$

tak, aby bola maximalizovaná funkcia hodnoty stavu:

$$V^\pi(s_t) = R_t = \sum_{k=1}^{T} \gamma^k r_{t+k}$$

Avšak v praxi sa tento prístup má potencionálny problém v skutočnosti, že nepoznáme prechodovú funkciu. Z tohto dôvodu sa častejšie používa funkcia takzvanej Q-hodnoty, ktorá priradzuje hodnotu páru stavu a akcie. Teda vzťah pre stratégiu je:

$$\pi^* = argmax_a Q(s, a)$$

pričom platí:

$$V^*(s) = max_a Q(s, a)$$

Teda, namiesto hľadania π ktorá by maximalizovania hodnotu pre všetky stavy, hľadá akciu ktorá by maximalizovala Q-hodotu pre všetky stavy. Ak teda poznáme pre každú akciu ďalší stav a odmenu, môžeme aplikovať iteratívny algoritmus na naučenie Q-hodnôt:

$$Q(s,a) = r + \gamma max_{a'} Q(s',a')$$

Tento vzťah sa nazýva Bellmanova rovnica, čo je podstata relatívne často používaného Q-učenia. Q-učenie sa snaží aproximovať hodnotu ako funkciu pre aktuálny stav a akciu.

3.3 Metódy založené na politike

Keďže Q-učenie mapuje hodnoty ku párom (s,a), je tento prístup použiteľný buď ťažko alebo vôbec ak máme buď veľký počet možných akcií, prípadne ak je akčný priestor spojitý. Jeden z alternatívnych prístupov ktoré sú vhodné pre riešenie týchto problémov sú metódy založené na politike, čo znamená že sa snažia priamo voliť optimálnu akciu namiesto určovania hodnoty pre stav. Tieto metódy však tiež majú svoje nedostatky, napríklad majú tendenciu často konvergovať k lokálnemu miesto globálnemu maximu a sú menej stabilné, aj keď existujú spôsoby ako tento nedostatok minimalizovať[19].

Úloha je aj v tomto prípade optimalizácia politiky aby kumulatívne odmeny boli čo najvyššie[9]:

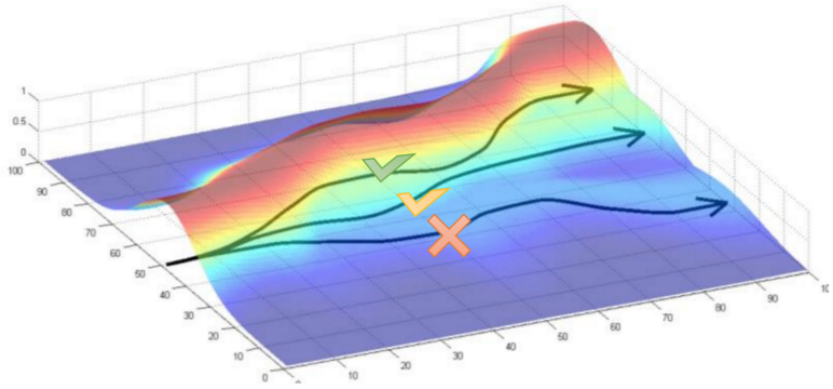
$$\pi^* = argmax_{\pi} R_t$$

Vďaka parametrizácii môžme použiť neurónovú sieť na naučenie funkcie politiky:

$$\pi(a_t, s_t) \rightarrow \pi(a_t, s_t, \theta)$$

Učením sa politiky v tomto prípade rozumieme maximalizáciu určitej funkcie $J(\theta)$ pomocou techniky "gradient ascent", čo je aplikácia gradientu na parameter θ. Vzťah na výpočet gradientu je:

$$\Delta J(\theta) = \mathbb{E}_{\pi}[\Delta_{\theta} \log \pi(a_t|s_t, \theta) R_t]$$



Obr. 3.2: **Znázornenie optimalizácie trajektórie pomocou gradientu** zdroj: [15]

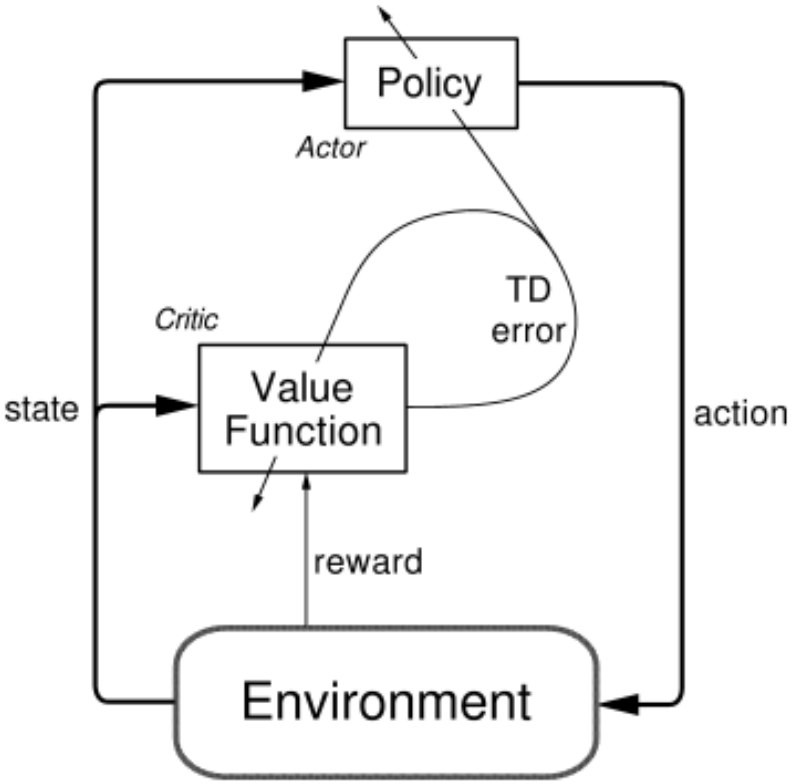
3.4 Metódy Actor Critic

Podstata tejto metódy spočíva v spojení oboch predošlých prístupov. Ich kombináciou sme schopní odstrániť niektoré nedostatky oboch predošlých prístupov[24]. Metóda pozostáva z dvoch podstatných častí:

- Actor -Rreprezentuje politiku, Actor je zodpovedný za výber akcie v danom stave.
- Critic - Je funkcia hodnoty, hodnotí buď priamo stav, alebo určuje Q hodnotu pre dvojicu (s,a).

Napríklad rovnicu pre metódu Q Actor Critic získame pridaním Q hodnoty do rovnice na výpočet gradientu:

$$\Delta J(\theta) = \mathbb{E}_\pi[\Delta_\theta \log \pi(a_t|s_t, \theta) Q^\pi(s_t, a_t)]$$



Obr. 3.3: **Model Actor Critic algoritmu** zdroj: [16]

3.5 Metóda A2C

Kompletný názov metódy je po anglicky synchronous advantage Actor Critic. A2C je matematicky v podstate zhodný s algoritom A3C (asynchronous advantage actor critic) ktorý je veľmi dobre spracovaný v článku od Google Deepmind[18].

Rozdiel medzi nimi je len v praktickej implementácii synchronizácie paralelných agentov, teda je pre nás tento zdroj relevantný.

Jedna z podstatných výhod A2C je schopnosť jednoduchej paralelizácie. Princíp fungovania je že n agentov vygeneruje v rámci batchu interakcie s prostredím, na jeho konci sa agregujú dohromady, vypočíta sa z nich gradient a ten sa aplikuje na parametre θ.

A2C je partí do množiny Actor Critic algoritmov a teda obsahuje ako zložku Actor predstavujúcu politiku výberu akcie v danom stave π_s, tak aj časť Critic, ktorá reprezentuje funkciu hodnoty $V_\pi(s)$ v danom stave.

Navyše však A2C zavádza pojem advantage funkcia, ktorá je definovaná nasledovne:

$$A(s_t, a_t, \theta, \theta_v) = \sum_{i=0}^{k-1} T\gamma^i r_{t+i} + \gamma^k V(s_{t+k}, \theta_v) - V(s_t, \theta_v)$$

Podstata advantage funkcie je v tom, že vďaka nej vieme retrospektívne zistiť o koľko boli akcie v konkrétnych stavoch zvolené Actor-om lepšie alebo horšie v porovnaní s tým, čo Critic očakával.

