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

HRANÍ HER POMOCÍ NEURONOVÝCH SÍTÍ

PLAYING GAMES USING NEURAL NETWORKS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PETR BUCHAL**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MICHAL HRADIŠ, Ph.D.**
SUPERVISOR

BRNO 2018

Zadání bakalářské práce/20907/2017/xbucha02

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Buchal Petr**
Obor: Informační technologie
Téma: **Hraní her pomocí neuronových sítí**
Playing Games Using Neural Networks
Kategorie: Zpracování obrazu

Pokyny:

1. Prostudujte základy neuronových sítí a posilovaného učení.
2. Vytvořte si přehled o současných metodách využívajících neuronové sítě a posilované učení.
3. Vyberte konkrétní metodu aplikovatelnou na určitý problém posilovaného učení.
4. Obstarejte si databázi vhodnou pro experimenty.
5. Implementujte navrženou metodu a proveďte experimenty nad datovou sadou.
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 bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Hradiš Michal, Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
L.S. 612 66 Brno, Božetěchova 2

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Cílem této práce je naučit neuronovou síť pohybu v prostředích s klasickou kontrolou řízení, hraní tahové hry 2048 a několika Atari her. Jedná se o oblast zpětnovazebního učení. Jako zpětnovazební algoritmus využívající neuronové sítě jsem použil Hluboké Q-učení. Ten jsem pro zvýšení efektivity učení obohatil o několik vylepšení. Mezi vylepšení patří přidání cílové sítě, DDQN, duální architektura neuronové sítě a prioritní vzpomínková paměť. Experimenty s klasickou kontrolou řízení zjistily, že nejvíce zvedá efektivitu učení přidání cílové sítě. V prostředích her dosáhlo Hluboké Q-učení několikanásobně lepších výsledků než náhodný hráč. Výsledky a jejich analýza mohou být využity ke vhledu do problematiky zpětnovazebních algoritmů využívajících neuronové sítě a zdokonalení použitých postupů.

Abstract

The aim of this bachelor thesis is to teach a neural network solving classic control theory problems and playing the turn-based game 2048 and several Atari games. It is about the process of the reinforcement learning. I used the Deep Q-learning reinforcement learning algorithm which uses a neural networks. In order to improve a learning efficiency, I enriched the algorithm with several improvements. The enhancements include the addition of a target network, DDQN, dueling neural network architecture and priority experience replay memory. The experiments with classic control theory problems found out that the learning efficiency is most increased by adding a target network. In the game environments, the Deep Q-learning has achieved several times better results than a random player. The results and their analysis can be used for an insight to reinforcement learning algorithms using neural networks and to improve the used techniques.

Klíčová slova

Strojové učení, Zpětnovazební učení, Neuronové sítě, Q-učení, Hluboké Q-učení

Keywords

Machine learning, Reinforcement learning, Neural networks, Q-learning, Deep Q-learning

Citace

BUCHAL, Petr. *Hraní her pomocí neuronových sítí.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Hradiš, Ph.D.

Hraní her pomocí neuronových sítí

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Hradiše, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Buchal
16. května 2018

Poděkování

Chtěl bych poděkovat vedoucímu mé bakalářské práce Ing. Michalu Hradišovi, Ph.D. za odborné vedení, za rady a za pomoc při zpracování této práce.

Tato práce vznikla za podpory projektů CERIT Scientific Cloud (LM2015085) a CESNET (LM2015042) financovaných z programu MŠMT Projekty velkých infrastruktur pro VaVaI.

Obsah

Kapitola 1

Úvod

Člověk se učí novým dovednostem celý život, jak chodit, jak mluvit, jak počítat atd. Učí se na základě podnětů, které dostává při provádění jednotlivých akcí. Když malé dítě nešikovně došlápne na zem a spadne, zjistí, že takovým způsobem se nedostane ke své oblíbené hračce a příště došlápne jinak. Učí se na základě zpětné vazby. Tento přístup se uplatňuje úplně stejně, když se člověk učí hrát nějakou hru. Jednotlivými tahy prozkoumává své možnosti a zjišťuje, jaké akce mu přinesou nejlepší možný výsledek. Tímto přístupem se rovněž zabývá jedna oblast strojového učení tzv. zpětnovazební učení.

Má práce se zabývá zkoumáním zpětnovazebního algoritmu hlubokého Q-učení (DQN) [17] a několika jeho vylepšeními. Cílem DQN je vytvořit agenta, který se v libovolném prostředí bude schopen naučit inteligentnímu chování podle reakcí okolí ve formě odměn. Jde tedy o formu umělé inteligence. Důležitým prvkem DQN je poté neuronová síť, která je „agentovým mozkem“. Na vstup jí přichází popis prostředí a její výstup určuje, kterou akci je nejlepší provést.

Historie má několik významných milníků, které jsou spjaty s uvedením agentů, kteří dokázali plnit konkrétní úlohu v konkrétním prostředí. Pravděpodobně nejznámějším takovým milníkem je vítězství počítače Deep Blue nad Garri Kasparovem v šachové partii v roce 1997 [25]. Tehdy se jednalo agenta specializovaného na hraní šachů, tedy nic jiného než hrát šachy nedokázal. Navíc se neučil pomocí zpětné vazby, ale pouze prozkoumával nejlepší možné budoucí tahy. Když v roce 2013 představila firma DeepMind algoritmus DQN, byla to do jisté míry revoluce. Algoritmus se totiž dokázal úspěšně naučit hrát různé Atari hry, aniž by se specializoval na každou hru zvlášť a to vše pouze se snímky obrazovky jako vstupem.

Trénování agenta algoritmem DQN má dva problémy, které spolu úzce souvisí. Prvním problémem je relativně pomalé učení. To například u Atari her trvá i několik dní. Druhým problémem je poté obtížné nastavení hyperparametrů algoritmu, kterých je nemalé množství a které kriticky ovlivňují úspěšnost a dobu učení. K lepšímu pochopení chování algoritmu v závislosti na hodnotě hyperparametrů práce nejprve zkoumá chování DQN v prostředích s klasickou kontrolou řízení. V těchto experimentech se pracuje s neuronovou sítí bez vrstev na zpracování obrazu, což urychluje učení a umožňuje lépe statisticky zkoumat algoritmus. V další části práce je algoritmus aplikován nejdříve na hru 2048, která mu zprostředkovává stav skrze matici hodnot hracího pole, a následně na několik Atari her, které neuronové síti poskytují svojí RAM paměť a svůj obraz.

Kapitola 2

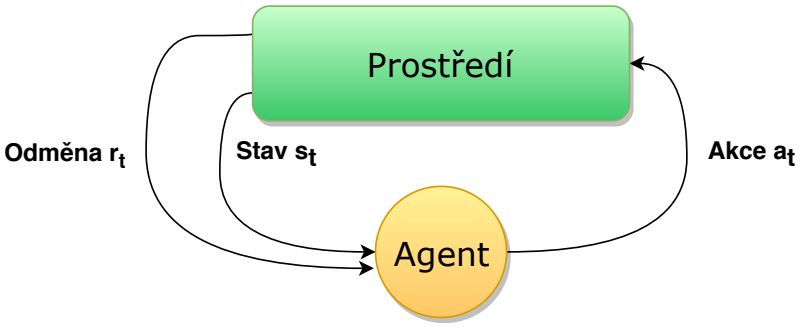
Zpětnovazební učení

Zpětnovazební učení je oblast strojového učení, zabývající se chováním agentů v různých prostředích. Cílem zpětnovazebního učení je, co nejlépe vytrénovat agenta k maximalizaci kumulativní odměny za akce, které v prostředí provádí [30] (obrázek 2.1). Prostředí v jakých se agenti pohybují bývají většinou formulovány jako Markovův rozhodovací proces. Zjednodušeně se jedná o diskrétní stochastický proces, ve kterém se agent nachází ve stavu s a následným provedením jedné z možných akcí a se dostane do stavu s' a obdrží odměnu r [28], dále viz kapitola 2.1.

Zpětnovazební učení se dělí na několik druhů podle různých kritérií. Základním dělením je dělení na pasivní a aktivní učení [1]. Při pasivním učení agent pracuje s pevnou strategií π a učí se užitek z různých stavů a akcí. Cílem je poté ohodnotit kvalitu dané strategie na základě očekávaného užitku. Při aktivním učení agent prozkoumává prostředí a sám si stanovuje strategii pomocí dosaženého užitku z různých akcí v různých stavech [31]. Užitek se stanovuje pomocí Bellmanovy rovnice, její varianta pro Q-učení se nachází v rovnici 2.4.

Dále můžeme rozdělit zpětnovazební učení na učení, které je založeno na modelu a na to které na něm založené není. Při učení, které je založeno na modelu, se nejprve vytváří model Markovovského rozhodovacího procesu. To znamená vytvoření funkce přechodu stavu $P_a(s, s')$ a oceňovací funkce $R_a(s, s')$, viz kapitola 2.1. Následně se vytvořený model použije k hledání optimální strategie. Zpětnovazební učení, které není založeno na modelu hledá optimální strategii bez použití modelu [4].

Dále je možné rozdělit zpětnovazební učení podle druhu strategie. Agenta, který získává podklady k učení na základě akcí, které provedl podle strategie, kterou se učí, nazýváme agentem se zapnutou strategií (on-policy). Oproti tomu agenta, který při učení používá



Obrázek 2.1: Vizualizace zpětnovazebního učení.

alespoň z části jinou než vlastní strategii, nazýváme agentem s vypnutou strategií (off-policy). Strategií takového agenta může být například chamtivá explorace nebo ε-chamtivá explorace [8].

Základní algoritmy zpětnovazebního učení lze rozdělit na ty které hledají hodnotovou funkci a na ty, které hledají strategii pomocí gradientů [22]. Algoritmy, které hledají hodnotovou funkci, poskytují na základě stavu a konkrétní akce předpokládaný užitek z jejího provedení. Následně se provede akce s největším předpokládaným užitkem. Hodnotová funkce vrací právě předpokládaný užitek z provedení akce, kterému se jinak také říká diskontovaná budoucí odměna. Ta hraje důležitou roli v problému „Credit assigment problem“, který je rozebrán v následujícím odstavci. Algoritmy, které hledají strategii pomocí gradientů, přiřazují každé akci v daném stavu určitou pravděpodobnost provedení. Agent poté na vstup dostane stav a na základě pravděpodobnostního ohodnocení jednotlivých akcí pro daný stav, vybere tu s nejlepší pravděpodobností. Pokud byla akce v daném stavu přínosem, agent zvýší její pravděpodobnost pro příště a naopak.

Credit assigment problem. Agent provede vynikající akci, ale její důsledek se projeví se zpožděním. To má za následek, že agent nepozná konkrétní akci, která k odměně vedla. Tento problém se zpětnovazební učení snaží odstranit pomocí diskontované budoucí odměny [15]. V markovovském rozhodovacím procesu lze celková odměna za působení v prostředí vyjádřit jako

$$R = r_1 + r_2 + r_3 + ... + r_n. \tag{2.1}$$

Tedy celková odměna je rovna součtu odměn za každý krok. Budoucí odměnu lze poté vyjádřit rovnicí

$$R_t = r_t + r_{t+1} + r_{t+2} + ... + r_{t+n}. \tag{2.2}$$

Markovovský rozhodovací proces je ale stochastický, což znamená, že provedení stejných akcí nemusí vést nutně ke stejnému výsledku. Skutečnost je taková, že čím více kroků agent provede, tím více se budou stavy, kterých může dosáhnout lišit. Kvůli této skutečnosti se zavádí faktor stárnutí γ. Ten určuje důležitost, jakou přikládá agent budoucím stavům, při čemž se jeho hodnota pohybuje v intervalu $0 \leq \gamma \leq 1$. Pomocí faktoru stárnutí se poté určuje diskontovaná budoucí odměna

$$R_t = r_t + _{t+1} + \gamma^2 r_{t+2} + ... + \gamma^{n-t} r_{t+n} = r_t + _{t+1} \,. \tag{2.3}$$

Ta stanovuje, že s každým budoucím krokem narůstá mocnina faktoru stárnutí a tedy se zmenšuje hodnota očekávané odměny. Pokud bude faktor stárnutí nastaven na hodnotu blízkou nule, agent se bude učit výhradně z okamžité odměny. Nejčastěji se používá hodnota kolem 0.9, příkladem DeepMind použil v DQN [17] hodnotu 0.99. V případě deterministického prostředí by poté bylo záhodné nastavit γ na hodnotu 1, protože odměna za provedení stejných akcí bude vždy stejná.

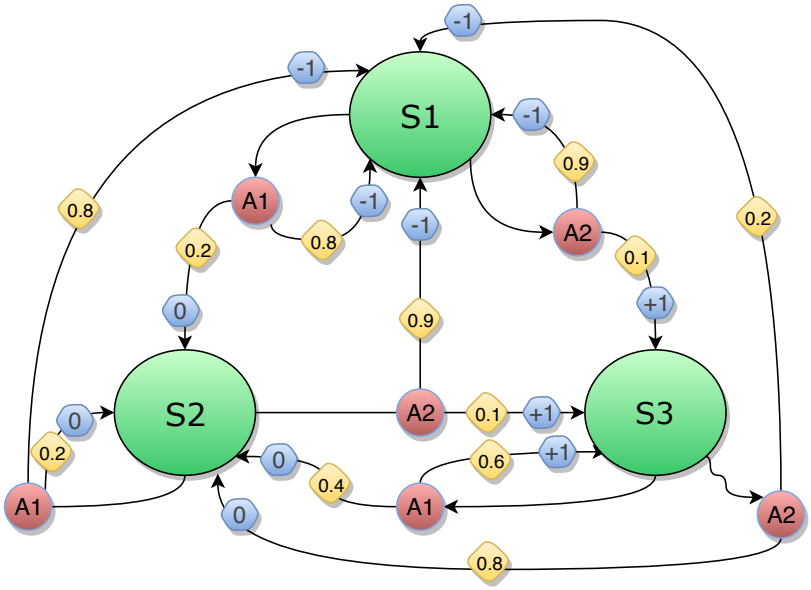
2.1 Markovův rozhodovací proces

Markovův rozhodovací proces předpokládá existenci agenta v prostředí, ve kterém má agent určitý stav s. Z takového stavu může provést dostupnou akci a podle strategie π a přejít do náhodného nového stavu s'. Za přechod dostane agent od prostředí určitou odměnou r, která může být jak kladná, tak záporná. Pravděpodobnost, že jako nový stav bude vybrán stav s', je určena funkcí přechodu stavu $P_a(s, s')$ (obrázek 2.2). Ta stanovuje, že následující

stav s' závisí pouze na současném stavu s a vybrané akci a, nikoliv na minulosti. Stav s je poté opět závislý pouze na předchozím stavu a akci atd. [15]. O přechodu mezi stavy můžeme říct, že má Markovovskou vlastnost, ta stanovuje, že pravděpodobnosti přechodu do nového stavu jsou závislé pouze na stavu současném, nikoliv na stavech předchozích [28].

Markovův rozhodovací proces je uspořádaná pětice $(S, A, P_a(s, s'), R_a(s, s'), \gamma)$, kde S je konečná množina stavů, A je konečná množina akcí, $P_a(s, s')$ neboli $Pr(s_{t+1} = s'|s_t = s, a_t = a)$ je funkce přechodu stavu, $R_a(s, s')$ je odměna získaná přechodem ze stavu s do stavu s' akcí a a γ je faktor stárnutí [28].

