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

POSILOVANÉ UČENÍ PRO 3D HRY

REINFORCEMENT LEARNING FOR 3D GAMES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MICHAL BERÁNEK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. MICHAL HRADIŠ, Ph.D.**
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: **Beránek Michal**
Program: Informační technologie
Název: **Posilované učení pro 3D hry**
Reinforcement Learning for 3D Games
Kategorie: Zpracování obrazu

Zadání:

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 vhodné herní prostředí a sadu úloh pro experimenty.
5. Implementujte navrženou metodu a proveďte experimenty ve zvoleném prostředí.
6. Porovnejte dosažené výsledky a diskutujte možnosti budoucího vývoje.
7. Vytvořte stručné video prezentující vaši práci, její cíle a výsledky.

Literatura:

- Machado et al., Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents, Journal of Artificial Intelligence Research, 2018.
- Hessel et al., Rainbow: Combining Improvements in Deep Reinforcement Learning. Proceedings of the AAAI Conference on Artificial Intelligence, 2018.
- Mnih et al., Human-level Control through Deep Reinforcement Learning. Nature, 2015.
- Mnih et al., Asynchronous Methods for Deep Reinforcement Learning. Proceedings of the International Conference on Machine Learning, 2016.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 4.

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

Vedoucí práce: **Hradiš Michal, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 6. listopadu 2018

Abstrakt

Práce se zabývá učením neuronové sítě na jednoduchých úlohách v prostředí 3D střílečky Doom, zprostředkovaném výzkumnou platformou ViZDoom. Hlavním cílem je vytvoření agenta, který se učí na několika úlohách zároveň. Použitým algoritmem posilovaného učení je Rainbow, který kombinuje několik vylepšení algoritmu DQN. Pro učení na více úlohách jsem navrhnul a otestoval dvě různé architektury sítě. Jedna z nich byla úspěšná a po relativně krátké době trénování dokázal agent získat téměř 50 % z maximální možné odměny. Klíčovým prvkem úspěchu je Embedding vrstva pro parametrický popis prostředí jednotlivých úloh. Hlavním zjištěním je, že Rainbow je schopen učit se v 3D prostředí a s pomocí Embedding vrstvy i na více úlohách zároveň.

Abstract

Thesis deals with neural network learning on simple tasks in 3D shooter Doom, mediated by research platform ViZDoom. The main goal is to create an agent, which is able to learn multiple tasks simultaneously. Reinforcement learning algorithm used to achieve this goal is called Rainbow, which combines several improvements of DQN algorithm. I proposed and experimented with two different architectures of neural network for learning multiple tasks. One of them was successful and after a relatively short period of learning it reached almost 50% of maximum possible reward. The key element of this achievement is an Embedding layer for parametric description of task environment. The main discovery is, that Rainbow is able to learn in 3D environment and with the help of Embedding layer, it is able to learn on multiple tasks simultaneously.

Klíčová slova

neuronová síť, posilované učení, zpětnovazební učení, ViZDoom, přenesené učení, algoritmus Rainbow, PyTorch, Embedding vrstva

Keywords

neural network, reinforcement learning, ViZDoom, transfer learning, Rainbow algorithm, PyTorch, Embedding layer

Citace

BERÁNEK, Michal. *Posilované učení pro 3D hry.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Hradiš, Ph.D.

Posilované učení pro 3D hry

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.

.....
Michal Beránek
16. května 2019

Poděkování

Chtěl bych poděkovat panu Ing. Michalovi Hradišovi za vedení této bakalářské práce, odbornou pomoc a cenné rady při jejím vypracování.

Obsah

Kapitola 1

Úvod

Posilované učení je jednou z oblastí strojového učení, jedná se tedy o formu umělé inteligence. Cílem algoritmů posilovaného učení je naučit se, jak splnit zadaný, mnohdy komplexní, úkol. V kontextu počítačových her se jedná například o dokončení úrovně, dosažení co nejvyššího skóre či poražení protihráče. Algoritmy se učí na základě zpětné vazby v podobě odměn. Za správná rozhodnutí jsou odměňovány a za špatná rozhodnutí jsou trestány. Tímto způsobem posilujeme požadované chování.

Tato práce se věnuje metodám posilovaného učení používaným při trénování algoritmů na počítačových hrách. Jedná se především o hluboké posilované učení, jehož trénovanou složkou je hluboká neuronová síť. Cílem je pomocí vhodného algoritmu naučit neuronovou síť plnit jednoduché úlohy v prostředí 3D střílečky Doom z roku 1993. Je potřeba najít algoritmus, který se učí na jednotlivých úlohách při stejném nastavení hyperparametrů a navrhnout architekturu sítě, která bude schopná učit se na více úlohách zároveň.