Rovnica pre výpočet gradientu na optimalizáciu pomocou gradient ascent teda vyzerá následovne:

$$\Delta J(\theta) = \Delta_{\theta'} \log \pi(a_t|s_t, \theta') A(s_t, a_t, \theta, \theta_v)$$

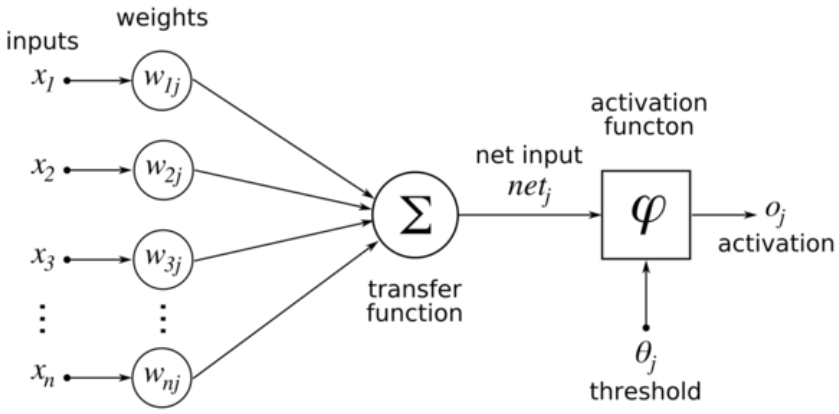
V praxi sa ku gradientu ešte pridáva takzvaná entropia, ktorá algoritmu pomáha s riešením exploration/exploitation problému. Exploration je náhodné prehľadávanie stavového priestoru s cieľom nájsť optimálnejšie stratégie a exploitation je nasledovanie aktuálne optimálnej politiky. Pri zlom vyvážení týchto prístupov agent môže buď uviaznuť v lokálnom maxime pri nedostatočnom náhodnom prehľadávaní, alebo nebyť schopný sa naučiť komplexnejšie stratégie kôli príliš veľkému pomeru náhodných akcií.

V praxi používaná stratégia na riešenie tohto problému je začať s pomerne vysokou šancou na náhodnú akciu a postupne ju s priebehom tréningu znižovať.

Kapitola 4

Umelé neurónové siete

Umelé neurónové siete siete sú výpočtové modely vágne inšpirované bilogickými neurónovými sieťami. Ako aj ich biologický vzor, pozostávajú z množiny prepojení a uzlov, uzly sa nazývajú neuróny. V prípade biologických sietí sú neuróny bunky, pri umelých to však sú matematické funkcie s n vstupmi a jedným výstupom, pomocou ktorých sú tieto neuróny prepojené. Neuróny majú na každom prichádzajúcom prepojení modifikátor váhy vstupnej hodnoty, prahovú hodnotu a môžu mať aktivačnú funkciu.



Obr. 4.1: **Umelý neurón** zdroj: en.wikibooks.org

V praxi sa zvyčajne vo vnútorných vrstvách používajú aktivačné funkcie: Tanh, Sigmoid, ReLU[14]:

$$Tanh : f(x) = \frac{e^x - e^{-x}}{e^x - e^{-x}}$$

$$Sigmoid : f(x) = \frac{1}{1 + e^{-x}}$$

$$ReLU : f(x) = max(0,\ x)$$

Tanh a Sigmoid sú matematicky priaznivé pretože sú spojité a diferncovateľné, avšak majú problém ktorý sa nazýva "vanishing gradient" čo je miznutie gradientov s postupným nárastom vstupných hodnôt.

Tento problém nemá aktivačná funkcia ReLU. Navyše je veľmi výpočtovo efektívna takže pri komplexných neurónových sieťach kde je rýchlosť dôležitá, si ReLU nájde svoje miesto. To však neznamená že je bez nevýhod, neuróny s ReLU aktivačnou funkciou niekedy môžu takzvane "zomrieť", čo znamená že budú na výstupe dávať už len hodnotu 0 bez možnosti ďalšej zmeny. Existujú aj varianty ako napríklad Leaky ReLU, ktoré tento nedostatok nemajú, ale cena za to je vyššia výkonová náročnosť.

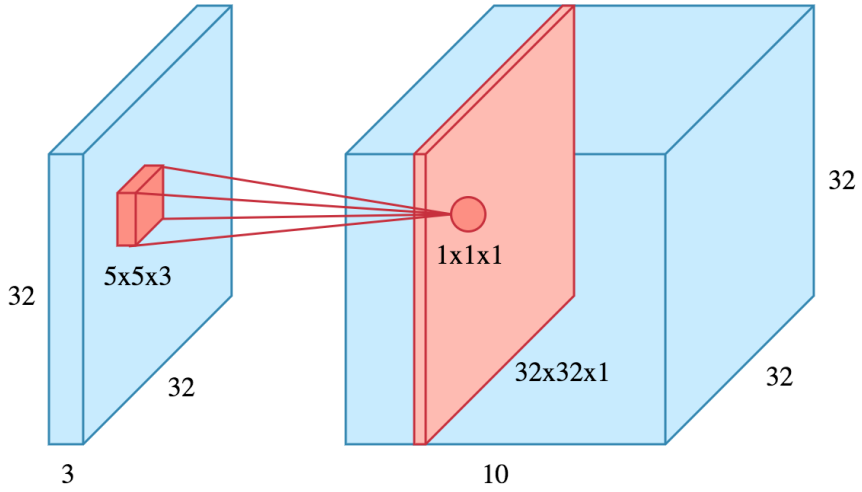
Na výstupe z neurónovej siete môžeme napríklad použiť Tanh ak chceme číselné hodnoty z rozsahu od -1 do 1, SoftMax na pravdepodobnostné rozloženie alebo nemusíme aktivačnú funkciu použiť žiadnu.

4.1 Konvolučné neurónové siete

Konvolučnými neurónovými sieťami sa nazývajú hlboké neurónové siete ktoré obsahujú konvolučné vrstvy. Tieto konvolučné vrstvy sa spravidla nachádzajú na vstupných vrstvách a sú navrhnuté špecificky pre spracovávanie vizuálneho vstupu.

Konvolučná vrstva pozostáva zo sady naučiteľných filtrov (kernelov) o veľkosti $N \times M \times C$ pričom N a M sú rozmery filtra a C je hĺbka obrazového vstupu. Tieto filtre sú následne po vstupnom obraze posúvané o krok K formou posuvného okna a pri každom posunutí je počítaný skalárny súčin a výstup je dvojdimenzionálna aktivačná mapa filtra[2]. Vo výsledku sa konvolučná vrstva pomocou filtra učí detekovať vo vstupe určité vzory.

Konvolučné vrstvy bývajú zvyčajne striedané s pooling vrstvami, ktoré znižujú dimenzionalitu dát kombinovaním niekoľkých výstupov neurónov predošlej vrstvy do jedného výstupu.



Obr. 4.2: **Znázornenie konvolúcie**, zdroj: [12]

4.2 Neurónové siete ako aproximátor funkcie

Jednoduché problémy z oblasti RL s relatívne malým počtom možných stavov sa dajú riešiť aj priamym mapovaním akcie na stav. Problém však nastáva ak je stavový priestor pre tento prístup príliš veľký. V tomto prípade je vhodné použiť aproximátor funkcie, ktorý je po natrénovaní schopný mapovať konkrétne výsledky na skupiny podobných vstupov. Takýto aproximátor sú práve neurónové siete.

Neurónové siete predstavujú v dnešnej dobe stavebný kameň praktickej implementácie nie len v RL, ale aj mnohých iných odvetviach strojového učenia.

4.3 Funkcia chyby

Častejšie sa stretneme s anglickým názvom loss function. Funkcia chyby je nevyhnutná pre fungujúce učenie nerónovej siete. Vo svojej podstate funkcia chyby pracuje na jednoduchom princípe, čím dáva trénovaný model nepresnejšie výsledky, tým väčšie hodnoty produkuje funkcia chyby, teda na základe funkcie chyby vieme určiť rozdiel medzi očakávaným výsledkom a aktuánym výsledkom.

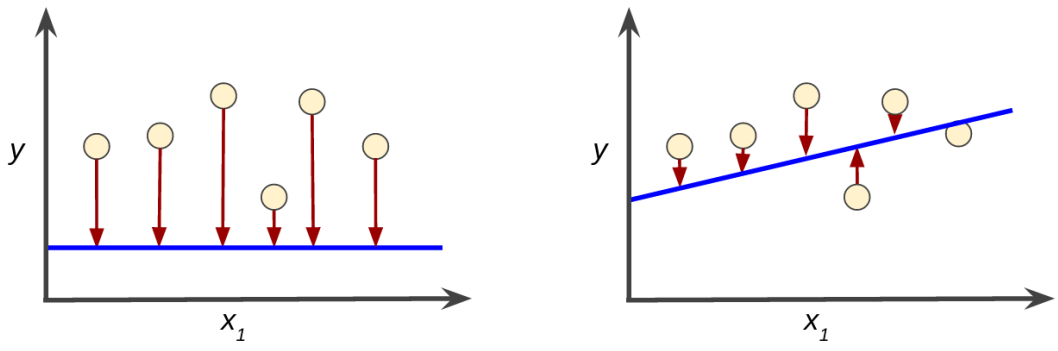
V praxi sa používajú rôzne varianty loss funkcií, napríklad:

- Funkcia strednej štvorcovej chyby (anglicky mean square error):

$$E = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

- Logaritmická chyba (nazýva s aj krížová entropia, anglicky cross entropy):

$$E = -(ylog(p) + (1-y)log(1-p))$$



Obr. 4.3: **Ukážka funkcie chyby**, červené šípky reprezentujú hodnotu chyby, modrá čiara reprezentuje predikciu modelu, napravo je model natrénovaný, hodnota chyby je menšia. zdroj: [5]

4.4 Spätná propagácia

Spätná propagácia (anglicky Backpropagation) je metóda, ktorá zodpovedá za učenie v neurónových sieťach. Princíp fungovania tejto metódy je postavený na spätnej propagácii chyby medzi vrstvami.
Samotný proces spätnej propagácie pozostáva z dvoch častí:

- Propagácia - Pre konkrétne vstupy siete sa prepočítajú výstupy a pomocou funkcie chyby sa určí celková chyba siete. Na základe tejto chyby získame spätnou propagáciou delty pre konkrétne váhy v sieti. Delty predstavujú vplyv na chybu.
- Aktualizácia váh - Výpočet hodnoty inkrementu váh na základe hodôt delta. Nasleduje aplikácia inkrementu na váhy.