Funkce přechodu stavu určuje s jakou pravděpodobností akce a ve stavu s v čase t povede ke stavu s' v čase $t + 1$. Faktor stárnutí určuje, jak velká váha je přikládána budoucím odměnám. Teorie Markovových rozhodovacích procesů nevyžaduje, aby S a A byly konečné množiny. Jako konečné se uvádí, protože většina základních algoritmů předpokládá, že konečné jsou [28].



Obrázek 2.2: Vizualizace Markovova rozhodovacího procesu. Zelené kruhy reprezentují stavy, červené kruhy reprezentují akce, žluté čtverce reprezentují pravděpodobnost přechodu a modré šestiúhelníky reprezentují odměnu za přechod.

2.2 Q-učení

Algoritmus DQN, kterým se tato práce zabývá, vychází z algoritmu Q-učení a řeší jeho největší neduh. Následující řádky se věnují Q-učení a intuici za vznikem DQN. Jedná se o algoritmus aktivního učení, který hledá hodnotovou funkci, nevyužívá model a je off-policy. Q-učení se snaží agenta naučit ideální strategii výběru akce pro každý stav, který může navštívit. Je založeno na Q-funkci $Q(s, a)$. Ta určuje kvalitu konkrétní akce v konkrétním stavu [29], tzv. Q-hodnotu.

Pro ukládání Q-hodnot se obvykle využívá tabulky, kde je záznam pro každou akci v každém stavu [11]. Na začátku učení se inicializuje Q-funkce $Q(s, a)$, tím se rozumí nastavení Q-hodnot v tabulce. Obvykle se všechny tyto hodnoty nastavují na nulu [11]. Po inicializaci Q-funkce se agent začne epizodicky pohybovat prostředím. V každém kroku pro-

Algoritmus 1 Pseudokód algoritmu Q-učení (převzato z [23]).

Libovolně inicializuj funkci $Q(s, a)$
repeat pro každou epizodu
  Inicializuj prostředí a pozoruj počáteční stav s
  repeat pro každý krok epizody
    Vyber akci a pro stav s podle strategie odvozené od Q
    Proveď akci a a pozoruj odměnu r a nový stav s'
    $Q(s, a) = Q(s, a) + \alpha[r + \gamma max_{a'} Q(s', a') - Q(s, a)]$
    $s = s'$
  until s je konečný stav
until učení je dokončeno

vede akci s největším ohodnocením a pozoruje odměnu, kterou za přechod dostal a nový stav, ve kterém se nachází. Následně se pomocí vzorce

$$Q(s, a) = Q(s, a) + \alpha[r + \gamma max_{a'} Q(s', a') - Q(s, a)] \tag{2.4}$$

odvozeného z Bellmanovy rovnice [26], aktualizují Q-hodnoty v tabulce. Konstanta α se nazývá učící konstanta, náleží do intervalu $0 \leq \alpha \leq 1$ a určuje jakou měrou se Q-funkce učí z nových stavů. Proměnná r je odměna získaná přechodem ze stavu s do stavu s' akcí a. Konstanta γ je faktor stárnutí, její hodnota náleží do intervalu $0 \leq \gamma \leq 1$ a určuje, jak velká váha je přikládána budoucím odměnám. Část vzorce $max_{a'} Q(s', a')$ určuje odhad očekávané odměny za nejkvalitnější akci v následujícím stavu.

Bellmanova rovnice předpokládá, že dlouhodobá očekávaná odměna je stejná jako součet okamžité odměny za akci v současném stavu a očekávané odměny za nejkvalitnější akci v následujícím stavu. Důležitost, jakou má očekávaná odměna v součtu s okamžitou, je dána faktorem stárnutí γ. Ten se pohybuje v intervalu od 0 do 1 a jeho správné nastavení je důležité pro stabilní učení. Pokud se $\gamma = 0$, agent se učí pouze z hodnot okamžitých odměn. Na druhou stranu, pokud by se od začátku učení blížila γ hodnotě 1, tak by se skrze nenaučenou Q-funkci propagovaly nepravdivé hodnoty očekávaných odměn a to by mohlo učení destabilizovat. Tento problém se řeší nastavením nižší hodnoty γ na začátku učení s jejím postupným zvětšováním [29]. Druhým důležitým parametrem v Bellmanově rovnici je α. Ta ovlivňuje jakou měrou se učí Q-funkce z nových stavů. V případě že by hodnota α byla 1, tak by Q-hodnoty byly tvořeny pouze nejaktuálnějšími stavy, oproti tomu pokud by se $\alpha = 0$, tak by Q-hodnoty byly stále stejné a agent by se neučil [29].

Explorace vs. exploitace. Na začátku trénování jsou všechny agentovy akce náhodné a agent provádí exploraci, tedy průzkum stavového prostoru. Po nějaké době začne agentova strategie konvergovat a začíná exploitace, tedy využívání nejlepší dostupné strategie. To znamená, že agent nadále neprozkoumává stavový prostor, i když se v něm může nacházet strategie, která by mu přinesla větší odměnu [15]. Tento jev se řeší například použitím ε-chamtivé explorace [19]. Pomocí této strategie agent s pravděpodobností ε provede náhodnou akci místo té, kterou by provedl na základě své znalosti. Na začátku trénování se hodnota ε zpravidla nastavuje na větší hodnotu, přičemž se poté v průběhu učení postupně snižuje. DeepMind použil v DQN jako počáteční hodnotu ε 1 a následně ji lineárně v průběhu 1 000 000 kroků snížil na hodnotu 0.1 [18]. S touto hodnotou se nadále pokračovalo po zbytek trénování agenta.

Intuice za vznikem DQN. Problém algoritmu Q-učení, ve kterém jsou Q-hodnoty ukládány v tabulce, je velikost potažmo konečnost tabulky. Piškvorky, které mají velikost hracího pole 3x3, mají 26 830 možných stavů [3]. Už tedy správa tabulky pro hraní piškvorek by byla velmi neefektivní. Poté u mírně složitějších prostředí by tabulka nefungovala vůbec. Q-hodnoty se ale místo ukládání v tabulce mohou aproximovat pomocí nějaké funkce a právě to DQN dělá pomocí neuronové sítě [17].

Kapitola 3

Algoritmus DQN a jeho vylepšení

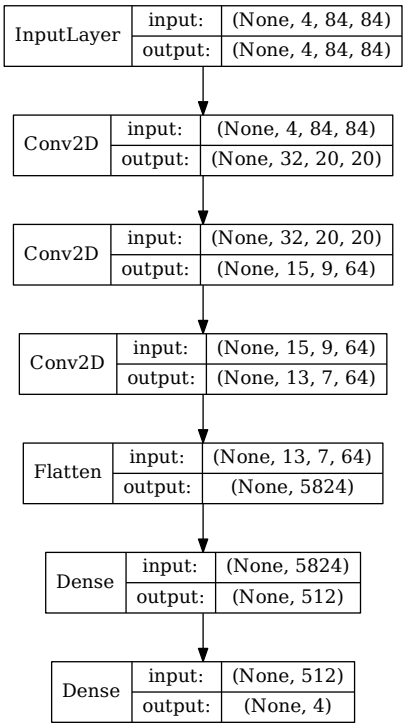
Největším nedostatkem Q-učení je problematické škálování tabulky pro popis stavů ve složitějších prostředích. V roce 2013 přišla firma DeepMind s řešením tohoto problému [17], když dokázala vytrénovat agenta v prostředí několika Atari her za použití neuronové sítě jako aproximátoru Q-hodnot. Tento algoritmus pojmenovali Hluboké Q-učení zkráceně DQN z anglického Deep Q-learning. Neuronová síť ovšem není jediným prvkem, díky kterému DQN funguje, přibyla například paměť se vzpomínkami, opakování akcí nebo ořezávání odměn. Pseudokód DQN se nachází v algoritmu 2. Později bylo DQN obohaceno o několik vylepšení, která několikanásobně zvýšila efektivitu učení [6] [20] [24].

Algoritmus 2 Pseudokód algoritmu DQN (převzato z [17]).

Inicializuj paměť D s kapacitou N
Inicializuj neuronovou síť Q s náhodnými váhami
for $epizoda = 1, M$ **do**
  Inicializuj prostředí a pozoruj počáteční stav s_t
  for $krok = 1, T$ **do**
    S pravděpodobností ε vyber náhodnou akci a_t
    jinak $a_t = argmax(Q(s_t))$
    Proveď akci a_t, pozoruj odměnu r_t a nový stav s_{t+1}
    Ulož vzpomínku (s_t, a_t, r_t, s_{t+1}) do paměti D
    Vezmi náhodný vzorek vzpomínek (s_i, a_i, r_i, s_{i+1}) z paměti D
    $$l_i = \begin{cases} r_i & \text{pro stav } s_i, \text{ který je koncový} \\ r_i + \gamma * \max(Q(s_{i+1})) & \text{pro stav } s_i, \text{ který není koncový} \end{cases}$$
    Za použití s_i jako trénovacích dat a l_i jako štítků trénuj Q
    $s_t = s_{t+1}$
  end for
end for

3.1 Neuronová síť

Základní architektura neuronové sítě algoritmu DQN se skládá ze tří konvolučních a dvou plně propojených vrstev [17] (obrázek 3.1). Architektura neuronové sítě DQN je díky konvolučním vrstvám podobná architektuře neuronové sítě na rozpoznání obrazu. Nachází se zde ovšem malý rozdíl, a to že zde chybí Pooling vrstvy. Ty zde nejsou, protože s jejich



Obrázek 3.1: Na obrázku se nachází schéma mé implementace neuronové sítě algoritmu DQN. Tento model je určen pro prostředí Breakout-v0 z toolkitu Open AI Gym. To na vstup neuronové sítě poskytuje 4 po sobě jdoucí snímky obrazovky. Ty mají velikost 84 x 84 pixelů a jsou šedotónové. Na výstup dává neuronová síť Q-hodnoty pro 4 akce, které lze v daném prostředí provést. Celkem síť obsahuje 3,046,052 parametrů.

použitím nedokáže síť určit, kde se objekty ve hře nachází a to je pro hraní her klíčové. Na vstup přijímá neuronová síť několik snímků obrazovky herního prostředí ve stupních šedi velikosti 84 x 84 pixelů. Jeden snímek totiž nedokáže poskytnout komplexní informace o prostředí, protože například rychlost z něj vyčíst nelze. Atari hry ale takový obraz neposkytují, většina jich dává na výstup obraz v RBG modelu o velikosti 210 x 160 pixelů a tak je potřeba jej upravit. Já obraz nejprve zmenším a poté z něj získám jas za pomocí vzorce

$$L = 0.299 * r + 0.587 * g + 0.114 * b, \tag{3.1}$$

kde r je červená složka, g je zelená složka a b je modrá složka.

Jako algoritmus učení podle gradientu zvolili autoři původního článku [17] RMSprop s učící konstantou 0.00025. Jako chybová funkce se obvykle u algoritmu učení podle gradientu používá střední čtvercová chyba (MSE)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(t_i - y_i)^2. \tag{3.2}$$

V rovnicích 3.2, 3.3 a 3.4 značí t_i cílovou Q-hodnotu pro i-tý vzorek a y_i sítí předpovězenou Q-hodnotu pro i-tý vzorek. Pro MSE funkci platí, že velikost chyby každého vzorku do značné míry ovlivňuje neznalost sítě vůči němu samému. Pokud se síť běžně nesetkává s jemu podobnými vzorky, jeho chyba může být nestandardně velká, což způsobí větší změny vah v síti při učení [10]. Tento jev je možné zmírnit použitím MSE v intervalu <-1,1> jakožto

rozdílu cílové a předpovězené Q-hodnoty a dále použitím chybové funkce střední absolutní chyby (MAE)

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|t_i - y_i| \tag{3.3}$$

mimo tento interval [10]. Kromě kombinace MSE a MAE se požívá ještě Huberova chybová funkce [10]

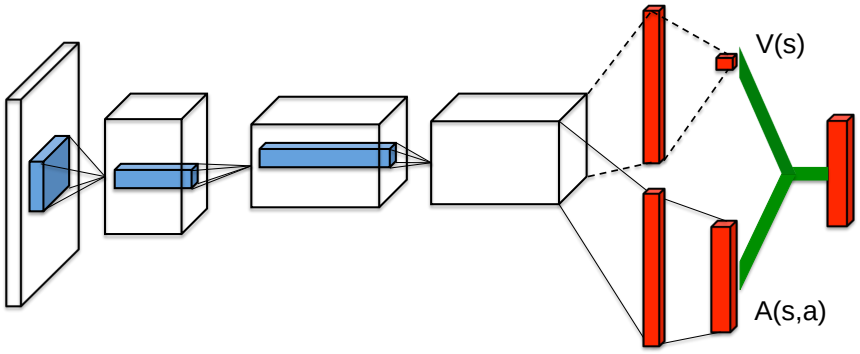
$$HUBER = \frac{1}{n}\sum_{i=1}^{n}\begin{cases}\frac{1}{2}(t_i \text{ - } y_i)^2 & \text{pro } |t_i - y_i| \leq \delta, \\ \delta(|t_i \text{ - } y_i| \text{ - } \frac{1}{2}\delta) & \text{jinak.}\end{cases} \tag{3.4}$$

Ta má podobný průběh jako kombinace MSE a MAE. Pro malé rozdíly t_i a y_i má kvadratický průběh, pro větší rozdíly má průběh lineární [27]. Konstanta δ určuje, od kterého bodu se kvadratický průběh funkce mění na lineární.

Duální architektura neuronové sítě [24]. Výstup originální architektury neuronové sítě algoritmu DQN určuje, jak dobré je provést konkrétní akci v konkrétním stavu. Tato skutečnost může být vyjádřena funkcí $Q(s, a)$. Cílem duální architektury neuronové sítě je zpřesnění výstupních hodnot této funkce skrze její rozložení na dvě jiné funkce [24]. Konkrétně na funkci hodnoty stavu $V(s)$, která určuje, jak dobré je být v daném stavu a na funkci výhody akce $A(s, a)$, která sděluje, jak kvalitní je daná akce oproti ostatním akcím. Vztah těchto funkcí je matematicky zapsán jako

$$Q(s, a) = V(s) + A(s, a). \tag{3.5}$$

Architektura duální neuronové sítě po zpracování dat konvolučními vrstvami dál samostatně počítá funkci hodnoty stavu $V(s)$ a funkci výhody akce $A(a)$ (obrázek 3.2). Následně jsou poté jejich výsledky zkombinovány na poslední vrstvě neuronové sítě, což vede ke zpřesnění odhadů kvality akcí.



Obrázek 3.2: Duální architektura neuronové sítě (převzato z [24]).

Ořezávání odměn. Různé hry poskytují agentovi různou odměnu. Například ve hře Beam Rider dostává agent za zničení nepřítele konstantně odměnu 44, ale ve hře Space Invaders dostává variabilní odměnu od 10 do 30 podle typu zničeného nepřítele. Tato variabilita negativně ovlivňuje stabilitu učení [21]. Tento jev se minimalizuje ořezáváním odměn tak, že všechny pozitivní odměny jsou +1 a všechny negativní jsou -1 [18].