V oblasti trénování neuronových sítí pro hraní počítačových her jsou nejznámějšími organizacemi OpenAI a DeepMind. OpenAI se podařilo naučit síť hrát hru Dota 2 a tým složený z těchto agentů porazil mistry světa[1]. DeepMind dokonce vytvořil agenta, který je schopen hrát 3D střílečku Quake III Arena (se zjednodušenou grafikou)[2]. Oba tyto projekty už nějakou dobu sleduji, což mě vedlo k výběru tohoto tématu.

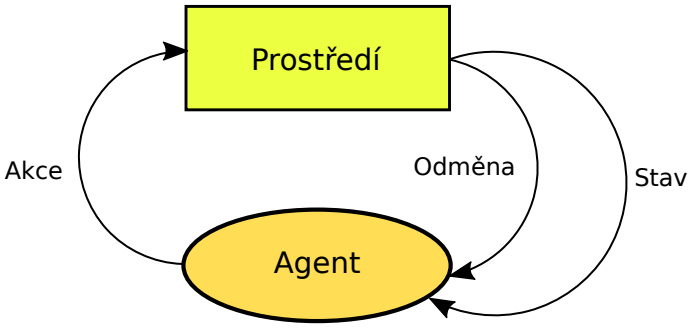
Úvodní část této práce se věnuje základním principům posilovaného učení a detailněji popisuje jeden z algoritmů hlubokého posilovaného učení — hluboké Q-učení. Dále se zabývá variantou hlubokého Q-učení, algoritmem Rainbow, který přichází s několika vylepšeními oproti původnímu algoritmu. Vylepšení jsou detailně popsána ve třetí kapitole. Ve čtvrté kapitole je popis trénovacího prostředí ViZDoom a konkrétní návrh agenta Rainbow pro experimenty. Je v ní charakterizována architektura sítě, zvolené hyperparametry algoritmu a úpravy původních trénovacích úloh. Zabývá se návrhem agenta pro učení na více úlohách. V poslední kapitole jsou výsledky provedených experimentů a jejich zhodnocení.

[1] Video: https://www.youtube.com/watch?v=yBEidvm_tZQ
[2] Detaily: https://deepmind.com/blog/capture-the-flag/

Kapitola 2

Posilované učení

Posilované učení se zabývá chováním agenta v prostředí. Prostředí poskytuje agentovi popis stavu. Agent na základě tohoto popisu a své strategie vybere jednu z možných akcí, kterou provede. Tím se prostředí dostane do nového stavu a agent dostane odměnu, která může být kladná, nulová nebo záporná. Agent vybere novou akci pro nový stav a celý cyklus se opakuje znovu. [13] Tento postup je znázorněn na obrázku 2.1. Cílem agenta je volit akce, které mu za jeho život přinesou největší kumulativní odměnu. Pokud následující stav závisí pouze na aktuálním stavu a zvolené akci, nikoli na minulosti, splňuje problém Markovskou vlastnost. V tom případě lze takové prostředí formulovat jako Markovův rozhodovací proces. [12]