V praxi sa však o spätnú propagáciu starajú "za oponou" knižnice ako napríklad Tensorflow alebo Pytorch.

Kapitola 5

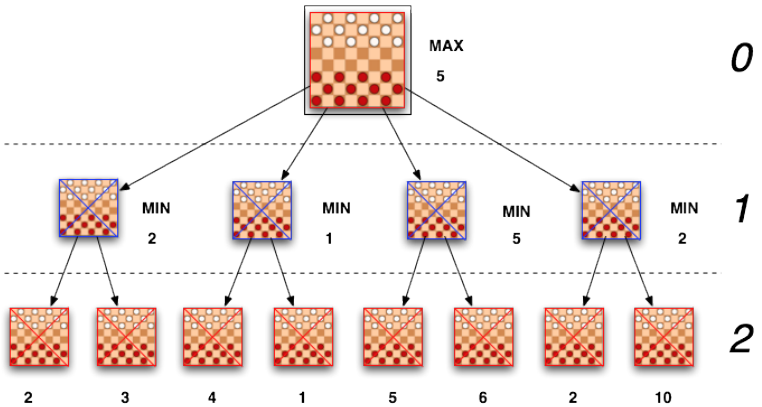
Starcraft 2

Starcraft 2 je druhá iterácia veľmi populárnej sci-fi strategickej série od spoločnosti Blizzard Entertainment. Prvá verzia vyšla v roku 1998 a prakticky hneď po svojom vydaní sa stala najpopulárnejšou hrou na najbližších pár rokov. Okolo starcraftu sa vytvorila jenda z najväčších a najstabilnejších kompetitívnych herných scén, a teda sa považuje za jeden z najvýznamejších titulov ktoré hrali úlohu v zrode elektronického športu.

5.1 Hry ako prostriedok na výskum umelej inteligencie

Hry predstavujú dôležitú súčasť vývoja umelej inteligencie. Predstavujú totiž pre agenta prostredie ktoré má jasne definované pravidlá a má väčšinou aj jasne definovaný cieľ. Tiež sú experimenty ľahko opakovane zreplikovateľné. Pragmatická výhoda je, že herný priemysel v dnešnej dobe je veľké odvetvie, produkované hry sú komplexné simulácie a teda to výskumníci môžu využiť. Keďže moderné hry sú hrané tisíckami hráčov a mnohé z nich majú aj aktívnu kompetitívnu scénu, sú vynikajúce prostredie na porovnávanie algoritmov s ľudským oponentom.

Počiatky pokusov implementácie rôznych agentov na hranie hier siahajú do 50. rokov[1]. Jeden z prvých krokov na tomto poli bol agent ktorý dokázal pomerne efektívne hrať dámu pomocou algoritmu Minimax[20].



Obr. 5.1: **Ukážka Minimax algoritmu**, zdroj: University of Porto

Ďalší vývoj vykulminoval až do momentu v roku 1997 keď sa programu Deep Blue od IBM podarilo prvý krát poraziť úradujúceho šampióna v šachu Garryho Kasparova. Ďalší podobne prelomový úspech na tomto poli prišiel až o 19 rokov neskôr, keď sa podarilo poraziť vo veľmi komlexnej hre Go jedného z najlepších hráčov, Lee Sadola. Tento algoritmus bol vyvinutý spoločnosťou Google Deepmind, pod názvom AlphaGo[3] a využíva kombináciu strojového učenia a prehľadávania stromov.

Od roku 2016 nabral výskum na tomto poli na obrátkach a v auguste 2017 sa podarilo dosiahnuť ďalší významný míľnik. Spoločnosť OpenAI založená Elonom Muskom vytvorila agenta schopného hrať populárnu hru Dota 2. Tento agent dokázal konzistentne porážať najlepších hráčov v 1 proti 1 scenári. Agent sa hrať učil úplne od podlahy iba z hry samého so sebou úplne autonómne. Je to pozoruhodný výsledok z dôvodu, že Dota 2 má obrovský stavový priestor a na rozdiel od stolných hier sa odohráva v reálnom čase, je teda potrebné zvládnuť precízne načasovať koordináciu jednotlivých akcií.

Obr. 5.2: **Snímok obrazovky z hry proti OpenAI agentovi v hre Dota 2**

V roku 2017 upriamil Google Deepmind svoju pozronosť na Starcraft 2. Pri predstavení tohto cieľa zverejnili v spolupráci so spoločnosťou Blizzard entertainment sprístupnili Starcraft 2 api a predstavili PySC2 prostredie špeciálne navrhnuté pre strojové učenie. Ako cieľovú métu si nastavili vytvorenie agenta, ktorý dokáže poraziť aj najlepších ľudských hráčov. Tento cieľ sa im vo februári tohto roku podarilo čiastočne naplniť, keď po prvý krát predstavili výsledky svojej dvojročnej práce[22]. Agent nazvaný Alphastar odohral proti dvom profesionálnym hráčom dohromady 11 kompletných hier s veľmi priaznivou konečnou bilanciou 10-1. Toto je prelomový úspech, keďže je to prvý krát, čo bol agent schopný poraziť človeka v plnej hre, avšak problém Starcraftu ešte vyriešený nie je. Agentovi by sa dali vytknúť 2 nedostatky. Agent je schopný hrať iba vo formáte jeden proti jednému a obe rasy musia byť protoss. Druhý detail je že v rozhodujúcich momentoch boja mal agent neúmerne vysoké počítadlo akcií za sekundu, rozhraním ktoré používa ľudský hráč prakticky nedosiahnuteľné.

5.2 Princíp hry Starcraft 2

Starcraft 2 je stratégia odohrávajúca sa v reálnom čase. V hre existujú 3 rozdielne rasy, Terran (ľudia), Protoss (technologicky vyspelí mimozemšťania) a Zerg (rasa biologických mutantov). Hráč si musí pred začiatkom hry jednu z nich zvoliť. Rasy sú asimetrické, to znamená že majú úplne odlišné jednotky a tiež aj odlišné budovy. Každá rasa má svoje silné a slabé stránky, napríklad Protoss má spravidla najsilnejšie jednotky ktoré majú regenerujúci sa štít, sú však najdrahšie na výrobu. Naopak Zerg je špecifický stratégiou "zerg rush" keďže má jednotky lacné a dokáže ich vyrobiť za krátky čas veľké množstvo, sú však slabšie. Terran je charakteristický tým, že nemá výrazné slabiny, avšak ani v ničom typicky nevyniká. Všetky rasy majú široký repertoár jednotiek, pričom väčšina z nich zapadá do kameň-papier-nožnice schémy, teda každá jednotka má svoje silné a slabé stránky.

Hra sa odohráva na hracej mape, na ktorej sú rozmiestnené surovinové náleziská o ktoré sa bojuje.

Obr. 5.3: **Na obrázku je začiatok hry, v strede je hlavná budova a jednotky ťažia suroviny z náleziska, v ľavom dolnom rohu je minimapa**

V hre je primárny cieľ zničiť všetky budovy nepriateľa, akonáhle jeden z hráčov príde o poslednú budovu, hra v momente končí a je jasný víťaz. V praxi však väčšinou je potrebné najprv poraziť nepriateľovu armádu, a na to optimálne potrebujeme silnejšiu armádu ako nepriateľ. No a na vyprodukovanie silnejšej armády potebujeme pozbierať viac surovín.

Rozhodnúť sa však akú stratégiu v momente zvoliť nie je jednoduché, pretože hra poskytuje informácie iba o prostredí v blízkosti našich jednotiek, a teda nevieme čo robí nepriateľ. Preto lepší hráči vysielajú periodicky smerom k súperovi jednotky za účelom zistiť čo súper robí.