Opakování akcí. DQN je schopno provést akci pro každý stav, tedy každý vyrenderovaný snímek obrazovky. Hry jsou ale uzpůsobené pro ovládání lidmi a ti nejsou schopni provádění akcí takovou rychlostí. DQN proto opakuje každou vybranou akci v následujících 3 stavech, tedy celkem 4krát [18].

3.2 Cílová síť a DDQN

V každém kroku DQN se mění váhy neuronové sítě. To způsobuje, že odhady kvality akcí uložených ve vzpomínkové paměti jsou nestabilní. Tato nestabilita může při učení neuronové sítě způsobit, že odhady kvality akcí začnou divergovat a naučená síť se destabilizuje. Ve snaze zmírnit toto riziko se do algoritmu přidává druhá tzv. cílová neuronová síť [12]. Ta se používá pro odhad kvality akcí během trénování. Její váhy se neaktualizují tak často a to má pozitivní efekt na stabilitu odhadů Q-hodnot.

Méně častá aktualizace vah v cílové síti není jediný způsob stabilizace učení skrze cílovou síť. Ještě se používá tzv. postupná aktualizace vah v cílové síti. Její princip je jednoduchý, v každém kroku se vezme zlomek vah z primární sítě se zlomkem vah z cílové sítě a jejich součet vytvoří nové váhy cílové sítě. Při trénování se aktuální Q-hodnota provedené akce získá jako

$$Q(s,a) = r + \gamma max_{a'} T(s',a'), \tag{3.6}$$

kde $T(s',a')$ je odhad Q-hodnoty pro konkrétní akci v následujícím stavu poskytnutý cílovou sítí.

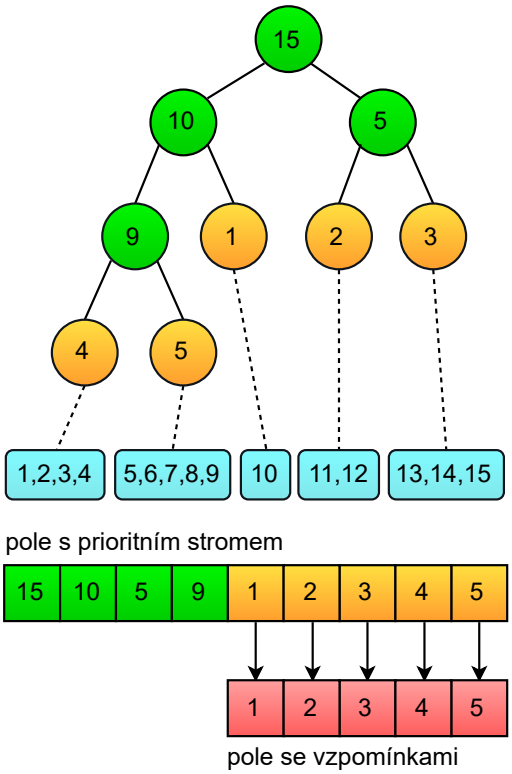
Vanilla verze DQN má tendenci nadhodnocovat kvalitu určitých akcí, což sám o sobě problém není. Ten nastává v okamžiku, kdy nejsou nadhodnoceny všechny akce stejnou měrou. V takovém případě nemůže síť konvergovat ke skutečným kvalitám akcí [12]. Tento jev se snaží eliminovat algoritmus DDQN (dvojité DQN) [6] pomocí přesnějšího využití cílové sítě. Při trénování pomocí DDQN vybírá akce v budoucích stavech primární neuronová síť, která má nejaktuálnější odhady Q-hodnot. Samotnou hodnotu vybrané akce ale dodává síť cílová, která má odhady Q-hodnoty stabilnější. Aktuální Q-hodnota provedené akce se tedy při trénování získá jako

$$Q(s,a) = r + \gamma T(s', a'[argmax_{a'} Q(s',a')]). \tag{3.7}$$

3.3 Paměť se vzpomínkami

Prvním krokem hlubokého Q-učení je inicializace vzpomínkové paměti. Jedná se o prvek, který v klasickém Q-učení úplně chybí. Do vzpomínkové paměti se ukládají informace o stavech, kterými agent během trénování prošel. Každá ukládaná informace se skládá z popisu současného stavu prostředí, provedené akce, odměny za přechod, popisu nového stavu prostředí a informace o tom, zdali je tento nový stav koncový. Práce DQN s těmito daty je na hranici učení s učitelem a učení bez učitele. K trénování neuronové sítě totiž agent nemá předem k dispozici cílové štítky, ale vytváří si odhad jejich hodnot pomocí Bellmanovy rovnice během učení.

Počáteční inicializace vzpomínkové paměti proběhne hrou agenta, jehož akce jsou zcela náhodné. Tato inicializace zabraňuje tomu, aby se agent učil z nízkého počtu vzpomínek, jejichž spojitost by velmi pravděpodobně negativně ovlivnila váhy neuronové sítě [19]. Během inicializace paměti trénování neprobíhá, to začíná v okamžiku, kdy je paměť plná. Od tohoto momentu jsou poté ze vzpomínkové paměti během každého agentova kroku vybírány



Obrázek 3.3: Vizualizace paměti s prioritními vzpomínkami uchovávající 5 vzpomínek. Obsah vzpomínky je zjednodušen na číselnou hodnotu, při čemž její priorita je shodná s touto hodnotou. V horní polovině obrázku je struktura binárního součtového stromu. Oranžové uzly uschovávají hodnoty jednotlivých priorit, zelené jsou důležité pouze pro průchod stromem. K jednotlivým prvkům se přistupuje pomocí funkcí v algoritmu 3. Modré obdélníky jsou spojeny s prioritami, které vrátí funkce $GET\_OBSARVATION$, pokud je nějaká hodnota z nich dosazena jako argument b. V dolní polovině obrázku se nachází mapování pole s prioritním stromem na pole obsahující vzpomínky.

náhodné vzorky vzpomínek, ze kterých se učí. Tyto vzorky jsou vybírány náhodně opět kvůli problému s jejich návazností, která učení může destabilizovat. Do paměti se s každým krokem algoritmu ukládá nová vzpomínka na místo aktuálně nejstarší vzpomínky. Dochází tedy k postupnému obnovování paměti. Vzpomínky s posledními navštívenými stavy jsou pro síť důležité, neboť se z nich učí kvalitě nových stavů, do kterých by se náhodnými akcemi nedostala.

Prioritní vzpomínková paměť. V původní verzi DQN probíhá výběr trénovacího vzorku náhodně, tedy všechny vzpomínky mají stejnou prioritu. Během takového výběru ale není využito skutečnosti, že z některých vzpomínek se může agent učit více než z jiných [9]. S prioritní pamětí přišla v roce 2016 opět firma DeepMind [20]. Priorita vzpomínky se stanovuje jako rozdíl Q-hodnot provedené akce podle primární a cílové neuronové sítě

$$p = (|Q(s,a) - T(s,a)| + \tau)^{\alpha}. \tag{3.8}$$

Konstanta τ je malé pozitivní číslo, které zajišťuje, že žádný stav nebude mít prioritu 0. Konstanta α nabývá hodnoty v rozsahu od 0 do 1 a určuje, jakou měrou se řídí výběr

Algoritmus 3 Pseudokód metod, které vrátí z prioritní paměti náhodnou vzpomínku. Rekurzivní metoda *GET_TREE_INDEX* (převzato z [9]) vrátí index priority vzpomínky určený parametrem b. Parametr b je hodnota v intervalu od 0 do $prioritni\_strom[0]$. Tento interval je rozdělen na podintervaly jednotlivých vzpomínek, jejichž velikost závisí na prioritě. Argument b vrátí index priority vzpomínky v jejímž intervalu se objeví. Metoda *GET_OBSERVATION* vrátí vzpomínku odpovídající prioritě, kterou určila metoda *GET_TREE_INDEX*.

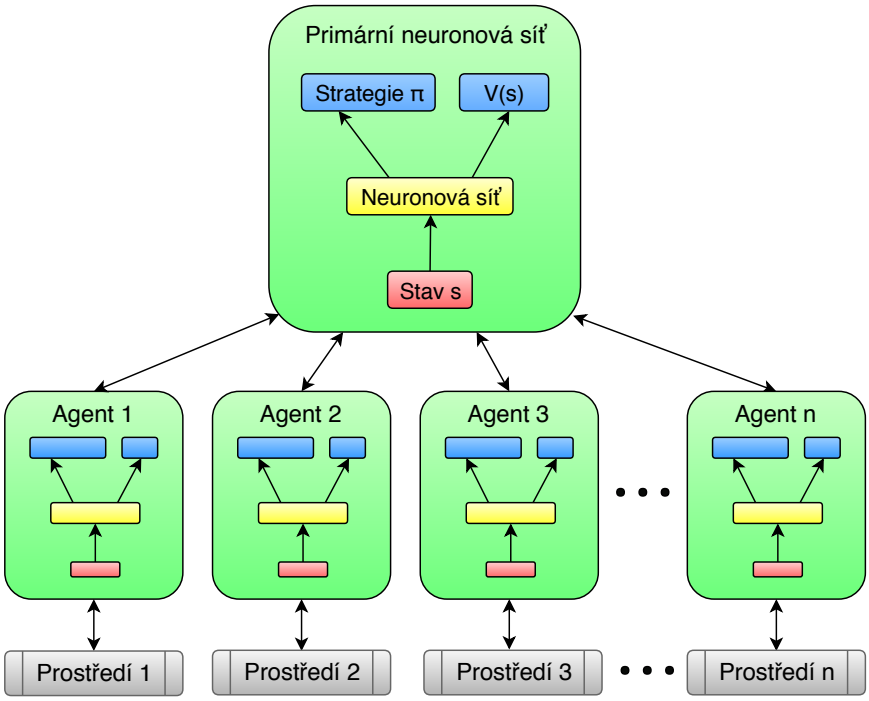
```
function GET_TREE_INDEX(strom, a, b)
    if strom[a] nemá synovský uzel then
        return a
    end if
    if strom[index_leveho_syna] <= b then
        return get_tree_index(index_leveho_syna, b)
    else
        return get_tree_index(index_praveho_syna, b - strom[index_leveho_syna])
    end if
end function

function GET_OBSERVATION(prioritni_strom, vzpominkovy_strom)
    nahoda_hodnota = Uniform(0, prioritni_strom[0])
    index = get_tree_index(prioritni_strom, 0, nahoda_hodnota)
    return vzpominkovy_strom[index - len(vzpominkovy_strom) + 1]
end function
```

trénovacího vzorku prioritou [9]. V případě, že se α rovná 0, je priorita všech vzpomínek stejná. Čím větší je rozdíl odhadů Q-hodnot primární a cílové sítě, tím více se z daného stavu může síť naučit. Priority se musí ukládat do paměti společně se vzpomínkami kvůli potřebě pozdější manipulace. Vhodnou strukturou pro výběr vzpomínek z paměti na základě priority je binární součtový strom, jehož listy ukazují do pole se vzpomínkami [20], viz obrázek 3.3. Každý uzel binárního součtového stromu má hodnotu rovnou součtu hodnot svých synů. Tuto strukturu je vhodné uložit jako pole, při čemž se indexy jednotlivých listů mapují na indexy druhého pole obsahujícího jednotlivé vzpomínky. Časová náročnost vkládání, úpravy nebo vybírání položky z prioritní paměti je $O(log\ n)$, kde n je počet uložených vzpomínek [9]. Pseudokód dvou metod pro výběr vzpomínky z prioritní paměti se nachází v algoritmu 3.

3.4 Další vývoj DQN a jeho alternativy

DQN a jeho variace s vylepšeními nejsou jedinými algoritmy, které se v posledních letech ukázaly jako účinné při hraní her. V kapitole 2 jsou zpětnovazební algoritmy rozděleny na ty, které hledají hodnotovou funkci a na ty, které hledají strategii pomocí gradientů. Q-učení a DQN náleží do první zmíněné kategorie. Obě kategorie mají své klady a zápory. Strategické gradienty (Policy Gradients) jsou použitelné na větší počet úloh. Q-učení je oproti nim zase stabilnější. Strategické gradienty konvergují obvykle rychleji, zároveň se ale častěji dostávají do lokálních minim. Q-učení je poté špatně použitelné v případě, že je stavový prostor příliš komplexní. S tím metody strategických gradientů nemají problém, jelikož upravují přímo strategii. V prostoru se spojitými akcemi poté lépe fungují strategické gradienty [32].

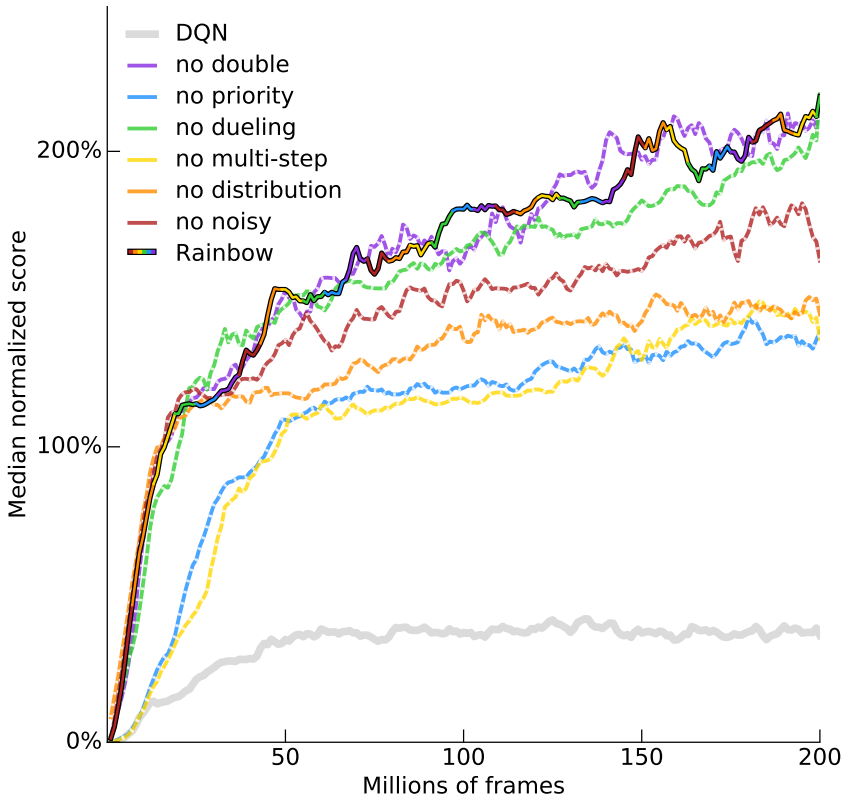


Obrázek 3.4: Vizualizace architektury algoritmu A3C (převzato z [13]).

Neuronová síť algoritmu hledající strategii pomocí gradientů se obvykle skládá ze dvou plně propojených vrstev [14]. Ty na vstup dostávají obraz nebo nějaký jiný popis prostředí a na výstup poskytují pravděpodobnost provedení konkrétních akcí.

Prvky z obou kategorií zpětnovazebních algoritmů se navzájem doplňují v algoritmu A3C (Asynchronous Advantage Actor-Critic) [16], jehož hlavní výhodou je rychlost. A3C využívá několik nezávislých agentů, kteří mají každý svoji kopii prostředí, na které se učí. Zkušenosti poté předávají primární síti, která má díky nim přehled o daleko více stavech. V A3C poskytuje neuronová síť na výstup odděleně výsledek funkce hodnoty stavu $V(s)$ a strategii $\pi(s)$. Agent následně využívá výstup funkce hodnoty stavu k aktualizaci strategie [13]. Znázornění architektury algoritmu A3C je na obrázku 3.4.