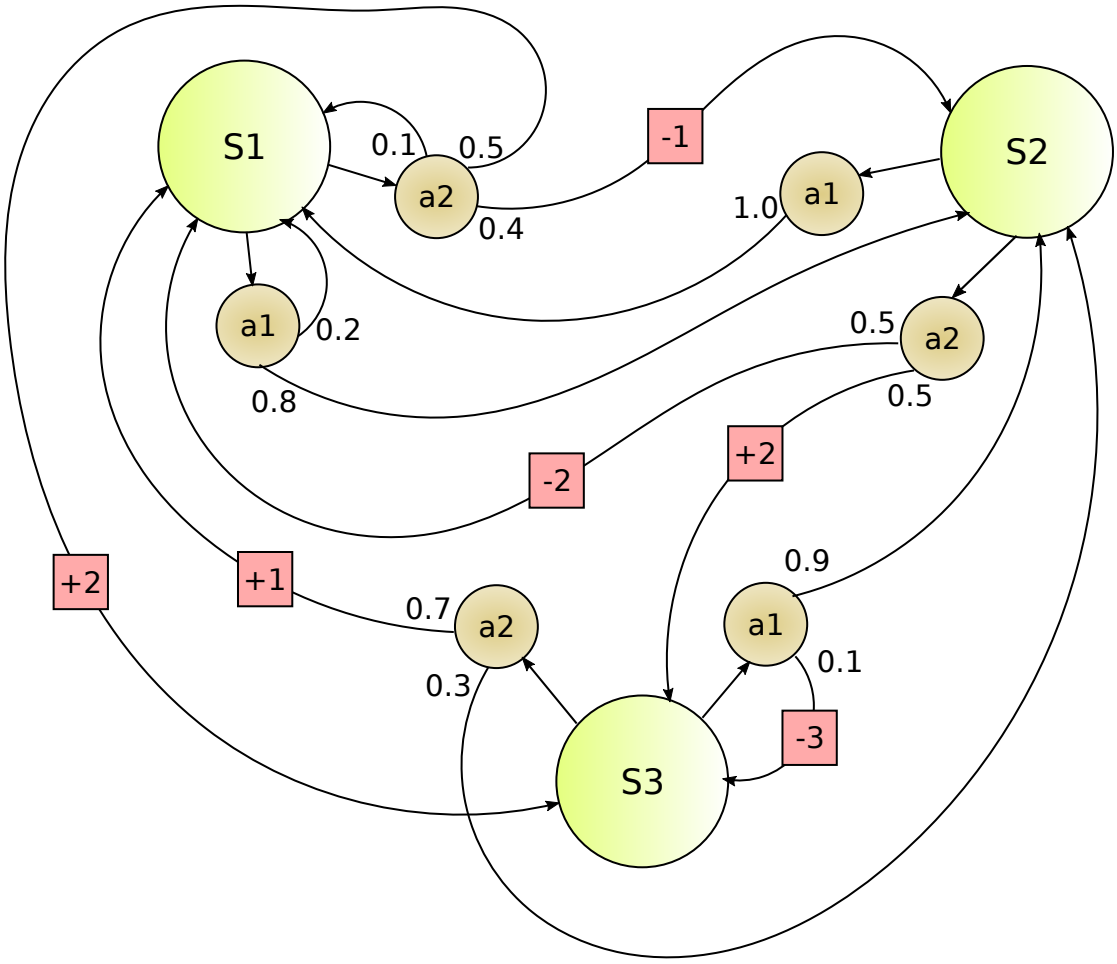
Obrázek 2.1: Princip posilovaného učení

2.1 Markovův rozhodovací proces

Markovův rozhodovací proces je diskrétní, stochaistický a kontrolovaný proces, který poskytuje matematický rámec pro modelování rozhodování v částečně náhodném prostředí. [27] Předpokládá existenci agenta v prostředí, který se nachází ve stavu s. V každém takovém stavu může agent provést jednu z dostupných akcí a, čímž přejde do nového náhodného stavu s'. Tato interakce je popsána přechodovou funkcí

$$P_a(s, s') = Pr(s_{t+1} = s' | s_t = s, a_t = a), \tag{2.1}$$

která určuje pravděpodobnost, se kterou se agent v čase $t+1$ dostane do stavu s' v případě, že v čase t ve stavu s provede akci a. Okamžitá odměna získaná při přechodu ze stavu s do stavu s' při provedení akce a je dána funkcí odměny $R(s, a, s')$. [27] Příklad Markovova rozhodovacího procesu je na obrázku 2.2.



Obrázek 2.2: Příklad Markovova rozhodovacího procesu. Stavy jsou reprezentovány žlutými kruhy, akce oranžovými kruhy a odměny za přechod červenými čtverci. Čísla reprezentují pravděpodobnost přechodu při zvolení příslušné akce.

Častým problémem prostředí je zpožděná odměna. [13] Může se stát, že agent provede skvělou akci, ale odměnu získá až po nějakém čase. Může nastat i opačný případ, kdy agent vybere akci, za kterou bude okamžitě odměněn, ale v budoucích stavech ho čekají vyšší záporné odměny, viz obrázek 2.3. Cílem je maximalizovat kumulativní odměnu, a proto je potřeba počítat i s budoucími odměnami. Celkovou odměnu jednoho běhu Markovova rozhodovacího procesu můžeme snadno vypočítat jako