V hráčskej terminológii sa herná stratégia skladá z dvoch zložiek:

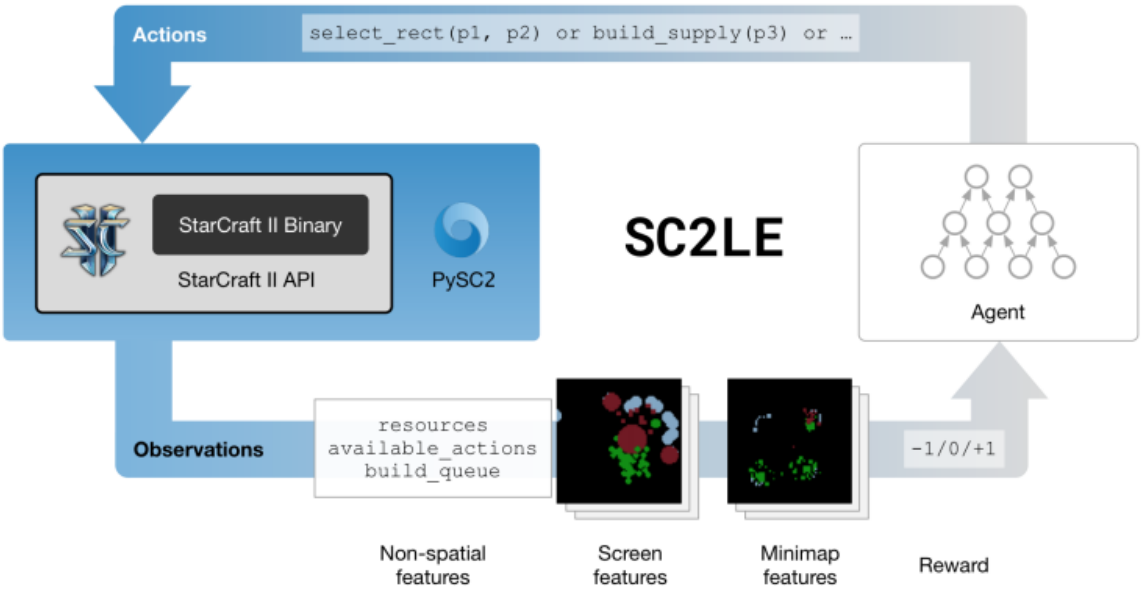
- Macro - Predstavuje dlhodobú stratégiu hry, teda ktoré budovy a kde postaviť, ktoré surovinové náleziská zabrať, aké bojové jednotky postaviť a aké technológie skúmať - tieto rozhodnutia majú dlhotrvajúci dopad na stav hry.
- Micro - V podstate ovládanie jednotiek v boji, snaha o získanie taktickej výhody nad súperom, využívanie silných stránok vlastných jednotiek - kedže sa boje odohrávajú pomerne rýchlo, je tu dôležitý faktor APM čo sú akcie za minútu.

Rozdiel medzi dobrým a excelentným hráčom je často v počte akcií za minútu ktoré dokáže vyprodukovať. Ak obaja hráči hrajú strategicky solídne, rozhodujúci je zvyčajne konflikt armád oboch hráčov a ten z nich, ktorý je lepšie schopný získať čo i len malú výhodu vďaka rýchlejším prstom, má väčšie šance na víťazstvo.

Kapitola 6

Prostredie PySC2

V tejto kapitole si popíšeme ako agent, teda v tomto konkrétnom prípade A2C algoritmus, komunikuje s hrou Starcraft 2. PySC2 rozhranie je na hru napojené prostredníctvom api ktorú sprístunili priamo vývojári hry. Táto api slúži na programovú komunikáciu s hrou, a je možné prostredníctvom nej naprogramovať agenta. Avšak v rámci výskumu strojového učenia je pre nás PySC2 rozhranie relevantnejšie, pretože sa snaží sprístupniť hru pre agenta čo najbližšie tomu, ako ju vníma hráč.



Obr. 6.1: **Komunikácia medzi agentom a prostredím** zdroj:[23]

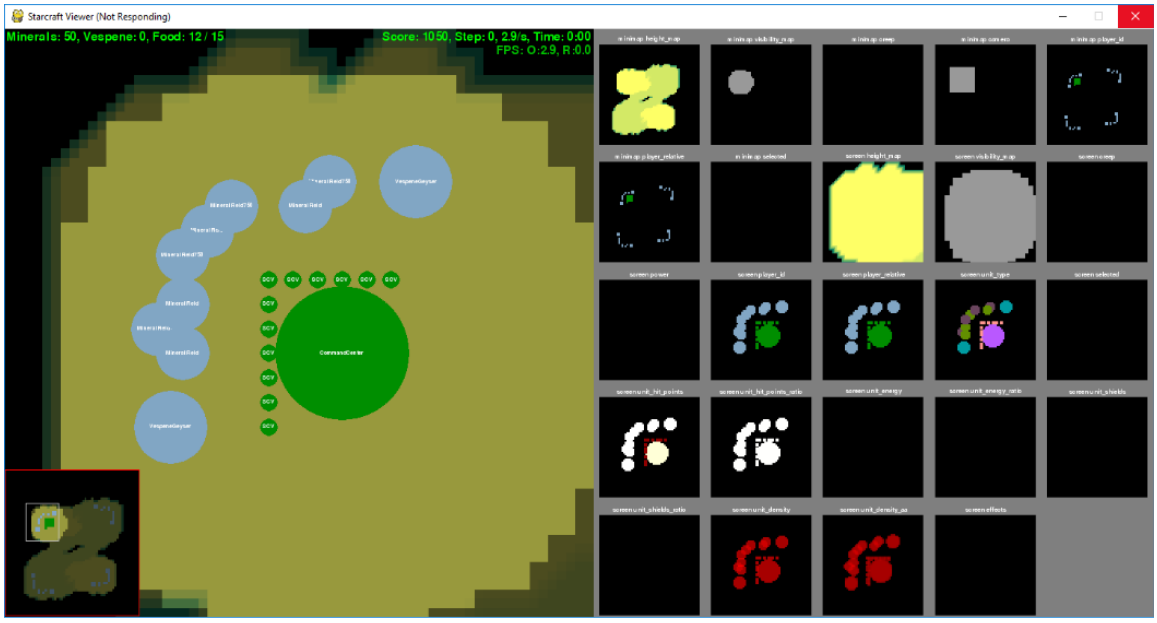
6.1 Obrazový vstup

Ľudský hráč má pri hraní starcraftu k dispozícii vizuálnu reprezentáciu hry na obrazovke. Tá pozostáva z konkrétnej časti mapy, ktorá zaberá väčšinu obrazovky. Naľavo dole je minimapa ktorá reprezentuje jednoduchou formou stav celej mapy a po celej spodnej strane obrazovky je lišta s rozširujúcimi informáciami o jednotkách a sprístupňuje rôzne akcie. V pravom hornom rohu sú informácie o stave surovín populácie jednotiek.

Takéto rozloženie, pomerne jednoducho spracovateľné pre človeka, je pre počítačové videnie nie príliš priaznivé. Problém hlavne predstavujú číselné hodnoty a rôzne ikonky, kde aj stredne skúsený hráč niekedy nevie čo znamnajú, kým si neprečíta rozširujúce informácie po nabehnutí myšou.

Stav o hre teda PySC2 poskytuje spôsobom, ktorý je podobný, ako to vidí hráč, len lepšie spracovateľný pre strojové videnie. Stav o obrazovke a minimape je poskytovaný vo vrstvách. Vrstvy zachovávajú rozlíšenie, teda aj priestorovú reprezentáciu. Každý bod vo vrstve poskytujú číselnú hodnotu o informácii ktorú reprezentuje. Napríklad vrstva "player_relative" na konkrétnom bode má hodnotu 0 ak sa tam nenachádza žiadna jednotka, hodnotu 1 ak je to vlastná jednotka, 2 pre neutrálnu a 3 pre nepriateľskú jednotku. Podobne existujú vrstvy s informáciou o výške terénu, pozícii kamery, alebo vybratých jednotkách.

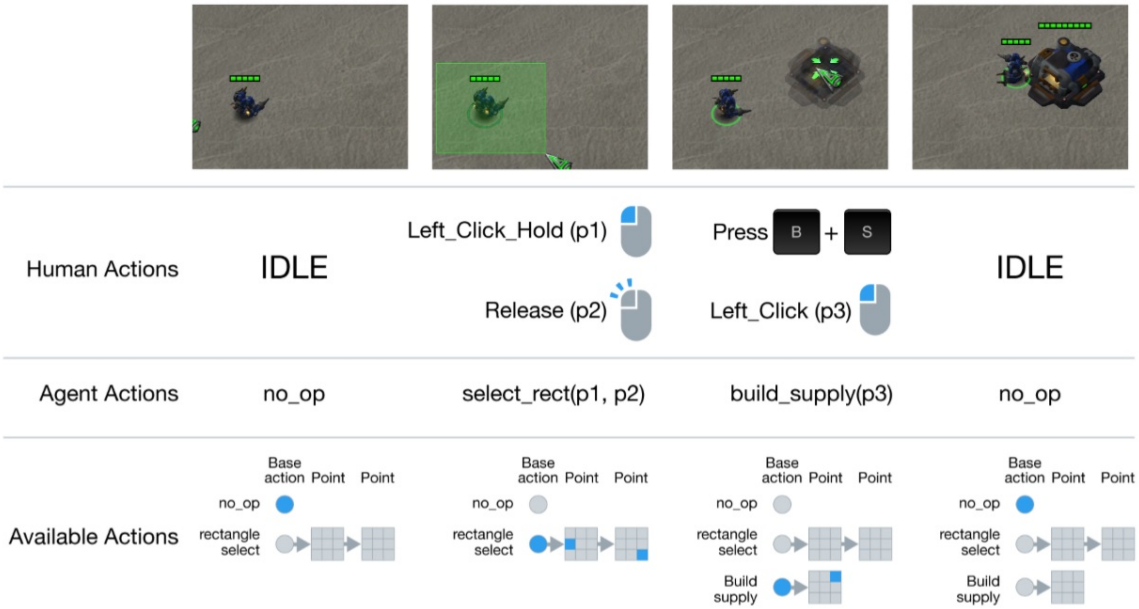
Okrem pozičných informácií z mapy poskytuje prostredie skalárne hodnoty reprezentujúce stav prostredia prostredníctvom vektora číselných hodnôt.