Nejmladším a nejúspěšnějším algoritmem současnosti je Rainbow [7]. Jedná se o DQN, které má nad rámec mnou implementovaných vylepšení v této práci navíc ještě prvky z vícekrokového [16], šumového [5] a distribuovaného DQN [2]. Tato metoda dosahuje úspěšnosti až 200 % lidského skóre (obrázek 3.5).

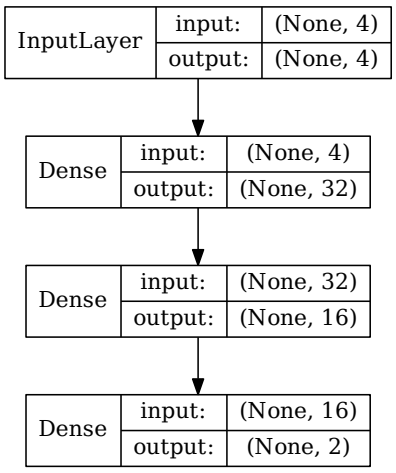


Obrázek 3.5: Znázornění efektivity učení algoritmu Rainbow na mediánu normalizovaného lidského skóre napříč 57 Atari hrami (převzato z [7]).

Kapitola 4

Experimenty s jednoduchými prostředími

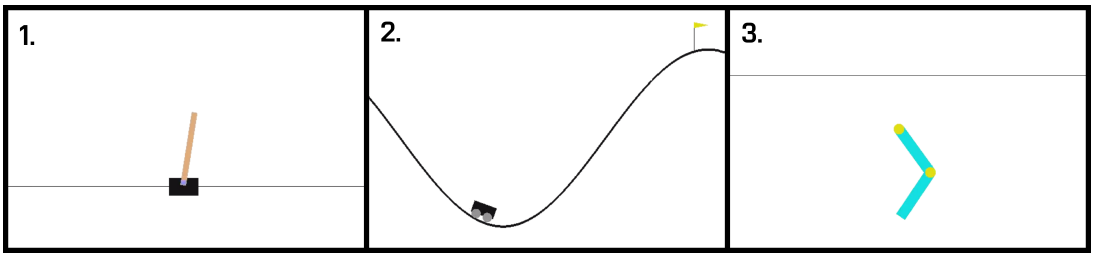
DQN má velké množství hyperparemetrů, které ovlivňují efektivitu učení. K lepšímu pochopení vlivu hyperparametrů na učení jsem provedl několik experimentů, které se zabývají hledáním jejich ideálních hodnot a celkovým dopadem výběru různých hodnot na efektivitu algoritmu. Ke zkoumání působení jednotlivých hyperparametrů na chování algoritmu, bylo zapotřebí provést velké množství testů. Trénování DQN s obrazem prostředí na vstupu, ale trvá relativně dlouhou dobu a to dělá statistické vyhodnocení experimentů takřka nemožnými. Aby bylo možné tyto testy provést, byla pozměněna architektura neuronové sítě tak, aby na vstup přijímala místo obrazu vektor čísel, který přímo popisuje prostředí. V této architektuře se tedy nenachází konvoluční vrstvy na zpracování obrazu. Vektor popisující prostředí je totiž defacto ideálním výstupem těchto vrstev. Neuronová síť se skládá ze tří plně propojených vrstev, které na vstup dostávají onen vektor čísel popisující prostředí. Práce s jednoduchými prostředími nevyžaduje velkou síť, první vrstva má tedy 32 neuronů, druhá vrstva 16 neuronů a třetí vrstva má počet neuronů roven počtu možných akcí v daném prostředí (obrázek 4.1).



Obrázek 4.1: Zjednodušená architektura DQN pro statistické testování hyperparametrů. Tento model je určen pro prostředí CartPole-v0 a CartPole-v1, které na vstup dostávají 4 informace o prostředí a na výstup dávají Q-hodnoty pro dvě akce, které lze v daném prostředí provést, viz tabulka 4.1. Celkem síť obsahuje 722 parametrů.

Tabulka 4.1: Vlastnosti jednoduchých prostředí.

Název prostředí	Informace o prostředí	Možné akce	Maximální počet kroků
CartPole-v0	Pozice vozíku Rychlost vozíku Úhel tyče Rychlost tyče na špičce	Pohyb doleva Pohyb doprava	200
CartPole-v1	Pozice vozíku Rychlost vozíku Úhel tyče Rychlost tyče na špičce	Pohyb doleva Pohyb doprava	500
MountainCar-v0	Pozice vozíku Rychlost vozíku	Pohyb doleva Žádný pohyb Pohyb doprava	200
Acrobot-v1	sin() pevného kloubu cos() pevného kloubu úhlová rychlost pevného kloubu sin() volného kloubu cos() volného kloubu úhlová rychlost volného kloubu	Pohyb doleva Pohyb doprava	500



Obrázek 4.2: Snímky obrazovky jednoduchých prostředí: 1. CartPole-v0 a CartPole-v1; 2. MountainCar-v0; 3. Acrobot-v1

4.1 Prostředí

Prostředí pro trénování agenta jsem získal z toolkitu Open AI Gym. Ten jich poskytuje rozmanitou řadu, od těch zabývajících se klasickou kontrolou řízení až po herní prostředí z konzole Atari 2600. Pro testování hyperparametrů jsem vybral čtveřici prostředí zabývající se klasickou kontrolou řízení. Snímky obrazovky těchto prostředí se nachází na obrázku 4.2 a popis vlastností v tabulce 4.1. Každé z těchto prostředí poskytuje agentovi specifický počet informací o prostředí, stanovuje počet a typ akcí, které v něm může agent provést a hlídá maximální počet kroků agenta. Po provedení takového počtu kroků nastane koncový stav a prostředí musí být zrestartováno. Obecně tedy prostředí na vstupu očekává číslo akce, kterou chce agent provést. Na výstup poté poskytuje vektor čísel popisující aktuální stav prostředí, velikost odměny za přechod mezi jednotlivými stavy a informaci o konečnosti nového stavu. Některá prostředí mají stanoveny podmínky, za kterých jsou považována za vyřešená, viz tabulka 4.2.

Tabulka 4.2: Odměny a řešení jednoduchých prostředí.

Název prostředí	Odměny	Řešení prostředí
CartPole-v0	+1 za krok s udržením rovnováhy -1 za krok se ztrátou rovnováhy	Dosažení průměrného skóre alespoň 195 ze 100 epizod
CartPole-v1	+1 za krok s udržením rovnováhy -1 za krok se ztrátou rovnováhy	Dosažení průměrného skóre alespoň 475 ze 100 epizod
MountainCar-v0	+1 za krok s dosažením cíle na kopci -1 za krok bez dosažení cíle na kopci	Dosažení průměrného skóre alespoň -110 ze 100 epizod
Acrobot-v1	0 za krok s dosažením cílové výšky -1 za krok bez dosažení cílové výšky	Nemá stanovené řešení, bere se průměrné skóre ze 100 epizod, po 100 epizodách trénování

CartPole-v0 a CartPole-v1. V obou prostředích řeší agent „problém inverzního kyvadla“. Jeho úkolem je balancovat s vozíkem tak, aby se tyč, kterou v sobě vozík veze, nenaklonila o více než 15 stupňů, pokud při tom selže, hra končí. Od začátku pohybu v prostředí agent dostává kladnou odměnu za udržování rovnováhy, v okamžiku její ztráty dostane odměnu zápornou. Díky těmto rozdílům v odměnách agent od začátku ví, které stavy jsou pro něj dobré a které ne, to mu umožňuje relativně rychlé učení. Tato prostředí jsou vhodná ke zkoumání rychlosti učení agenta. To je dáno tím, že agent v těchto prostředích není limitován nutností prohledávání stavového prostoru za vidinou lepšího stavu, jelikož se v nejlepším možném stavu nachází od začátku. Rozdílem těchto prostředí je agentův maximální počet kroků a tedy i maximální výše kumulativní odměny, viz tabulky 4.1 a 4.2. Zvýšenými hodnotami u prostředí CartPole-v1 jsou kladeny větší nároky na stabilitu učení.

MountainCar-v0. Toto prostředí se skládá ze dvou kopců, mezi kterými je údolí, na jehož dně se nachází vozík, který je kontrolován agentem. Úkolem agenta je vyjet na pravý vrchol údolí. Problém spočívá v tom, že nemá dostatečnou hybnost, aby na vrchol vyjel pouze pomocí pohybu doprava. K dosažení cíle se musí nejprve rozjet na levý kopec a až poté na pravý, pomocí tohoto manévru získá dostatečnou hybnost pro dosažení cíle. Prostředí MountainCar-v0 je přesným opakem prostředí CartPole-v0 a CartPole-v1. Agent prozkoumává svahy kopce a dostává stále stejnou zápornou odměnu, která mu neposkytuje žádnou informaci o prostředí a v důsledku čehož se nemůže učit. Učení začíná až po získání jiné než záporné odměny, tedy až po dosažení cíle. Než ovšem agent cíle náhodnými akcemi dosáhne, chvíli to trvá a proces učení probíhá delší dobu. V některých případech dokonce agent vrchol vůbec objevit nemusí a učení skončí neúspěchem. Toto prostředí je vhodné k experimentování s prohledáváním stavového prostoru.

Acrobot-v1. Zde má agent podobu ramena se dvěma klouby, při čemž jeden je pevně ukotvený a druhý je volný. Při spuštění prostředí je rameno ve svislé poloze směrem dolů. Agentovým úkolem je poté pohybovat ramenem tak, aby dosáhlo určité výšky, ta je znázorněna čarou nad ramenem. Toto prostředí je do jisté míry kombinací všech předchozích. Agent musí prohledávat stavový prostor (MountainCar-v0), jehož prohledávání ovšem není tak náročné, takže má relativně brzo přehled o kvalitě různých stavů (CartPole-v0), při čemž je důležitá stabilita učení (CartPole-v1). Její důležitost je dána jevem, který v tomto prostředí poměrně často nastává, a to že se během 100 epizod trénování zvládne neuronová síť natrénovat i destabilizovat.

Ve všech experimentech jsem použil vzpomínkovou paměť kapacity 10 000 vzorků, faktor stárnutí hodnoty 0.99 a učící konstanta hodnoty 0.001. Jako chybová funkce byl použit MSE, jelikož se ukázalo, že Huberova chybová funkce má problémy konvergovat. Pokud není explicitně zmíněno jinak, je každá hodnota v grafech průměrem z 25 stejných experimentů. Pro všechna prostředí kromě Acrobot-v1 reprezentují hodnoty počet epizod, za který byl agent schopen prostředí vyřešit. V případě Acrobot-v1 se jedná o průměrné skóre ze 100 epizod po 100 epizodách trénování. Pokud je u grafu zmíněno, že byly hodnoty normalizovány, bylo tak učiněno podle vzorce

$$n = \frac{x - min(X)}{max(X) - min(X)}, \tag{4.1}$$

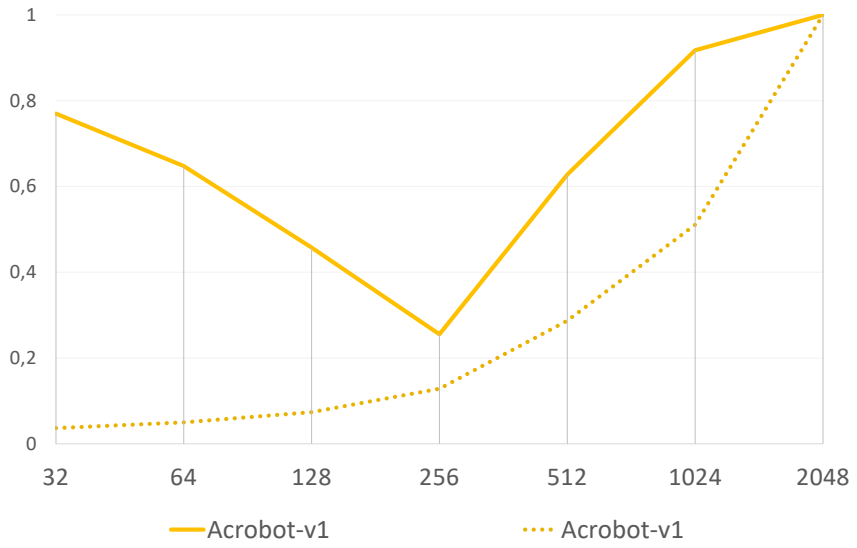
ve kterém je X soubor všech hodnot a x aktuálně normalizovaná hodnota. Jako $min(X)$ byla dosazena 0, neboť se jedná nejmenší počet epizod, ve kterém mohlo být prostředí úspěšně vyřešeno.

4.2 Velikost trénovacího vzorku

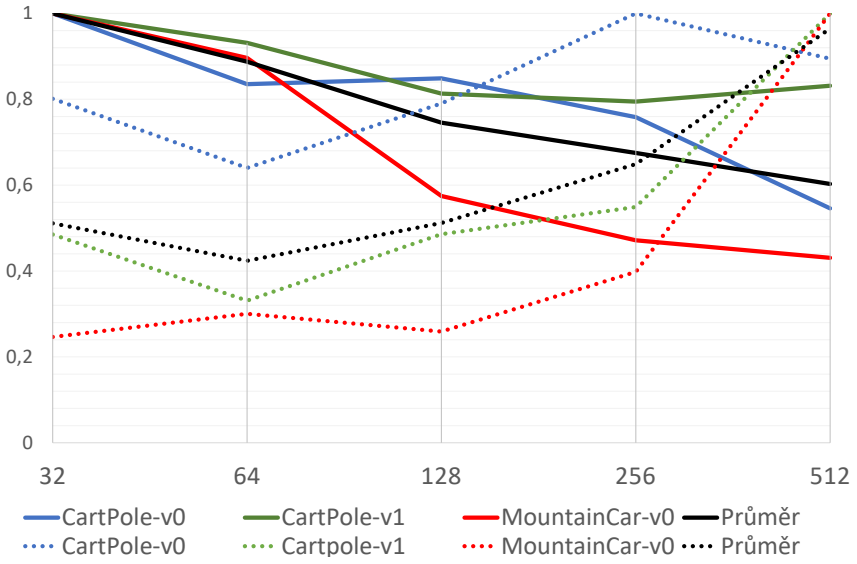
Důležitou roli při učení agenta hraje velikost trénovacího vzorku. Jedná se o počet vzpomínek, ze kterých se agent v každém kroku učí. Obecně platí, že čím je trénovací vzorek větší, tím více se agent učí. Nastavení velké hodnoty trénovacího vzorku ovšem nemusí být ideální krok za efektivnějším učením agenta. Pravděpodobně totiž povede k trénování, které bude trvat méně epizod, ale jehož časová náročnost bude větší než při použití menší velikosti trénovacího vzorku. Následující experiment hledá hranici, od které už není výhodné velikost vzorku zvětšovat. Autoři originální práce použili při trénování agenta v prostředích Atari her velikost trénovacího vzorku 32 [17]. Následující testy byly provedeny na vanilla verzi DQN.