$$R = r_1 + r_2 + r_3 + ... + r_n, \tag{2.2}$$

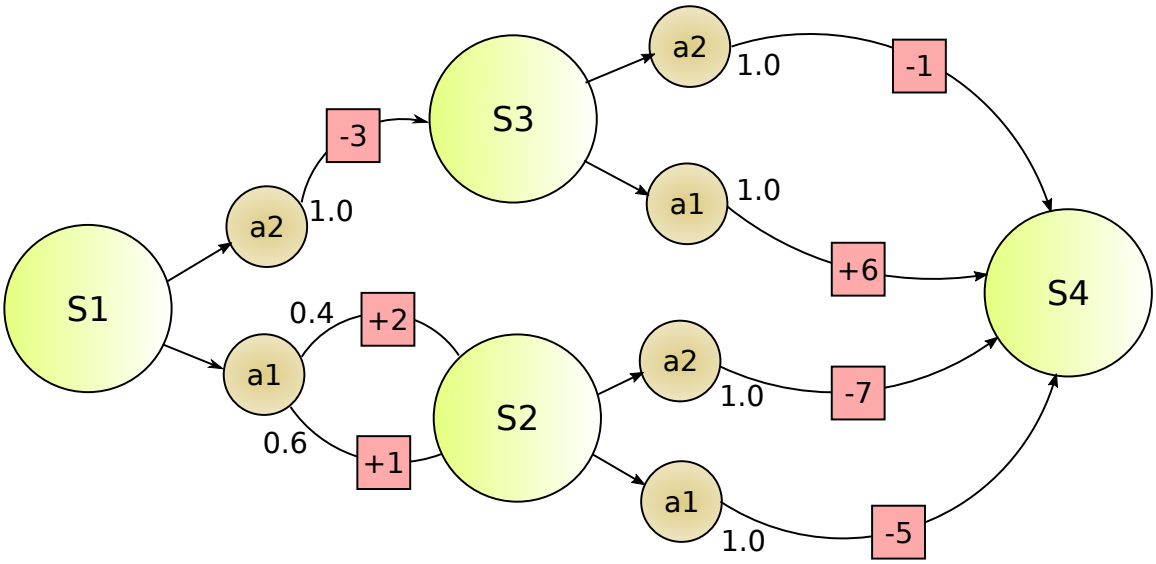
kde r_i je odměna získaná ve stavu i. Celkovou budoucí odměnu od času t do konce běhu můžeme vyjádřit jako

$$R_t = r_t + r_{t+1} + r_{t+2} + ... + r_n. \tag{2.3}$$

Ale protože se agent nachází ve stochaistickém prostředí, není jisté, že při stejné posloupnosti akcí dostaneme stejné odměny. Proto se zavádí faktor stárnutí γ, který snižuje váhu budoucích odměn. Výslednou rovnicí je

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + \gamma^{n-t} r_n, \tag{2.4}$$

kde γ je konstanta v rozmezí 0 — 1. Nastavíme-li faktor stárnutí blízko nule, bude se agent učit pouze z okamžité odměny. V případě deterministického prostředí je vhodné nastavit γ na hodnotu 1, neboť stejné akce vždy přinesou stejné odměny. [13] Nejčastěji se využívají hodnoty v rozmezí 0.8 — 0.99. [24]



Obrázek 2.3: Ukázka Markovova procesu s opožděnou odměnou. Jednotlivé prvky jsou shodné s obrázkem 2.2. Je vidět, že ve stavu S1 je z hlediska okamžité odměny provést akci a1. Tím se agent dostane do stavu S2, kde za všechny možné akce získá vysokou zápornou odměnu a maximální celková odměna je -3. Pokud ale agent zvolí ve stavu S1 akci a2, dostane sice zápornou odměnu, ale při další volbě má možnost získat odměnu 6. Maximální celková odměna je v tomto případě 3. Správnost rozhodnutí pro akci a2 ve stavu S1 se projeví až v budoucím stavu.

Markovův rozhodovací proces je pětice (S, A, P, R, γ), kde S je konečná množina stavů, A je konečná množina povolených akcí, P je přechodová funkce, R je funkce okamžitých odměn a γ je faktor stárnutí. [12] Teorie Markovových procesů nevyžaduje, aby množiny S a A byly konečné, ale většina základních algoritmů s nimi počítá jako s konečnými. [27]

2.2 Definice problému

Aby se agent mohl chovat optimálně, je potřeba definovat, jak optimální chování vypadá. Agent se chová podle strategie (policy) π, která udává rozložení pravděpodobnosti nad množinou dostupných akcí v daném stavu. Rovnice

$$\pi(a|s) = Pr(A_t = a|S_t = s) \tag{2.5}$$

vyjadřuje pravděpodobnost zvolení akce a z možných akcí A_t, za předpokladu, že se agent nachází ve stavu s z možných stavů S_t, při řízení strategií π. [1] Pro hledání optimální strategie se zavádí dvě nové funkce, které pomáhají interpretovat očekávanou hodnotu daného stavu. Jedná se o funkci ohodnocení stavu (Value-function) $V(s)$ a funkci ohodnocení akce (Q-function) $Q(s, a)$. [5] Hodnota stavu

$$V_\pi(s) = \mathbb{E}[R_t|s_t = s] \tag{2.6}$$

vyjadřuje celkovou očekávanou budoucí odměnu při rozhodování podle strategie π ze stavu s. Hodnota akce

$$Q_\pi(s,a) = \mathbb{E}[R_t|s_t = s, a_t = a] \tag{2.7}$$

vyjadřuje celkovou očekávanou hodnotu při provedení akce a ve stavu s a poté následování strategie π. [12]