Obr. 6.2: **Snímok obrazovky zobrazuje jednotlivé priestorové vrstvy ako ich poskytuje PySC2**

6.2 Akcie v PySC2

Starcraft 2 má veľmi veľký akčný priestor. Na volanie akcie PySC2 poskytuje Function-Call() funkciu. Ako prvý argument berie jedno z 524 ID kde každé reprezentuje konkrétny druh akcie. Situáciu nám však podstatne komplikuje fakt že tieto akcie majú ešte vlastné parametre. Napríklad, jedna z možných akcií je útok, ten však berie súradnice obrazovky ako parameter, aby jednotky vedeli kam zaútočiť. Keď je teda rozlíšenie napríklad 64x64, tak máme len pre jednu z 524 akcií 4096 rôznych variant.



Obr. 6.3: **Porovnanie použitia akcií pri štandardnej hre a pri hre cez PySC2** zdroj:[23]

Kapitola 7

Návrh A2C agenta

Z predchádzajúcich kapitol je zjavné, že strategická hra Starcraft 2 predstavuje pre odvetvie strojového učenia veľmi komplexný problém, podstatne prevyšujúci svojou komplexitou hry ako Go, nehovoriac o šachu. Deepmindu sa ho však po dvoch rokoch podarilo vyriešiť s použitím komplexnej architektúry a najnovších technológií[22]. Trénovanie prebiehalo na serveroch Google a agent bol trénovaný na 200 rokoch Starcraft2 herného času.

Cieľom tejto práce však nie je vytvorenie agenta schopného hrať plnú hru, pretože to je príliš komplexný problém aj algoritmicky, aj čo do výpočtovej náročnosti.

Úloha agenta v tejto práci bude navigácia v prostredí minihier špeciálne navrhnutých ako vstupný problém do tejto problematiky.

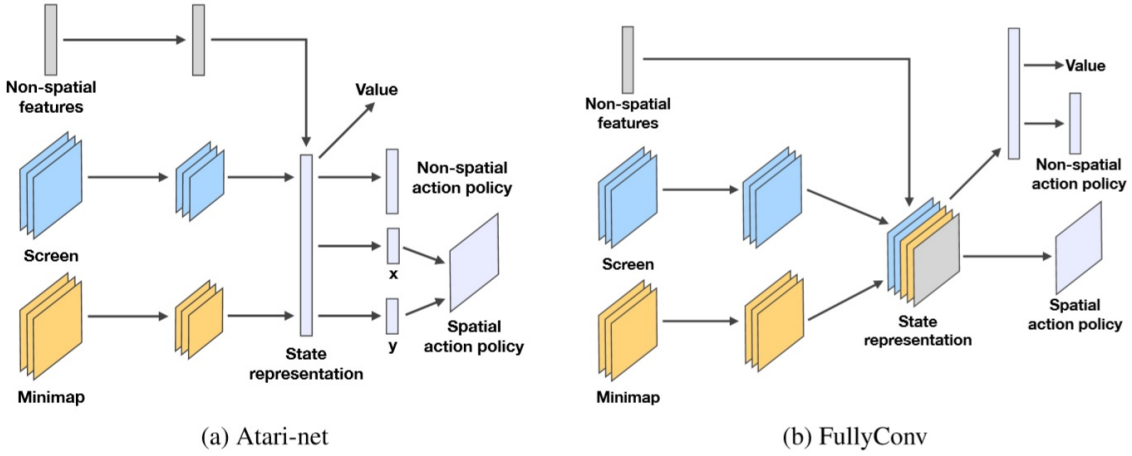
7.1 Agenti použití v SC2LE

Aj keď Deepmind nezverejnil zdrojové kódy agentov použitých v ich experimentoch, v SC2LE[23] je k dispozícii popis ich architektúry. Strojové učenie je jedna z najrýchlejšie napredujúcich odvetví dnešnej vedy. Je to podoblasť umelej inteligencie ktorá sa zaoberá problémom, ako pomocou počítačov riešiť komplexné problémy. Vo vývoji tohto odvetvia hrajú dôležitú úlohu hru, pretože predstavujú optimálne prostredie na testovanie nových prístupov a ich porovnávanie so schopnosťami človeka. Jedna z hier ktoré sú v tejto oblasti stredobodom pozronosti je Starcraft 2, vďaka svojej širokej hráčskej základni a svojej komplexnosti. Cieľom tejto práce je vytvoriť A2C agenta, ktorý ktorý bude schopný navigovať v prostredí tejto hry. V dokumente sú vypísané 3 rôzne implementácie:

- Atari-Net Agent - Táto architektúra je pôvodne úspešne použitá v atari benchmarku [11]. Spracováva vizuálne dáta aj z obrazovky, aj z minimapy dvomi konvolučnými vrstvami o 16, 32 filtroch, veľkosti 8, 4 a krokom 4, 2. Vektor so skalárnymi hodnotami o stave prostredia je spracovaný jednou lineárnou vrstvou s aktivačnou funkciou Tanh. Ďalej sú vstupy spojené a predané lineárnej ReLU vrstve. Táto reprezentácia stavu je predaná lineárnym vrstvám ktorých výstupy sú ID akcie a argumenty.
- FullyConv Agent - Tento návrh agenta je špecifický tým, že konvolučné vrstvy zachovávajú rozlíšenie počas celého priechodu neuorónovou sieťou až do priestorového výstupu, ktorý predstavuje parameter pre priestorové akcie. Konvolučné vrstvy sú 2, o 16 a 32 filtroch, veľkosť 5x5 a 3x3, krok je 1 a používa sa padding aby nedošlo k strate priestorovej informácie. Reprezentácia stavu je vytvorená spojením výstupov konvolučných vrstiev obrazovky a minimapy na dimenzii kanálov a skalárnych hod-

nôt. Pre výstupy, ktoré nepotrebujú priestorovú reprezentáciu, je tento výstup ešte prehnaný cez plne prepojenú ReLU vrstvu nasledovanú plne prepojenými linárnymi vrstvami. Priestorová informácia sa získa použitím konvolučnej vrstvy o veľkost 1x1, a jedným filtrom. Počet výstupov je teda rovný vstupnému rozlíšeniu obrazovky.

- FullyConv LSTM Agent - Implementácia vychádza z FullyConv agenta, jediný podstatný rozdiel je že za reprezentáciou stavu je pridaná LSTM vrstva.



Obr. 7.1: **Ukážka architektúry Atari a FullyConv agentov** zdroj:[23]

7.2 Implementačné prostredie

Samotná implementácia algoritmu a následné experimenty prebiehali na operačnom systéme Windows 10. Program je napísaný v jazyku Python, verzia 3.6.8. Vytvorenie neurónovej siete a jej tréning zabezpečuje knižnica tensorflow-gpu v. 1.12. Zvyšok konfigurácie je popísaný v readme.txt súbore priloženom k bakalárskej práci.

V prostredí školských počítačov sa mi prostredie PySC2 nepodarilo spustiť pretože na danej verzii naisnštalovaného systému chýbala jedna z potrebných systémových knižníc a teda som implementáciu a testy vykonával na vlastných počítačoch.

Hardvérová konfigurácie prvého stroja je šesťjadrový procesor Intel i7, 16 GB pamäte RAM, a grafickú kartu Nvidia GeForce 1060 6GB s podporou technológie CUDA.

Druhý počítač má štvorjadrový procesor Intel i7, 16 GB RAM, a grafický adaptér Nvidia Geforce 860M 2GB.

7.3 Vlastná implementácia

Samotná architektúra programu je inšpirovaná implementáciou A2C z voľne dostupného repozitára OpenAI Baslines[13]. Programová časť práce pozostáva z týchto súborov:

- readme.txt - Popis softvérovej konfigurácie potrebnej na spustenie programu a taktiež návod na jeho obsluhu.
- network.py - Súbor s implementáciou neurónovej siete.

- agent.py - V tomto súbore sa nachádza tensorflow graf, je zodpovedný za obsluhu a trénovanie neurónovej siete.
- runner.py - Centrálna časť programu ktorá ho spája dohromady. Runner komunikuje ako s prostredím PySC2, tak aj s agentom. Je zodpovedný za vytváranie trénovacích batchov.
- run.py - Vstupný bod programu, po spustení spracuje prepínače, vytvorí vektor n paralelných prostredí PySC2 a spustí sa trénovanie
- env.py - Súbor ktorý umožňuje vytvorenie vektora PySC2 proostredí komunikujúcich medzi sebou pomocou rút, kód prebratý od [7]
- plot.py - Skript na vygenerovanie grafu zo súboru results.txt po ukončení trénovania.

Implementácia neurónovej siete je inširovaná Deepmind FullyConv Agentom. Podobne sú teda na vstupe za sebou dve konvolučné vrstvy s ReLU aktiváciou o veľkosti 5x5 a 3x3, 8 a 16 filtroch, pre vstup z obrazovky a minimapy. Tieto vrstvy sú následne vzájomne spojené aj so skalárnymi hodnotami do reprezentácie stavu. Odtiaľ sa sieť vetví, prvá vetva vedie do konvolučnej vrstvy s jedným filtom o veľkosti 1x1, ktorý poskytne priestorové argumenty. Druhá vetva smeruje do 100 neurónov počítajúcej plne prepojenej ReLU vrstvy. Za ňou je jeden numerický výstup, ktorý reprezentuje hodnotu stavu(Critic) a n výstupov, jeden pre každé id akcie a výstupy pre zvyšné argumenty. Okrem výstupu pre hodnotu sú všetky ostatné výstupy prevedené pomocou funkcie SoftMax na pravdepodobnostné rozloženie.