Na obrázku 4.3 se nachází výsledky experimentu v prostředí Acrobot-v1. Díky stejnému počtu epizod pro každou velikost trénovacího vzorku je na tečkované křivce dobře vidět závislost doby trénování na velikosti vzorku. Každé zdvojnásobení velikosti trénovacího vzorku vedlo téměř ke zdvojnásobení doby trénování, přesněji se jednalo o průměrný nárůst přibližně o 76 %. Pokud se zaměříme na hodnoty průměrného skóre, z grafu vyplývá, že algoritmus je nejstabilnější pro velikost trénovacího vzorku 256. U této varianty byl zaznamenán nárůst ve skóre přibližně o 86 % oproti původní variantě s velikostí vzorku 32, nicméně s tím značně narostla i časová náročnost a to přibližně o 250 %. Výrazné zhoršení efektivity učení u vzorků větších než 256, mohlo nastat kvůli kombinaci malé velikosti vzpomínkové paměti a velkých trénovacích vzorků. Tato kombinace mohla zapříčinit přílišnou spojitost vzpomínek (kapitola 3.3), což mohlo vést k divergování sítě.

Na obrázku 4.4 se nachází výsledky experimentu pro zbývající 3 prostředí. Pro jejich analýzu je důležité stanovit, zdali se bere jako nejúspěšnější výsledek experimentu ten s nejkratší dobou trénování nebo ten s nejmenším počtem epizod. Jako první bude adresována doba trénování. Protože mají tato prostředí stanovené řešení po nichž učení končí, nebyl v každém testu proveden stejný počet epizod jako v prostředí Acrobot-v1. Tento fakt se odráží na těčkovaných křivkách, které znázorňují časovou náročnost řešení. Ta už se s každým zvětšením trénovacího vzorku nezdvojnásobuje, protože zvětšení zde pomáhá k rychlejšímu řešení prostředí. Průměr doby trvání učení je nejkratší pro velikost vzorku 64, tato varianta je časově efektivnější oproti původní velikosti trénovacího vzorku přibližně o 17.5 % a v porovnání počtu epizod o přibližně 11 %. Počet epizod, který je potřeba k úspěšnému



Obrázek 4.3: V grafu se nachází křivka efektivity učení DQN s různými velikostmi trénovacího vzorku pro prostředí Acrobot-v1. Hodnoty na ose y jsou normalizované podle vzorce 4.1 kvůli porovnání časové a výsledkové efektivity. Místo 0 za $min(X)$ byla dosazena hodnota -80, která je maximálním dosažitelným skóre pro Acrobot-v1. Hodnoty na ose x představují různé velikosti trénovacího vzorku. Plná čára znázorňuje průměrné skóre ze 100 epizod po 100 epizodách trénování a tečkovaná čára dobu trénování 100 epizod. Obecně platí, že čím je normalizovaná hodnota nižší, tím bylo řešení rychlejší/úspěšnější.



Obrázek 4.4: V grafu se nachází křivky efektivity učení DQN s různými velikostmi trénovacího vzorku pro 3 prostředí a jejich průměr. Hodnoty na ose y jsou normalizované podle vzorce 4.1 kvůli porovnání časové a výsledkové efektivity. Hodnoty na ose x představují různé velikosti trénovacího vzorku. Plné čáry znázorňují průměrný počet epizod, za který bylo prostředí vyřešeno a tečkované čáry průměrný dobu trénování. Obecně platí, že čím je normalizovaná hodnota nižší, tím bylo řešení rychlejší/úspěšnější.

řešení prostředí, se s narůstající velikostí trénovacího vzorku snižuje, z tohoto pohledu nejlépe vychází velikost vzorku 512, jeho časová náročnost je ovšem největší ze všech.

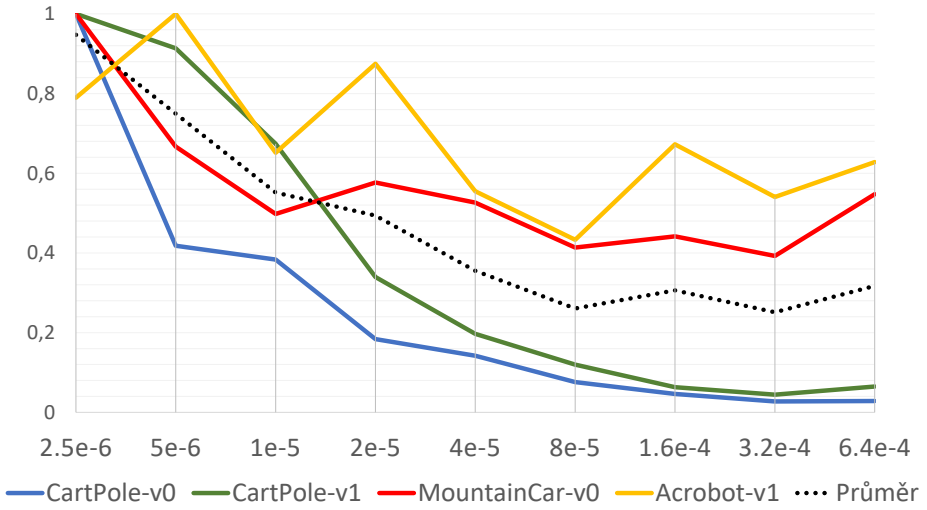
Jako kvalitní velikosti vzorku by šlo označit hodnoty 64, 128 a 256. Hodnota 128 má přibližně stejnou časovou náročnost jako původní verze trénovacího vzorku, ale do počtu epizod je efektivnější přibližně o 25.5 %. Hodnota 256 je poté časově náročnější o 27 %, ale efektivnější v počtu epizod o 32.5 %, při čemž je ještě důležité podotknout, že v prostředí Acrobot-v1 dosáhla tato varianta nejvyššího skóre ze všech. V prostředích, které agentovi poskytují informace ve formě obrazu místo vektoru čísel, je ovšem kladen důraz na časovou efektivitu, protože i její malé zhoršení může znamenat o několik hodin delší dobu trénování. V takovém případě by byla nejlepší variantou velikost trénovacího vzorku 64.

4.3 Vliv náhodných akcí na učení

Problém „explorace vs. exploitace“ je možné řešit přidáním hyperparametru ε, který funguje jako pravděpodobnost provedení náhodné akce místo té, kterou by provedl agent, viz kapitola 2.2. Hodnota ε je na začátku trénování vždy 1, tedy všechny prováděné akce jsou náhodné. Postupně se hodnota ε snižuje

$$\epsilon = \epsilon - snizujici\_konstanta \tag{4.2}$$

a agent začíná provádět akce podle předpovědí neuronové sítě. Otázkou zůstává, jak rychle by měl agent ε snižovat. V případě, že bude ε snižovat příliš pomalu, trénování bude trvat



Obrázek 4.5: V grafu se nachází porovnání hodnot efektivity učení DQN s různými hodnotami konstanty, o kterou se snižuje ε pro všechna prostředí a jejich průměr. Hodnoty na ose y jsou normalizované podle vzorce 4.1 kvůli vzájemnému porovnání hodnot z různých prostředí. Pro prostředí Acrobot-v1 byla místo 0 za $min(X)$ dosazena hodnota -80, která je maximálním dosažitelným skóre. Hodnoty na ose x představují různé konstanty, o které je snižováno ε. Plné čáry barev modré, žluté a zelené znázorňují průměrný počet epizod, za který bylo prostředí vyřešeno, plná čára barvy žluté znázorňuje průměrné skóre ze 100 epizod po 100 epizodách trénování a tečkovaná čára znázorňuje průměr všech zmíněných hodnot. Obecně platí, že čím je normalizovaná hodnota nižší, tím bylo řešení rychlejší/úspěšnější.

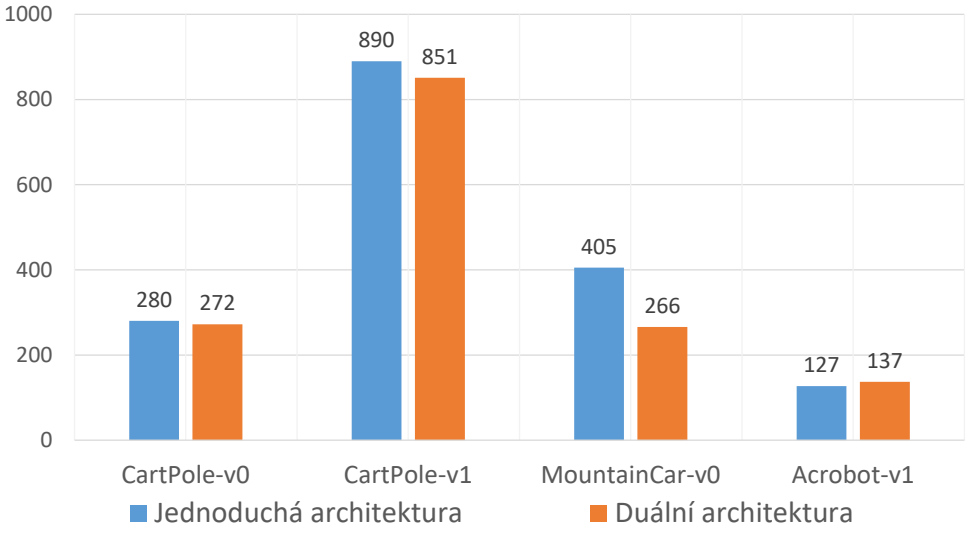
dlouho. Na druhou stranu, když ho bude snižovat moc rychle, agent nikdy nemusí nalézt cíl (prostředí MountainCar-v0), protože s nízkým ε se bude strategie řídit předpověďmi nenaučené neuronové sítě, která bude provádět stále stejné kroky, které nepovedou k cíli. Tento experiment zkoumá, jaká je ideální velikost konstanty, o kterou se v každém kroku trénování bude ε snižovat. Následující testy byly provedeny na vanilla verzi DQN.

V grafu na obrázku 4.5 lze pozorovat klesající trend počtu epizod potřebných pro vyřešení prostředí s rostoucí hodnotou konstanty pro snižování ε. U Acrobot-v1 je tento trend vidět až po proložení výsledků jednotlivých testů křivkou. Z experimentu vyplývá, že obecně je pro tato prostředí nejlepší hodnota snižování ε 3.2e-4. U nejbližší větší zkoumané hodnoty 6.4e-4 byl ve všech prostředích zaznamenán nárůst počtu epizod potřebných pro vyřešení. Zároveň je tato hodnota nejefektivnější pro všechna prostředí kromě Acrobot-v1.

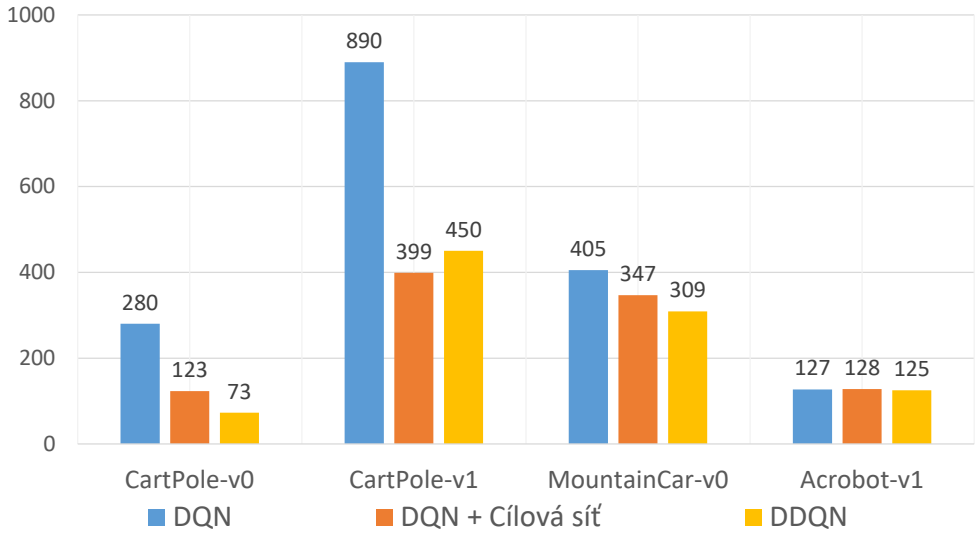
4.4 Vylepšení algoritmu DQN

Následující experiment se věnuje porovnání variant DQN s různými vylepšeními na všech jednoduchých prostředích.

Prvním zkoumaným vylepšením byla duální architektura neuronové sítě. Podstatou tohoto vylepšení je, že agent bere při výpočtu Q-hodnot v úvahu kvalitu stavu, ve kterém se nachází. Na základě této premisy by mělo dojít ke zlepšení efektivity v prostředích CartPole-v0, CartPole-v1 a MountainCar-v0, ovšem ne nutně v prostředí Acrobot-v1. Zlepšení efektivity učení v prvních třech prostředích by se mělo na experimentu projevit, protože k úspěšnému řešení prostředí by měla být neuronová síť naučena a k naučení by mělo u duální architektury dojít dříve než u základní verze. U prostředí Acrobot-v1, ale není stanovená hranice řešení prostředí a učení končí po 100. epizodě. Duální architektura tedy může být celkově efektivnější, ale efektivita se nemusí projevit během prvních 100 epizod trénování.



Obrázek 4.6: V grafu se nachází porovnání efektivity učení DQN s různými architekturami neuronové sítě pro všechna prostředí. Testy byly provedeny na vanilla verzi DQN, kde se měnila pouze architektura neuronové sítě. Obecně platí, že čím je hodnota nižší, tím bylo řešení rychlejší/úspěšnější.



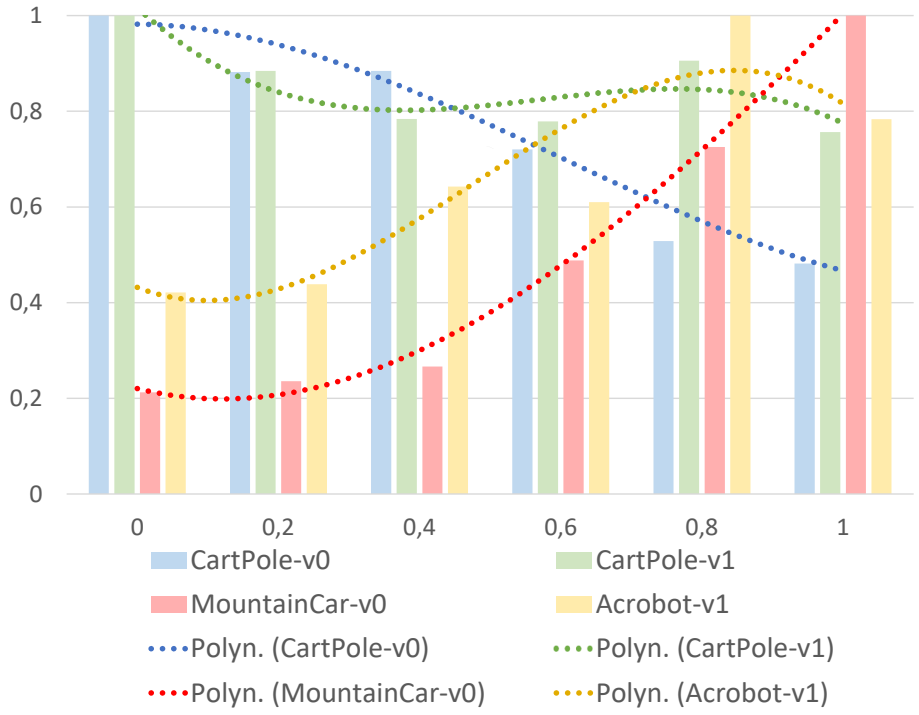
Obrázek 4.7: V grafu se nachází porovnání efektivity učení DQN s různými variantami použití cílové sítě pro všechna prostředí. Testy byly provedeny na vanilla verzi DQN, kde se měnily pouze věci spojené s cílovou sítí. Obecně platí, že čím je hodnota nižší, tím bylo řešení rychlejší/úspěšnější.

Z obrázku 4.6 vyplývá, že hypotéza o konkrétních efektivitách duální neuronové sítě v různých prostředích se potvrdila. Pro všechna prostředí kromě Acrobot-v1 se ukázala duální architektura neuronové sítě jako efektivnější. U prostředí Acrobot-v1 se poté varianta jako efektivnější neukázala, což může být způsobeno buď výše zmíněnou hypotézou a nebo tím, že v tomto prostředí je jednoduše efektivnější základní architektura. Zvýšení efektivity se pohybovalo od cca 3 % v prostředí CartPole-v0 do cca 34 % u MountainCar-v0.

Další experiment zkoumá změny přímo v algoritmu DQN, tedy porovnává efektivitu vanilla verze DQN s DQN s cílovou sítí a s dvojitým DQN. Cílová síť má za úkol poskytovat stabilnější Q-hodnoty při učení agenta. Dvojité DQN by poté ještě mělo nad rámec funkce cílové sítě zařizovat, aby nedocházelo k nerovnoměrnému nadhodnocování kvality akcí. Nejvíce by se rozdíl měl projevit v prostředích CartPole-v0 a CartPole-v1. V těchto prostředích totiž nedochází k velkému prohledávání stavového prostoru a řešení prostředí jím není limitováno. V prostředí MountainCar-v0 naopak k prohledávání stavového prostoru dochází. Drtivá většina akcí při prohledávání stavového prostoru je vybrána náhodně, což vede k relativně pravidelnému nalezení cíle. Nalézat ho začíná agent kolem epizody 200. Důsledkem je, že agent se v tomto prostředí nemůže zlepšit stejně výrazně jako například v prostředích CartPole-v0 nebo CartPole-v1. Podobnou limitaci zlepšení má prostředí Acrobot-v1, zde zase agent nemůže dosáhnout lepšího skóre než je 80.

Z obrázku 4.7 vyplývá, že přidání cílové sítě zvyšuje efektivitu učení. Rozdíl v efektivitě mezi DQN s cílovou sítí a DDQN ale není nijak významný. Z toho vyplývá, že v těchto prostředích dochází k nestabilním odhadům Q-hodnot, ale nedochází ve větší míře k nerovnoměrnému nadhodnocování kvality jednotlivých akcí. V prostředí CartPole-v0 zvýšilo přidání cílové sítě efektivitu až o 73 %, v prostředí CartPole-v1 až o 55 %, v prostředí MountainCar-v0 až o 23 % a v prostředí Acrobot asi o 1.5 %.

Další experiment zkoumá prioritní vzpomínkovou pamět. Problém je, že pro každé prostředí může být efektivní odlišná míra důrazu kladeného na prioritu vzpomínek (konstanta α ze vzorce 3.8). Experiment je proto prováděn pro několik hodnot α z intervalu od 0 do 1.

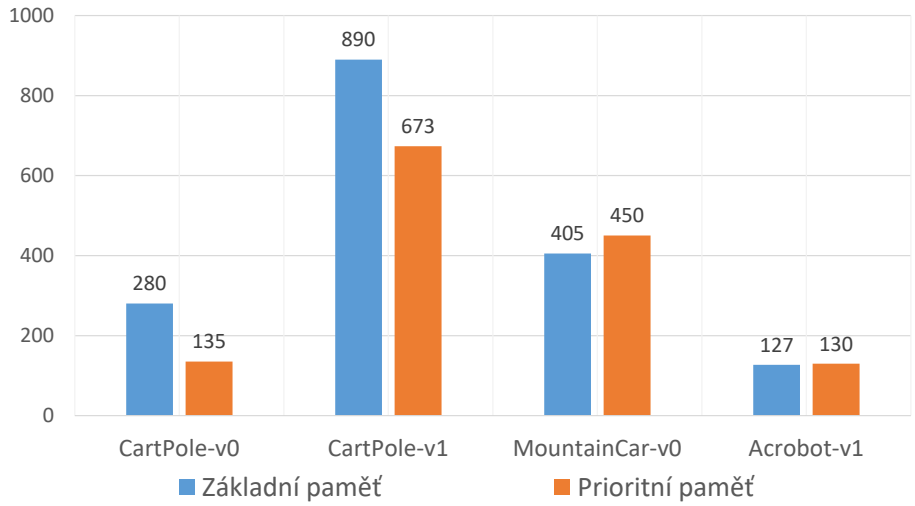


Obrázek 4.8: V grafu se nachází porovnání efektivity učení DQN s různými druhy paměti se vzpomínkami pro všechna prostředí. Testy byly provedeny na vanilla verzi DQN, kde se měnil pouze druh paměti se vzpomínkami. Hodnoty na ose y jsou normalizované podle vzorce 4.1 kvůli vzájemnému porovnání hodnot z různých prostředí. Hodnoty na ose x představují různé varianty konstanty α. Spojnice trendů jsou polynomy 3. stupně. Obecně platí, že čím je hodnota nižší, tím bylo řešení rychlejší/úspěšnější.

Z těchto hodnot se následně určí trendy pro jednotlivá prostředí, při čemž nejefektivnější hodnota se porovná s hodnotou vanilla verze DQN.

Na obrázku 4.8 je znázorněn vliv konstanty α na efektivitu učení při použití prioritní paměti. U prostředí CartPole-v0 a CartPole-v1 se objevuje společný trend rostoucí efektivity učení s rostoucí konstantou α. U prostředí MountainCar-v0 a Acrobot-v1 je pozorován opačný efekt. S rostoucí konstantou α klesá efektivita učení. Nejmenší měřená hodnota α=0.2 se poté dokonce ukázala být méně efektivní než varianta DQN bez prioritní paměti a to pro obě zmíněná prostředí. Na obrázku 4.9 se poté nachází porovnání efektivity vanilla verze DQN a nejlepších dosažených výsledků při použití prioritní pamětí. U prostředí CartPole-v0 a CartPole-v1 byly nejúspěšnější varianty prioritní paměti s hodnotou α=1. U CartPole-v0 byla tato varianta efektivnější o 52 % a u CartPole-v1 o 24 %. U prostředí MountainCar-v0 a Acrobot-v1 byla nejúspěšnější varianta prioritní paměti s α=0.2. Nicméně sám o sobě byl výsledek v prostředí MountainCar-v0 horší o 11 % a v Acrobot-v1 o 2 %.

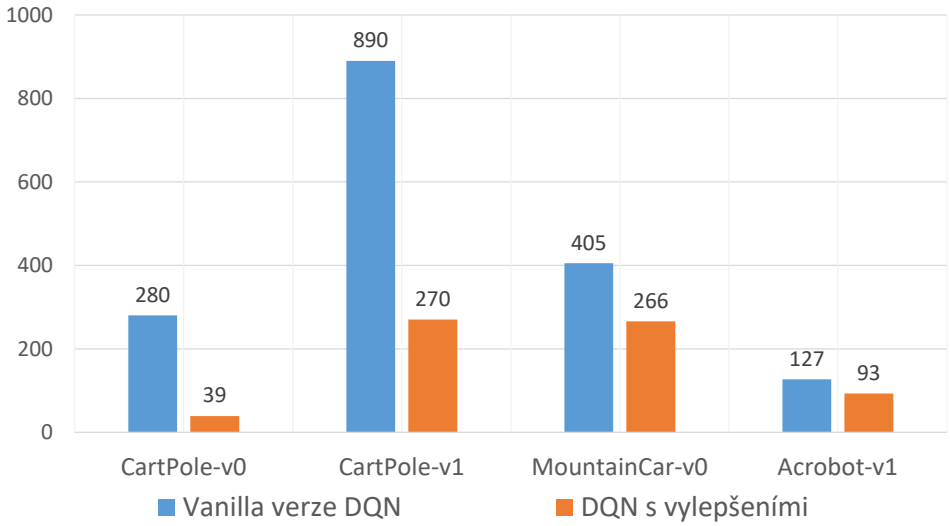
Dalším krokem ke zvýšení efektivity učení DQN je kombinace jednotlivých vylepšení. Následující experiment zkoumá efektivitu učení všech kombinací výše zmíněných vylepšení. Ukázalo se, že efektivita učení při zkombinování jednotlivých vylepšení obecně roste. Na obrázku 4.10 se nachází graf porovnávající výsledky vanilla verze DQN s nejefektivnějšími variantami obsahující vylepšení. Parametr prioritní paměti α byl jednotně nastaven na hodnotu 0.6. V případě CartPole-v0 byl nejefektivnější variantou DDQN bez prioritní paměti se základní architekturou neuronové sítě, tato varianta byla efektivnější cca o 86 %. V pro-



Obrázek 4.9: V grafu se nachází porovnání efektivity učení DQN s různými druhy paměti se vzpomínkami pro všechna prostředí. Testy byly provedeny na vanilla verzi DQN, kde se měnil pouze druh paměti se vzpomínkami. Obecně platí, že čím je hodnota nižší, tím bylo řešení rychlejší/úspěšnější.

středí CartPole-v1 byla nejúspěšnější variantou algoritmu verze DDQN s prioritní pamětí a duální architekturou neuronové sítě, tato varianta byla efektivnější cca o 70 %. V prostředí MountainCar-v0 byla nejúspěšnější variantou algoritmu verze DQN s duální architekturou neuronové sítě, ale bez prioritní paměti, tato varianta byla efektivnější cca o 34 %. V prostředí Acrobot-v1 byla nejúspěšnější variantou algoritmu verze DDQN bez prioritní paměti s duální architekturou neuronové sítě, tato varianta byla efektivnější cca o 27 %.

Při zkombinování vylepšení podporuje duální architektura neuronové sítě efektivitu učení ve všech čtyřech prostředích, využití cílové sítě jako DDQN ve třech prostředích a prioritní paměť ve dvou prostředích. Myšlenka rozdělení výpočtu Q-hodnot na dva podvýpočty se tedy ukázala jako účinná. Při zvolení duální architektury neuronové sítě se nenastavují parametry, které by nějakým větším způsobem ovlivňovali míru učení. Efektivitu tohoto vylepšení tedy není možné výrazněji zvýšit. Na druhou stranu efektivitu učení DQN s cílovou sítí potažmo DDQN ovlivnit lze, a to frekvencí aktualizací vah cílové sítě. Efektivitu učení při použití prioritní vzpomínkové paměti, lze přímo ovlivnit parametrem α. Nicméně v prostředích s nutností většího prozkoumávání stavového prostoru se použití prioritní paměti se všemi zkoumanými hodnotami α ukázalo jako neúčinné. Tento jev mohl být způsoben relativně malou vzpomínkovou pamětí nebo relativně častou aktualizací vah v cílové síti.



Obrázek 4.10: V grafu se nachází porovnání efektivity učení vanilla verze DQN s nejúspěšnější vylepšenou variantou algoritmu pro každé prostředí. Obecně platí, že čím je hodnota nižší, tím bylo řešení rychlejší/úspěšnější.

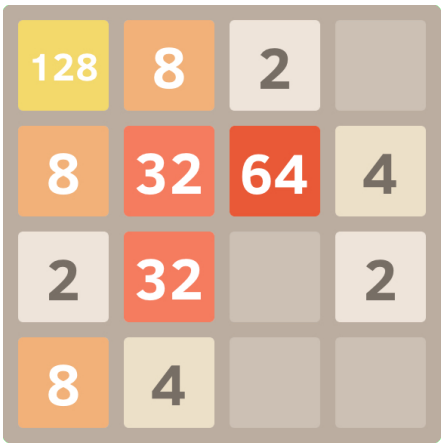
Kapitola 5

Experimenty s prostředími her

Tato kapitola se věnuje experimentům s použitím DQN na tahovou hru 2048 a Atari hry Space Invaders, Breakout a Beam Rider. Výzvu představují odlišné popisy prostředí - matice s hodnotami hrací desky, RAM paměť a obraz prostředí.

5.1 Hrací pole jako popis prostředí

Hra 2048 (obrázek 5.1) představuje pro DQN zajímavou výzvu. K dosažení slušného skóre je totiž zapotřebí dlouhodobé plánování akcí. Takovou úlohu by tedy měly lépe zvládat algoritmy, které jsou schopny prozkoumat budoucí stavy jako minimax, alfa-beta nebo expectimax. DQN pouze aproximuje, jaké akce budou pravděpodobně nejhodnotnější. Zpřesňování této aproximace, ale nepochybně zabere značné množství času. To je způsobeno velkým stavovým prostorem a pomalou propagací odměn za budoucí akce do aktuálních Q-hodnot.



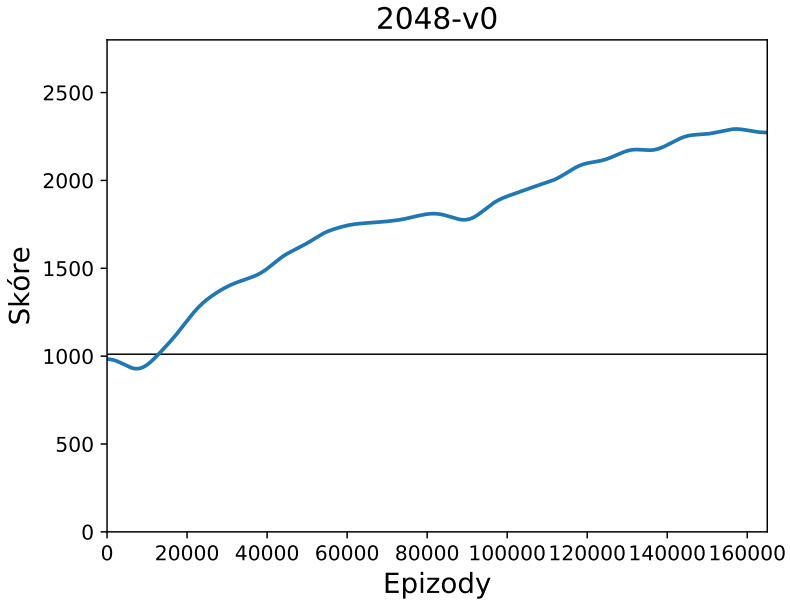
Obrázek 5.1: Hrací pole hry 2048.

Pravidla. Hráč přesouvá herní pole obsahující čísla ve čtyřech směrech: doleva, doprava, dolů a nahoru. Těmito pohyby se poté snaží srazit pole, která obsahují stejná čísla. Pokud se dvě takováto pole srazí, v poli, které je blíže stěně hrací desky ve směru pohybu, se objeví jejich součet a druhé pole zanikne. Akce je proveditelná pouze pokud pomocí ní hráč přesune alespoň jedno pole. Na začátku každého tahu se náhodně zaplní jedno prázdné hrací pole a to buď číslem 2 nebo 4. Skóre hráče začíná na nule. Při každém spojení polí

se k němu přičte hodnota nově vzniklého pole. Kromě tohoto sčítaného skóre se jako berná mince úspěšnosti hráče bere ještě hodnota pole s nejvyšším číslem. Hra končí, když prostředí na začátku tahu nemůže přidat nové pole, protože jsou všechna obsazena a zároveň nemůže hráč provést žádný tah.