Run.py po spustení vytvorí najprv pomocou env.py vektor prostredí a nainicializuje triedy Agent a Runner. Následne iteratívne volá procedúru run() triedy Runner, kde jedno zavolanie predstavuje odohratie jedného batchu a na jeho konci jeden trénovací krok. V rámci procedúry run() sa najprv spracujú vstupné dáta od prostredia PySC2, stav prostredia sa predá v parametroch procedúre step() z triedy Agent, ten týmito informáciami nakŕmi neurónovú sieť a z nej dostane akcie a ich argumenty a tieto vráti triede Runner.

Na základe týchto akcií sa vykoná krok v prostredí PySC2, to nám vráti za daný krok odmenu a nový stav prostredia. Ďaľej iterujeme znovu od začiatku až na koniec batchu.

Všetky relevantné údaje sa ukladajú pre trénovací krok na konci, po skončení ktorého sa kontrola vracia späť do run.py.

Kapitola 8

Experimenálna časť

Pri trénovaní som sa snažil použiť podobné parametre systému aké boli použité v [23]. Asi najvýraznejší rozdiel je v použitom rozlíšení. Deepmind použil 64, v prípade tejto práce však z výkonnostných dôvodov bolo vo väčšine experimentov rozlíšenie obrazovky nastavené na hodnotu 16. Testované boli aj varianty s hodnotou 32, konvergovali však kôli väčšej komplexite pomalšie a neboli zaznamenané rozdiely v schopnosti nájsť lepšiu stratégiu. Pomer počtu krokov hry na krok agenta je nastavený na 8, a počet krokov v rámci trénovacieho batchu je 16. Použitý optimalizátor bol RMSProp a discount factor 0.99.

Deepmind každý experiment pustil 100 krát s náhodne zvolenými hyperparametrami. V našom prípade však taký experiment nie je z časových dôvodov reálny, teda optimálne trénovacie parametre som experimentálne hľadal iba na najjednoduchšej úlohe FollowBeacon a použil ich aj na trénovanie ostatných úloh. Experiment bol pustený 20 krát a parametre boli vyberané z rozmedzí:

- learning rate: 10^{-3} až 10^{-5}
- epsilon: 10^{-5} až 10^{-10}
- decay: 0.9 až 0.99

Experiment bol vykonaný nasledovným spôsobom. Najprv boli zvolené stredné hodnoty z daných rozmedzí a sledoval som ako efektívne algoritmus konverguje. Následne som vždy zmenil jeden parameter a sledoval účinok na rýchlosť konvergencie.

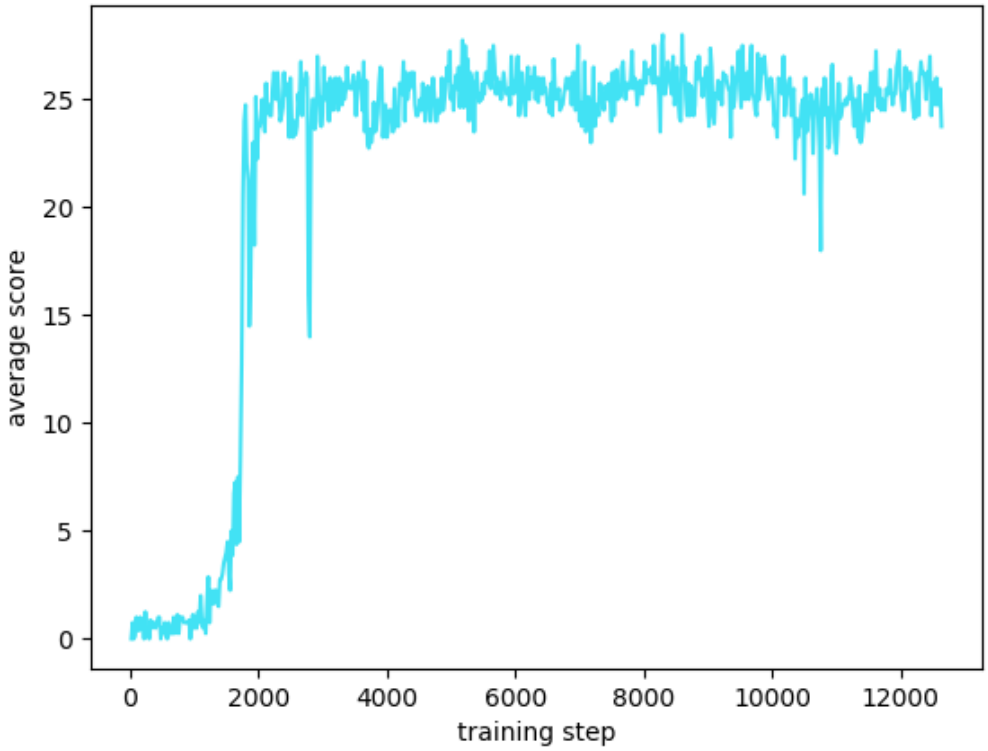
Najrýchlejšie dosiahol globálne maximum v tejto úlohe agent s parametrami $5 * 10^{-4}$ pre learning rate, 10^{-5} pre epsilon a 0.99 decay za približne 90 epizód, čo je približne 1800 trénovacích krokov.

Exerimenty prebiehali na jednotlivých úlohách vyonávané paralelne na dvoch počítačoch po dobu jedného týždňa. okrem prvého experimentu bol každý pustený 2 krát po dobu približne 24 hodín.

Čo sa týka paralelizácie, na oboch počítačoch boli pri trénovaní pustené paralelne 4 inštancie Starcraftu 2, v prípade oboch počítačov bol bottleneck ktorý neumožnil pustiť viac inštancií naraz nedostatok ako pamäte v grafickej karte tak aj pamäť RAM.

8.1 Experiment MoveToBeacon

Najjednoduchšia úloha, agent má k dispozícii jednu jednotku a s ňou musí dôjsť na vyznačené miesto. Za úspech dostáva jeden bod a vyznačené miesto sa objaví na inom mieste.

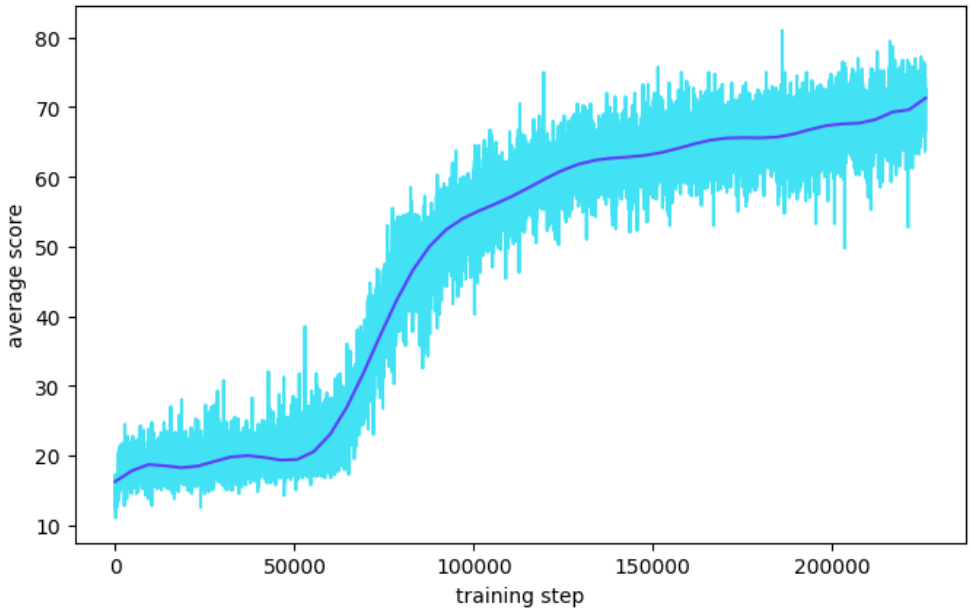


Obr. 8.1: **Priebeh tréningu na mape MoveToBeacon**

S touto úlohou algoritmus problém nemal, pomerne rýchlo zvládol skonvergovať k optimálnemu riešeniu a cielene okamžite chodil na vyznačené miesta hneď po ich objavení.

8.2 Experiment CollectMineralShards

V tejto úlohe má agent k dispozícii dve jednotky a na mape sú náhodne rozostavené minerály, ktoré má za úlohu pozbierať. Za každý zdvihnutý kryštál získava odmenu.

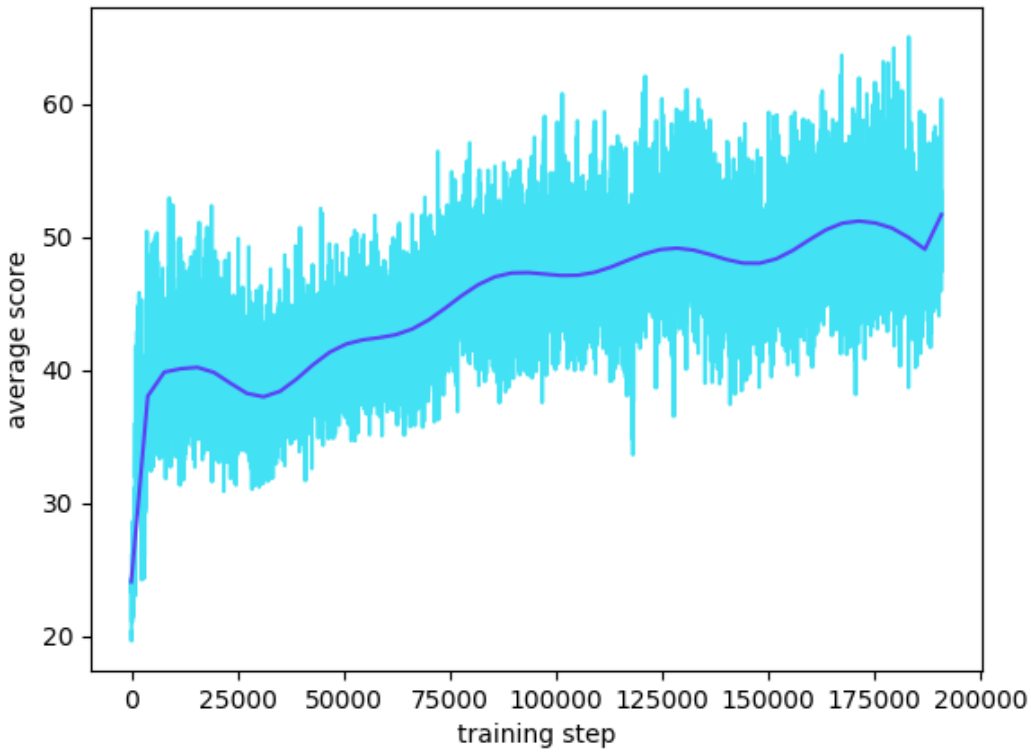


Obr. 8.2: **Priebeh tréningu na mape CollectMineralShards**

Priebeh tohto tréningu je už pomalší ako pri prvej úlohe. Agentovi sa podarilo dosiahnuť lokálne maximum. Kryštály zbieral cielene, problém však bol že sa pohyboval s jednotkami spolu a optimálna stratégia je jednotky rozdeliť a s každou zbierať kryštály samostatne. Bohužiaľ sa túto pokročilú taktiku nepodarilo objaviť.

8.3 Experiment DefeatZerglingsAndBanelings

Týmto prechádzame na komplexnejšie úlohy. Konkrétne táto je zameraná na efektívne ovládanie boja jednotiek. Na začiatku minihry má agent k dispozíi niekoľko jednotiek ktoré strieľajú na diaľku a na mape sú aj nepriateľské jednotky ktoré je potrebné poraziť. Niektoré z nepriateľských jednotiek sú špecifické tým, že vybuchujú v určitej oblasti a teda sa ich agent musí snažiť rozdeliť do niekoľkých skupín aby o všetky naraz neprišiel. Ďalšia dôležitá stratégia ktorú sa agent musí naučiť pre úspešné zvládnutie úlohy je sústredenie streľby prioritne vybuchujúce jednotky, pretože tie predstavujú najväčšie riziko. Za každú porazenú nepriateľskú jednotku získava agent pozitívnu odmenu, za stratenie jednotky v boji však dostáva odmenu negatívnu.

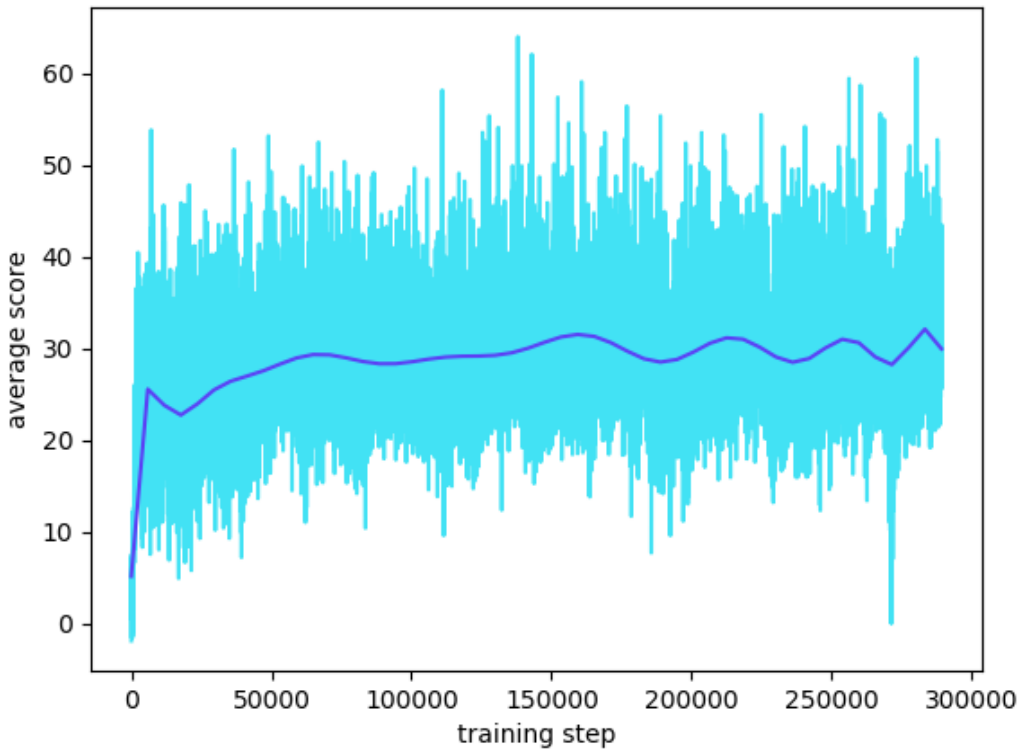


Obr. 8.3: **Priebeh tréningu na mape DefeatZerglingsAndBanelings**

Agent sa v tejto úlohe zvládol naučiť jednotky efektívne rozdeliť, avšak sa nezvládol vo vymedzenom časovom limite naučiť koncentráciu streľby.

8.4 Experiment DefeatRoaches

Ďalší bojový scenár. Agent má znovu k dispozícii zopár strieľajúcich jednotiek a nepriateľ má tentokrát tiež strieľajúce jednotky, ktoré sú o niečo silnejšie. Ak teda má byť agent v tejto úlohe úspešný, potrebuje zvládnuť znovu dve stratégie. Jedna je spoločná s predošlou minihrou, to je koncentrácia paľby na jednu konkrétnu jednotku a teda jej rýchle vyradenie z hry. Druhá je taktická pozícia, rozostavenie jednotiek. Na začiatku hry sú nepriateľské jednotky postavené v rade, teda je optimálne na nich zaútočiť zboku na konci rady aby všetky naše jednotky mohli útočiť, ale iba časť nepriateľovych.

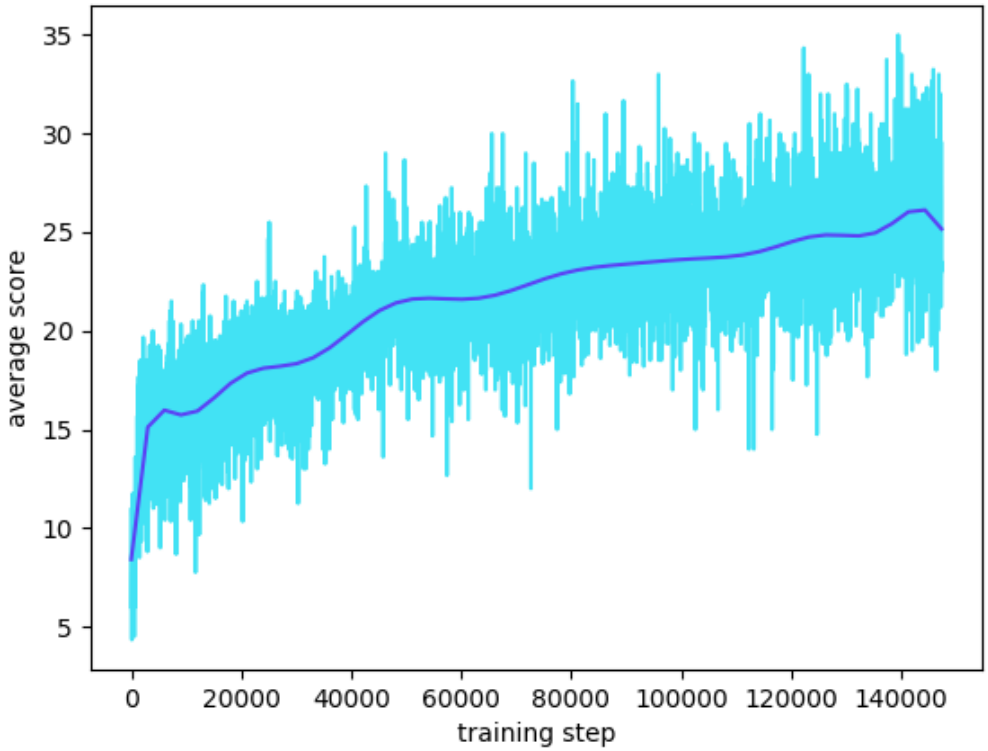


Obr. 8.4: **Priebeh tréningu na mape DefeatRoaches**

Agent sa zvládol naučiť zaútočiť zo strany aj keď mu táto taktika niekedy nevyšla. Avšak koncentráciu paľby sa mu znova nepodarilo naučiť.