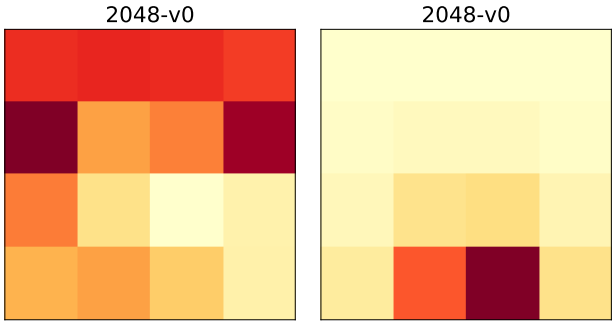
Open AI Gym obsahuje implementaci hry 2048, která jako popis prostředí poskytuje zploštěnou matici šestnácti hodnot reprezentující skóre v jednotlivých polích. Před vstupem do neuronové sítě tuto matici normalizuji. Za spojení polí prostředí poskytuje odměnu rovnou součtu aktuálně spojených polí, jinak je odměna 0. Má implementace má odměnový systém pozměněný. Pokud agentem provedenou akcí dojde ke spojení polí, dostane odměnu 1 a to univerzálně, ať došlo o spojení jakkoliv hodnotných polí. Jestliže agent provede akci, kterou nelze provést, dostane zápornou odměnu -0.1 a nadále se bude nacházet v tomtéž stavu. V okamžiku, kdy agent provede akci, jenž způsobí konec hry dostane zápornou odměnu -1. Ve všech ostatních případech dostane agent odměnu 0.

Jako nejefektivnější se ukázala varianta DDQN bez duální architektury neuronové sítě a bez prioritní paměti. Neuronová síť měla tři plně propojené vrstvy, při čemž první dvě měli 256 neuronů a třetí 4 neurony - pro každou akci jeden. Paměť měla kapacitu 200 000 vzorků. Jako chybová funkce byla použita MSE. Konstanta ε byla lineárně snižována z hodnoty 1 na hodnotu 0.1 během milionů kroků, od miliontého kroku měla nadále stejnou hodnotu a to 0.1. Učící konstanta byla nastavena na hodnotu 0.00025. Faktor stárnutí může být u deterministických her nastaven na hodnotu 1. Nicméně hra 2048 zcela deterministická není a to kvůli náhodnému vkládání polí na začátku tahu. Proto byla hodnota faktoru stárnutí nastavena na 0.99. Aktualizace vah v neuronové síti probíhala každých 10000 kroků. Graf průběhu učení se nachází na obrázku 5.2.

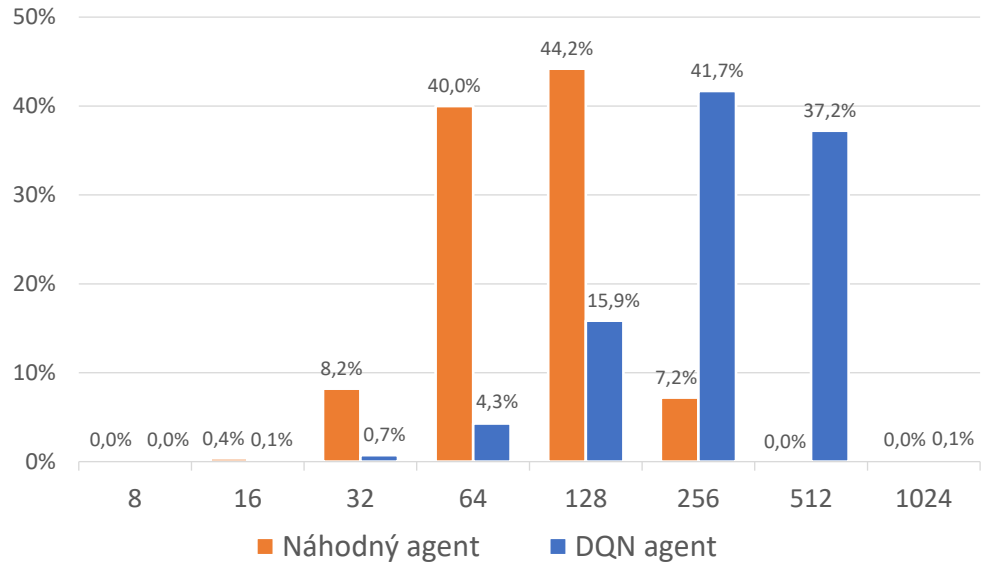


Obrázek 5.2: V grafu se nachází křivka učení agenta používajícího algoritmu DQN. Černá přímka znázorňuje průměrné skóre hráče s náhodnými pohyby. Modrá křivka zobrazuje aktuální skóre v dané epizodě vyhlazené pomocí Gaussova filtru se standardní odchylkou Gaussova jádra 3100.

Trénování trvalo celkem 165 000 epizod s 37 500 000 kroky. Agent byl následně podroben 10 000 trénovacích her, z nichž byla stanovena jeho úspěšnost. Agentovo průměrné skóre činí 3 700 bodů a maximální dosažené poté 10 896 bodů. Na obrázku 5.4 se nachází procentuální rozložení největších dosažených polí na konci hry. V této kategorii úspěšnosti agentův rekord činí pole s hodnotou 1024. Ideální strategie hry 2048 čítá uchovávání polí s největší hodnotou v některém z rohů. Z obrázku 5.3 ale vyplývá, že agent uchovává pole s největší hodnotou uprostřed dolní poloviny hracího pole, nikoliv v rozích. Z dlouhodobého hlediska tedy nemá naučenou úplně ideální strategii. Z obrázku 5.3 dále vyplývá, že agent rozhoduje o svých akcích hlavně podle uskupení polí, které mají menší hodnotu. Tato pole jsou na opačné straně hrací desky od polí s největšími hodnotami. Agent totiž za spojení dostane vždy stejnou odměnu a snadněji se spojují pole menších hodnot do kterých se nemíchají pole s velkými hodnotami.



Obrázek 5.3: Na levém obrázku se nachází teplotní mapa, která zobrazuje důležitost hracího pole pro první vrstvu neuronové sítě. Čím je pole tmavší, tím je důležitější. Na pravém obrázku se nachází teplotní mapa, která zobrazuje, kde agent uchovává pole s největší hodnotou. Čím je pole tmavší, tím větší hodnoty se v něm nacházejí.



Obrázek 5.4: V grafu se nachází procentuální rozložení největších dosažených hodnot v jednom poli na konci hry hrané náhodným agentem a agentem trénovaným DQN.

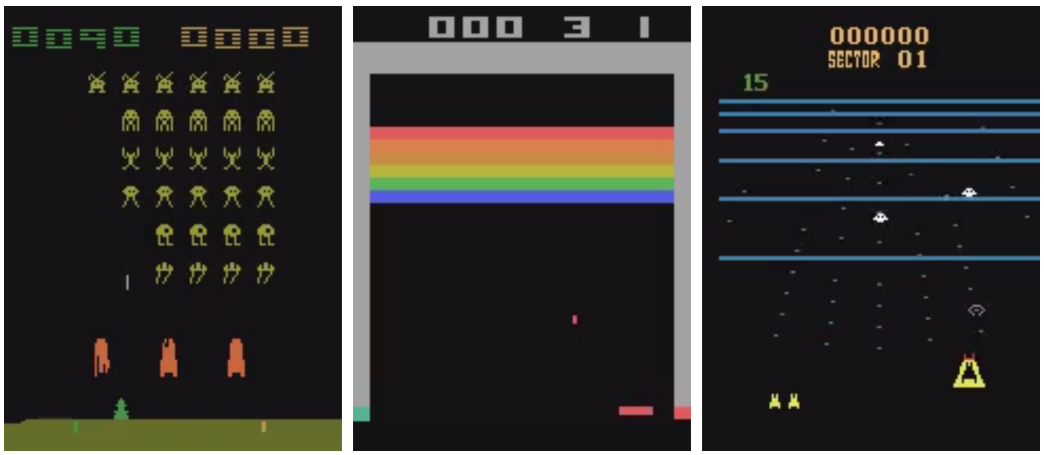
5.2 RAM paměť jako popis prostředí

Další možností popisu stavu hry je její RAM paměť. Ta obsahuje všechny důležité informace o poloze agenta, o jeho rychlosti nebo o blížícím se útoku nepřátel. Trénování agenta s RAM pamětí na vstupu neuronové sítě probíhá v principu úplně stejně, jako když od prostředí dostává stavový vektor nebo jeho obraz. Zprvu neví, které bajty jsou pro něj důležité, ale s postupem času začne vnímat, jaký bajt má pozitivní a jaký má negativní vliv na získání odměny.

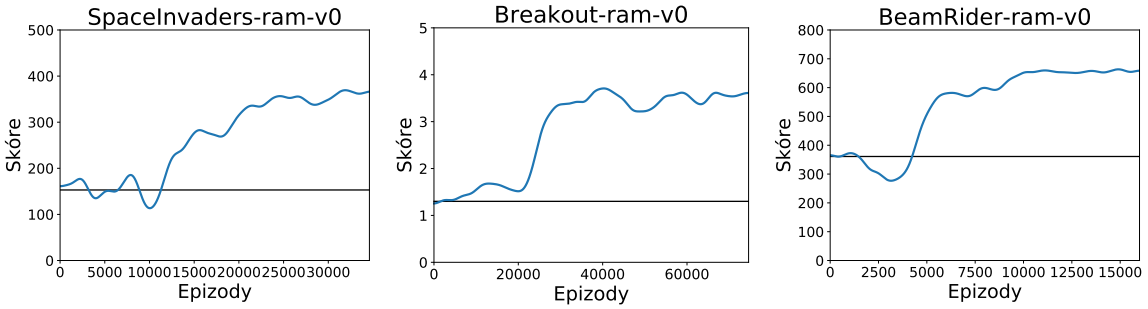
Open AI Gym obsahuje řadu Atari her, které agentovi jako popis prostředí poskytují buďto RAM paměť velikosti 128 bajtů a nebo svůj obraz. Já jsem vybral trojici takovýchto her, u nichž porovnám efektivitu učení z RAM paměti a z obrazu prostředí. Konkrétně jsem vybral hry Space Invaders, Breakout a Beam Rider. Obraz jejich prostředí je vidět na obrázku 5.5. Dosažené výsledky nejsou přímo porovnatelné s těmi, kterých dosáhl DeepMind v článku [18]. Za prvé bylo v mých experimentech provedeno méně kroků. Za druhé tato prostředí opakují vybranou akci následujících k stavů, kde k je vybráno s rovnoměrným rozložením pravděpodobnosti z intervalu $\{2,3,4\}$, místo následujících tří stavů jako ve zmíněném článku.

Space Invaders. Agentův avatar se nachází na spodní straně obrazovky a má podobu vesmírné lodi. Ta má za úkol zničit nepřátelské lodě nacházející se nad ní. Agentova loď se může pohybovat do stran a nebo střílet na nepřátele. Nepřátelé se pohybují rovněž ze strany na stranu a opětují palbu po hráčově avataru. Na zničení lodi stačí jedna střela, a to ať se jedná o loď nepřátelskou nebo o loď agenta. Před nepřátelskou palbou se může agent schovat za zábrany. Ty ale nepřátele, střílením po čase zničí. Hráč má celkem tři životy.

Breakout. Agentův avatar má podobu desky. Nachází se na spodní straně obrazovky, při čemž má za úkol ničit barevné krychle nacházející se nad ním. Deskou může agent pohybovat do stran a tím odrážet kuličku, která bude ničit jednotlivé krychle. Hráč má celkem tři životy, pokud při hře mine kuličku, ztrácí jeden život.



Obrázek 5.5: V grafu se nachází snímky obrazu Atari her, zleva Space Invaders, Breakout a Beam Rider.

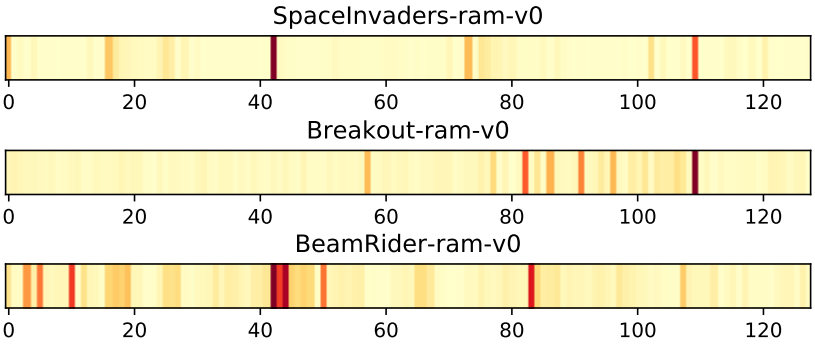


Obrázek 5.6: V obrázku se nachází grafy s křivkami učení agenta používajícího algoritmus DQN v Atari hrách, zleva Space Invaders, Breakout a Beam Rider. Černá přímka znázorňuje průměrné skóre hráče s náhodnými pohyby. Modrá křivka zobrazuje aktuální skóre v dané epizodě vyhlazené pomocí Gaussova filtru se standardní odchylkou Gaussova jádra 300 pro Beam Rider, 650 pro Space Invaders a 1400 pro Breakout.

Beam Rider. Agentův avatar má podobu vesmírné lodě a nachází se na spodní straně obrazovky. Jeho úkolem je zničit nepřátelské lodě, které nad ním krouží. Jeho loď se při tom může pohybovat do stran a střílet. Nepřátelské lodě opětují střelbu směrem na hráčova avatara. Na zničení lodi stačí jedna střela, a to ať se jedná o loď nepřátelskou nebo o loď agenta. Hráč má celkem tři životy. Při jejich ztrátě je může doplňovat pomocí speciálních objektů, které občas letí v jeho směru.

Na obrázku 5.6 se nachází křivky učení nejúspěšnějších agentů využívajících DQN. Otestovány byly všechny kombinace vylepšení. Jako nejúspěšnější se ukázalo DDQN se základní pamětí a základní architekturou neuronové sítě. Ta obsahovala dvě plně propojené vrstvy, při čemž první dvě měla 512 neuronů a druhá měla počet neuronů rovný počtu akcí v dané hře. Jednalo se o tak architekturu originálního DQN, bez konvolučních vrstev. Paměť měla kapacitu 200 000 vzorků. Jako chybová funkce byla použita MSE. Konstanta ε byla lineárně snižována z hodnoty 1 na hodnotu 0.1 během milionů kroků, od miliontého kroku měla nadále stejnou hodnotu a to 0.1. Učící konstanta byla nastavena na hodnotu 0.00025 a faktor stárnutí na hodnotu na 0.99. Aktualizace vah v neuronové síti probíhala každých 10 000 kroků.