8.5 Experiment FindAndDefeatZerglings

Tento scenár je špecifický tým, že sa neodohráva len na mape veľkosti obrazovky, ale hracia plocha je väčšia. Teda je nutné, aby sa agent naučil hýbať s kamerou a taktiež naučil nepriateľské jednotky hľadať, keďže jednotky vidia prostredie len do určitej vzdialenosti okolo seba. Úloha je nájsť a zničiť všetky nepriateľské jednotky schovávajúce sa na mape.



Obr. 8.5: **Priebeh najlepšieho tréningu na mape FindAndDefeatZerglings**

Agent sa naučil navigovať po mape a väčšinou ísť tam, kde ešte nebol. Avšak niekedy na už objavenom mieste mapy zabudol zničiť jednu jednotku a potom ju dlho náhodne hľadal čím prichádzal o skóre.

8.6 Porovnanie výsledkov

Tabuľka 8.1: **Porovnanie výsledkov s FullyConv agentom**

Názov minihry	**Vlastná implementácia A2C**	**FullyConv**
MoveToBeacon	26	26
CollectMineralShards	70	103
DefeatZerglingsAndBanelings	51	62
DefeatRoaches	29	100
FindAndDefeatZerglings	32	45

Z tabuľky je vidieť že na jednoduchších úlohách výsledky niesu veľmi ďaleko od skóre ktoré v svojich testoch dosiahol Deepmind. Z grafov priebehu tréningu je vidieť že agent má v niektorých úlohách potenciál sa ešte zlepšovať, len bol tréning z časových dôvodov ukončený. Pomerne neuspokojivý výsledok je akurát z mapy DefeatRoaches, ktorú FullyConv agent zvládol podstatne lepšie. Dôvody prečo je výsledok taký aký je môžu byť rôzne, predčasné ukončenie tréningu, na konkrétnu úlohu nevhodne nastavené hyperparametre trénovania, alebo potencionálne môže dôvod byť zníženie rozlíšenia vstupu, alebo o niečo menej komplexná architektúra neurónovej siete.

8.7 Ďalší možný vývoj

Čo sa týka ďalšieho vývoja tejto práce, cesta by určite viedla cez zdokonaľovanie architektúry neurónovej siete, napríklad ako to spravil deepmind pridaním LSTM vrstvy. Taktiež by mohla pomôcť ďalšia optimalizácia trénovacích parametrov.

Jedna z možných variant ako agenta spraviť efektívnejším je tiež aj použitie iného, potencionálne efektívnejšieho algoritmu, napríklad PPO alebo SAC.

Zaujímavé riešenie by mohlo byť aj zakomponovať do riešenia učenie zo záznamov hry kvalitných hráčov.

Kapitola 9

Záver

V práci bol najprv prebratý význam umelej inteligencie, strojového učenia a ich potenciónálny prínos. Následne bolo skúmané ako strojové učenie vo svojej podstate funguje a aké ciele dnes sleduje. Bol prebratý aj význam hier v tejto problematike a ich história spojená s výskumom umelej inteligencie. Následne sme sa dostali ku strategickej hre Starcraft 2, ktorá v tomto momente predstavuje horúci objekt záujmu mnohých výskumníkov v oblasti strojového učenia.

V praktickej časti tejto práce bol implementovaný algoritmus A2C na hranie Starcraftu 2 prostredníctvom PySC2 rozhrania na strojové učenie. Potom boli v tomto prostredí vykonané experimenty, tie boli boli následne zhodnotené a prirovnané k referenčným experimentom od spoločnosti Deepmind. Na záver boli prebraté možnosti ďalšieho vývoja.

Literatúra

[1] *Artificial intelligence in video games*. [Online; navštívené 12.05.2019].
URL
https://en.wikipedia.org/wiki/Artificial_intelligence_in_video_games

[2] *Convolutional neural network*. [Online; navštívené 11.05.2019].
URL https://en.wikipedia.org/wiki/Convolutional_neural_network

[3] *Deepmind - AlphaGo*. [Online; navštívené 12.05.2019].
URL https://deepmind.com/research/alphago/

[4] *Deepmind blogpost*. [Online; navštívené 5.01.2019].
URL https://deepmind.com/blog/deepmind-and-blizzard-open-starcraft-ii-ai-research-environment/

[5] *Descending into ML: Training and Loss*. [Online; navštívené 12.05.2019].
URL https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss

[6] *Markov decision process*. [Online; navštívené 4.05.2019].
URL https://en.wikipedia.org/wiki/Markov_decision_process

[7] *Pekka Aalto github repozitár*. [Online; navštívené 12.05.2019].
URL https://github.com/deepmind/pysc2

[8] *Timeline of machine learning*. [Online; navštívené 4.05.2019].
URL https://en.wikipedia.org/wiki/Timeline_of_machine_learning

[9] Atienza, R.: *Advanced Deep Learning with Keras*. Packt Publishing, 2018, ISBN 9781788629416.

[10] Baum, S.: *A Survey of Artificial General Intelligence Projects for Ethics, Risk, and Policy*. [Online; navštívené 2.05.2019].
URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3070741

[11] Bellemare, M. G.; Naddaf, Y.; Veness, J.; aj.: The Arcade Learning Environment: An Evaluation Platform for General Agents. *CoRR*, ročník abs/1207.4708, 2012, 1207.4708.
URL http://arxiv.org/abs/1207.4708

[12] Dertat, A.: *Applied Deep Learning - Part 4: Convolutional Neural Networks*. [Online; navštívené 11.05.2019].
URL https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

[13] Dhariwal, P.; Hesse, C.; Klimov, O.; aj.: OpenAI Baselines. https://github.com/openai/baselines, 2017.

[14] Kızrak, A.: *Comparison of Activation Functions for Deep Neural Networks*. [Online; navštívené 11.05.2019].
URL https://towardsdatascience.com/comparison-of-activation-functions-for-deep-neural-networks-706ac4284c8a

[15] Levine, S.: *Policy Gradients* . [Online; navštívené 4.05.2019].
URL http://rail.eecs.berkeley.edu/deeprlcourse-fa17/f17docs/lecture_4_policy_gradient.pdf

[16] Levine, S.: *Reinforcement Learning w/ Keras + OpenAI: Actor-Critic Models*. [Online; navštívené 6.05.2019].
URL https://towardsdatascience.com/reinforcement-learning-w-keras-openai-actor-critic-models-f084612cfd69

[17] McCorduck, P.: *Machines Who Think*. A K Peters, 2004, ISBN 978-1-56881-205-2.

[18] Mnih, V.; Badia, A. P.; Mirza, M.; aj.: Asynchronous Methods for Deep Reinforcement Learning. *CoRR*, ročník abs/1602.01783, 2016, 1602.01783.
URL http://arxiv.org/abs/1602.01783

[19] Salloum, Z.: *Policy Based Reinforcement Learning, the Easy Way*. [Online; navštívené 4.05.2019].
URL https://towardsdatascience.com/policy-based-reinforcement-learning-the-easy-way-8de9a3356083

[20] Samuel, A.: Some studies in machine learning using the game of checkers. 1969.

[21] Sutton, R. S.: *Reinforcement Learning: An Introduction*. MIT Press, Second edition, 2018, ISBN 9780262039246.

[22] Vinyals, O.; Babuschkin, I.; Chung, J.; aj.: AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/, 2019.

[23] Vinyals, O.; Ewalds, T.; Bartunov, S.; aj.: StarCraft II: A New Challenge for Reinforcement Learning. *CoRR*, ročník abs/1708.04782, 2017, 1708.04782.
URL http://arxiv.org/abs/1708.04782

[24] Yoon, C.: *Understanding Actor Critic Methods and A2C*. [Online; navštívené 4.05.2019].
URL https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f

Príloha A

Obsah pamäťového média

- readme.txt - Popis softvérovej konfigurácie potrebnej na spustenie programu a taktiež návod na jeho obsluhu.
- network.py - Súbor s implementáciou neurónovej siete.
- agent.py - V tomto súbore sa nachádza tensorflow graf, je zodpovedný za obsluhu a trénovanie neurónovej siete.
- runner.py - Centrálna časť programu ktorá ho spája dohromady. Runner komunikuje ako s prostredím PySC2, tak aj s agentom. Je zodpovedný za vytváranie trénovacích batchov.
- run.py - Vstupný bod programu, po spustení spracuje prepínače, vytvorí vektor n paralelných prostredí PySC2 a spustí sa trénovanie
- env.py - Súbor ktorý umožňuje vytvorenie vektora PySC2 proostredí komunikujúcich medzi sebou pomocou rút, kód prebratý od [7]
- plot.py - Skript na vygenerovanie grafu zo súboru results.txt po ukončení trénovania.
- bc - adresár obsahujúci zdrojové súbory bakalárskej práce