Jedno trénování trvalo celkem 260 hodin, během kterých každý agent provedl cca 30 000 000 kroků. Ve hře Space Invaders poté agent odehrál celkem 34 500 epizod, ve hře Breakout 74 500 epizod a ve hře Beam Rider 16 000 epizod. Průměrné a maximální agentovo skóre z jednotlivých her se nachází v tabulce 5.1. Nárůst skóre činil oproti náhodnému hráči až 160 % ve hře Space Invaders, 190 % ve hře Breakout a 87 % ve hře Beam Rider. Na obrázku 5.7 se nachází teplotní mapy agentem určené důležitosti jednotlivých bajtů RAM paměti.



Obrázek 5.7: Na obrázku se nachází teplotní mapy RAM pamětí jednotlivých prostředí, které znázorňují důležitost jednotlivých bajtů pro rozhodování agenta. Čím je bajt tmavší, tím je pro síť důležitější.

5.3 Obraz jako popis prostředí

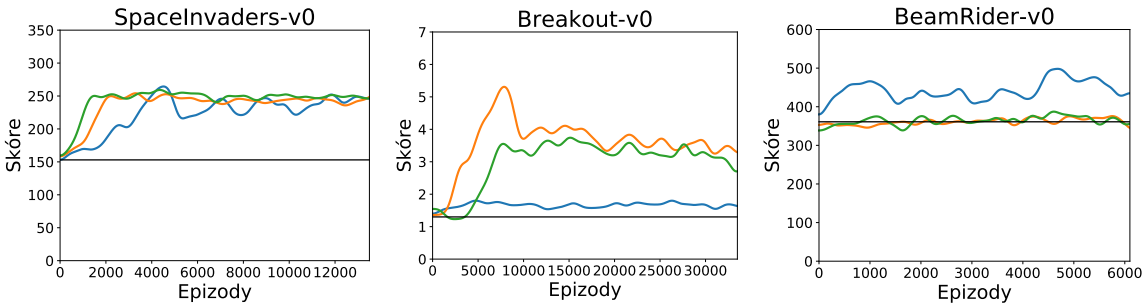
Posledním zkoumaným popisem stavu prostředí je jeho obraz. Jeho předzpracování a použitá architektura neuronové sítě je popsána v kapitole 3.1. Kvůli časové a zdrojové náročnosti na učení z obrazu jsem zkoumal pouze tři varianty agenta.

První varianta DQN přijímá na vstup neuronové sítě 2 snímky obrazu a její vzpomínková paměť má kapacitu na 200 000 vzorků. Druhá varianta na vstup neuronové přijímá 4 snímky obrazu a má poloviční velikost vzpomínkové paměti 100 000 vzorků. Třetí varianta se liší použitou chybovou funkcí. První dvě varianty používají MSE, třetí varianta používá Huberovu chybovou funkci. Na vstup neuronové sítě poté přijímá 4 snímky obrazu a má kapacitu vzpomínkové paměti 100 000 vzorků.

Počet snímků, který popisuje stav prostředí a je vstupem neuronové sítě, citelně ovlivňuje množství použité RAM paměti během trénování. V případě 4 snímků popisujících prostředí bude paměť potřebovat jednou tolik místa jako v případě popisu prostředí pomocí dvou snímků. V originální verzi DQN uchovávala paměť 1 000 000 vzpomínek [17]. Při šedotónovém snímku velikosti 84x84 pixelů, činí velikost jednoho snímku 56 kB. V okamžiku, kdy je stav popsán čtyřmi takovými snímky při kapacitě paměti 1 000 000 vzorků, bude množství požadované RAM paměti 225 Gb. Výpočty s takovými požadavky na RAM paměť nebyly možné. Ve všech testovaných variantách jsem nakonec nastavil vzpomínkovou paměť na takovou kapacitu, která si nárokuje desetinu velikosti RAM paměti z článku [17] tedy 22 Gb.

Experiment jsem provedl s DDQN s prioritní vzpomínkovou paměti (α=0,6) a se základní architekturou neuronové sítě. V experimentu byl nastaven faktor stárnutí na hodnotu 0.99 a učící konstanta na hodnotu 0.00025. Konstanta ε byla lineárně snižována z hodnoty 1 na hodnotu 0.1 během milionů kroků, od miliontého kroku měla nadále stejnou hodnotu a to 0.1. Aktualizace vah v neuronové síti probíhala každých 10 000 kroků.

Na obrázku 5.8 se nachází křivky učení jednotlivých zkoumaných variant DQN. Ve hře Space Invaders dosahují všechny tři na konci trénování podobného skóre. Nejrychleji k tomuto skóre konvergovala varianta s Huberovou chybovou funkcí. Varianty se čtyřmi snímky obrazu jako popisem prostředí obecně konvergovali rychleji. Učení varianty se dvěma snímky obrazu bylo nestabilní. V prostředí hry Breakout nejrychleji konvergovala varianta se čtyřmi snímky obrazu a chybovou funkcí MSE. Varianta s Huberovou chybovou funkcí dosahovala lehce horších výsledků. Varianta se dvěma snímky obrazu nekonvergovala vůbec. Ve hře Beam Rider se jako jediná úspěšná varianta ukázala ta se dvěma snímky obrazu,



Obrázek 5.8: Na obrázku se nachází grafy s křivkami učení agenta používajícího algoritmus DQN v Atari hrách, zleva Space Invaders, Breakout a Beam Rider. Černá přímka znázorňuje průměrné skóre hráče s náhodnými pohyby. Modrá křivka znázorňuje agenta, jehož neuronová síť přijímá na vstupu 2 snímky obrazu. Oranžová křivka znázorňuje agenta, jehož neuronová síť na vstup přijímá 4 snímky obrazu. Zelená křivka znázorňuje agenta, který na vstup neuronové sítě dostává 4 snímky obrazu a používá Huberovu chybovou funkci. Křivky jsou vyhlazené pomocí Gaussova filtru se standardní odchylkou Gaussova jádra 110 pro Beam Rider, 240 pro Space Invaders a 600 pro Breakout.

ostatní nekonvergovaly vůbec. Z výsledků experimentu nelze vyvodit jednoznačný závěr, o tom jakým způsobem ovlivňuje počet snímků učení agenta, ani o tom, jestli je Huberova chybová funkce vhodnější než MSE.

Průměrná doba trénování se pohybovala kolem 160 hodin a bylo v ní provedeno cca 11 500 000 kroků. V prostředí Space Invaders trénování probíhalo 13 500 epizod, v prostředí Breakout 33 500 epizod a v prostředí Beam Rider 6 100 epizod. Průměrné a maximální agentovo skóre z jednotlivých her se nachází v tabulce 5.1 Nárůst skóre činil oproti náhodnému hráči 83 % ve hře Space Invaders, 138 % ve hře Breakout a 20 % ve hře Beam Rider.

Tabulka 5.1: Tabulka porovnává skóre náhodného agenta a agenta trénovaného DQN. Průměrné skóre agenta s DQN bylo stanoveno jako průměr skóre v posledních 100 epizodách trénování. Nejlepší skóre poté jako maximální hodnota z těchto epizod.

	Space Invaders	Breakout	Beam Rider
Náhodné akce	153	1.3	361
RAM (průměr)	398	3.8	674
RAM (nejlepší)	955	12	1284
Obraz (průměr)	280	3.1	433
Obraz (nejlepší)	830	8	804

Kapitola 6

Závěr

Cílem práce bylo naučit neuronovou síť hrát hry. K dosažení tohoto cíle byl vybrán algoritmus DQN. Ten byl implementován a obohacen o vylepšení v podobě cílové sítě, DDQN, duální architektury neuronové sítě a prioritní vzpomínkové paměti.

Nejdříve byly provedeny experimenty na jednoduchých prostředích. Ty zkoumaly vliv hodnot hyperparametrů algoritmu na efektivitu učení a následně vliv vylepšení algoritmu na efektivitu učení. Prvním zkoumaným hyperparametrem byla velikost trénovacího vzorku. Jako vhodné byly vybrány velikosti 64, 128 a 256. Druhým zkoumaným hyperparametrem byl rozpad ε. Zde byla jako nejvhodnější vybrána hodnota 3.2e-4. Vylepšení DQN zvedly efektivitu učení oproti vanilla verzi algoritmu v prostředí CartPole-v0 až o 86 %, v prostředí CartPole-v1 až o 70 %, v prostředí MountainCar-v0 až o 34 % a v prostředí Acrobot-v1 až o 27 %.

Zkušenosti z těchto experimentů byly následně využity při trénování agenta v tahové hře 2048. Zde bylo dosaženo průměrného skóre 3 700, což představuje zlepšení o 265 % oproti náhodnému hráči. Nejvyšší dosažená hodnota v jednom poli byla až 1024.

Následně bylo DQN použito při trénování agenta na Atari hrách Space Invaders, Breakout a Beam Rider. Jako popis prostředí byla v jednom případě zvolena RAM paměť a ve druhém obraz prostředí. Ve všech případech dosahoval lepšího skóre agent, který jako popis prostředí přijímal paměť RAM. Ve hře Space Invaders bylo skóre DQN navýšeno až o 160 % oproti náhodnému hráči, ve hře Breakout o 190 % a ve hře Beam Rider o 87 %. Experimenty dohromady zabraly celkem 433 dní CPU času.

Práce mi poskytla vhled do zpětnovazebního učení a jeho algoritmů využívajících neuronové sítě. Vylepšení, které by dále mohli zvednout efektivitu implementovaného algoritmu, jsou prvky vícekrokového, šumového a distribuovaného DQN.

Literatura

[1] Barták, R.: Umělá inteligence II, Zpětnovazební učení. 2018, http://ktiml.mff.cuni.cz/~bartak/ui2/lectures/lecture13.pdf.

[2] Bellemare, M. G.; Dabney, W.; Munos, R.: A Distributional Perspective on Reinforcement Learning. *CoRR*, ročník abs/1707.06887, 2017, 1707.06887. URL http://arxiv.org/abs/1707.06887

[3] Bottomley, H. G.: How many Tic-Tac-Toe (noughts and crosses) games are possible? 2002, http://www.se16.info/hgb/tictactoe.htm.

[4] Fern, X. Z.: Reinforcement Learning. 2008, http://web.engr.oregonstate.edu/~xfern/classes/cs434/slides/RL-1.pdf.

[5] Fortunato, M.; Azar, M. G.; Piot, B.; aj.: Noisy Networks for Exploration. *CoRR*, ročník abs/1706.10295, 2017, 1706.10295. URL http://arxiv.org/abs/1706.10295

[6] van Hasselt, H.; Guez, A.; Silver, D.: Deep Reinforcement Learning with Double Q-learning. *CoRR*, ročník abs/1509.06461, 2015, 1509.06461. URL http://arxiv.org/abs/1509.06461

[7] Hessel, M.; Modayil, J.; van Hasselt, H.; aj.: Rainbow: Combining Improvements in Deep Reinforcement Learning. *CoRR*, ročník abs/1710.02298, 2017, 1710.02298. URL http://arxiv.org/abs/1710.02298

[8] Huang, S.: Introduction to Various Reinforcement Learning Algorithms. Part I (Q-Learning, SARSA, DQN, DDPG). blogpost (english), January 2018, https://goo.gl/3XmGnn.

[9] Janisch, J.: Let's make a DQN: Double Learning and Prioritized Experience Replay. blogpost (english), November 2016, https://goo.gl/5WPedM.

[10] Janisch, J.: Let's make a DQN: Full DQN. blogpost (english), October 2016, https://goo.gl/gZGiFg.

[11] Juliani, A.: Simple Reinforcement Learning with Tensorflow Part 0: Q-Learning with Tables and Neural Networks. blogpost (english), August 2016, https://goo.gl/q6V5fv.

[12] Juliani, A.: Simple Reinforcement Learning with Tensorflow Part 4: Deep Q-Networks and Beyond. blogpost (english), September 2016, https://goo.gl/LWvMWe.

[13] Juliani, A.: Simple Reinforcement Learning with Tensorflow Part 8: Asynchronous Actor-Critic Agents (A3C). blogpost (english), December 2016, https://goo.gl/N9zEja.

[14] Karpathy, A.: Deep Reinforcement Learning: Pong from Pixels. blogpost (english), May 2016, http://karpathy.github.io/2016/05/31/rl/.

[15] Matiisen, T.: Demystifying Deep Reinforcement Learning. blogpost (english), December 2015, https://ai.intel.com/demystifying-deep-reinforcement-learning.

[16] 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

[17] Mnih, V.; Kavukcuoglu, K.; Silver, D.; aj.: Playing Atari with Deep Reinforcement Learning. *CoRR*, ročník abs/1312.5602, 2013, 1312.5602. URL http://arxiv.org/abs/1312.5602

[18] Mnih, V.; Kavukcuoglu, K.; Silver, D.; aj.: Human-level control through deep reinforcement learning. *Nature*, ročník 518, Feb 2015: s. 529 EP –. URL https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf

[19] Patterson, J.: Introduction To Deep Q-Learning. blogpost (english), April 2017, https://goo.gl/6x3xuN.

[20] Schaul, T.; Quan, J.; Antonoglou, I.; aj.: Prioritized Experience Replay. *CoRR*, ročník abs/1511.05952, 2015, 1511.05952. URL http://arxiv.org/abs/1511.05952

[21] Seno, T.: Welcome to Deep Reinforcement Learning Part 1 : DQN. blogpost (english), August 2017, https://goo.gl/bLq79g.

[22] Smart, B.: Reinforcement Learning: A User's Guide. 2018, http://www2.econ.iastate.edu/tesfatsi/RLUsersGuide.ICAC2005.pdf.

[23] Sutton, R. S.; Barto, A. G.: *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. A Bradford Book, 1998, ISBN 0262193981.

[24] Wang, Z.; de Freitas, N.; Lanctot, M.: Dueling Network Architectures for Deep Reinforcement Learning. *CoRR*, ročník abs/1511.06581, 2015, 1511.06581. URL http://arxiv.org/abs/1511.06581

[25] Wikipedia: Artificial intelligence — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Artificial%20intelligence&oldid=839797551, 2018, [Online; accessed 06-May-2018].

[26] Wikipedia: Bellman equation — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Bellman%20equation&oldid=838122252, 2018, [Online; accessed 15-May-2018].

[27] Wikipedia: Huber loss — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Huber%20loss&oldid=820848344, 2018, [Online; accessed 13-May-2018].

[28] Wikipedia: Markov decision process — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Markov%20decision%20process&oldid=838091969, 2018, [Online; accessed 06-May-2018].

[29] Wikipedia: Q-learning — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Q-learning&oldid=837234781, 2018, [Online; accessed 06-May-2018].

[30] Wikipedia: Reinforcement learning — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Reinforcement%20learning&oldid=838124791, 2018, [Online; accessed 06-May-2018].

[31] Wikipedia: Zpětnovazební učení — Wikipedia, The Free Encyclopedia. http://cs.wikipedia.org/w/index.php?title=Zp%C4%9Btnovazebn%C3%AD%20u%C4%8Den%C3%AD&oldid=15606283, 2018, [Online; accessed 12-May-2018].

[32] Yu, F.: Deep Q Network vs Policy Gradients - An Experiment on VizDoom with Keras. blogpost (english), October 2017, https://flyyufelix.github.io/2017/10/12/dqn-vs-pg.html.

Příloha A

Obsah přiloženého CD

- **doc/** – adresář s technickou zprávou v PDF a s jejími zdrojovými kódy
- **src/** – adresář se zdrojovými kódy k experimentům
- **videos/** – adresář s videi zachycující výsledky trénování
- **poster.pdf** – plakát k práci