Hledání optimální strategie lze přeformulovat na hledání takové strategie, která dosahuje maximální hodnoty stavu ze všech možných strategií pro stav s. Platí tedy [12]

$$V_*(s) = \max_\pi V_\pi(s), \tag{2.8}$$

kde $V_*(s)$ je optimální hodnota stavu s a π je jedna z možných strategií. Tuto rovnici lze přepsat jako Bellmanovu rovnici

$$V_*(s) = \max_a \sum_{s',r} P(s',r|s,a)[r + \gamma V_*(s')], \tag{2.9}$$

kde a je zvolená akce, r je okamžitá odměna, s' je budoucí stav, $P(s',r|s,a)$ je pravděpodobnost přechodu do stavu s' a γ je faktor stárnutí. Je vidět, že optimální hodnota stavu s závisí na okamžité odměně a optimální hodnotě následujícího stavu s'. Stejný princip lze aplikovat na funkci ohodnocení akce. Optimální hodnota akce je dána

$$Q_*(s,a) = \max_\pi Q_\pi(s,a), \tag{2.10}$$

tedy maximální hodnotou akce a ve stavu s které lze dosáhnout při řízení některou z dostupných strategií. Po rozložení na Bellmanovu rovnici vznikne vztah

$$Q_*(s,a) = \sum_{s',r} P(s',r|s,a)[r + \gamma \max_{a'} Q_*(s',a')], \tag{2.11}$$

kde a' je akce ve stavu s' a ostatní parametry jsou stejné jako v rovnici 2.9.

Hlavní myšlenkou základních algoritmů je pomocí Bellmanovy rovnice iterovat nad jednou z těchto funkcí, a postupně se přibližovat optimálním hodnotám. Mezi tyto algoritmy patří iterace hodnot, kde se iteruje nad funkcí V. Tato práce se věnuje primárně algoritmům vycházejícím z iterování nad funkcí Q, tedy Q-učení. [13]

Q-učení

Jednou z možností, jak implementovat Q-funkci, je formou tabulky, kde řádky reprezentují stavy a sloupce akce. [24] Prvním krokem je vytvoření tabulky Q-hodnot a svévolná inicializace hodnot. Tato tabulka poslouží agentovi jako referenční tabulka pro výběr nejlepší akce na základě Q-hodnoty. Agent se nachází v počátečním stavu s a vybere akci a, kterou provede. Získá odměnu r a dostane se do nového stavu s'. Aktualizuje Q-hodnotu právě provedené akce [10]

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)], \tag{2.12}$$

kde γ je faktor stárnutí a α je konstanta označující míru učení. Tato konstanta určuje, jak velký vliv bude mít nově získaná hodnota na starou hodnotu. V případě, že se $\alpha = 1$, bude aktualizovaná hodnota stejná jako v Bellmanově rovnici, čili vůbec nezáleží na původní

hodnotě. Nový stav je nastaven jako aktuální stav agenta, a celý proces se od výběru akce opakuje.

Otázkou zůstává jakým způsobem má agent akci vybrat. Jednou z možností je vždy vybrat akci, která má nejvyšší Q-hodnotu a jeví se jako nejlepší. Tomu se říká exploitace, tedy využití již získaných informací pro udělání rozhodnutí. Tento způsob prozkoumávání prostředí se nazývá „chamtivou“ (greedy) explorací. [13] Agent se totiž spokojí s první efektivní strategií, kterou nalezne, ta ale nemusí být optimální. Na druhou stranu pokud agent vybírá akce zcela náhodně, bude se zbytečně dostávat do „špatných“ stavů, o kterých už nemá význam se něco učit. Jednoduchým řešením je zavedení ϵ-chamtivé (ϵ-greedy) explorace. S pravděpodobností ϵ agent vybere náhodnou akci, jinak chamtivě zvolí akci s nejvyšší Q-hodnotou. Hodnoty v Q-tabulce se postupem času přibližují skutečným Q-hodnotám. Proto se nejčastěji používá ϵ inicializované na 1, které se postupem času snižuje na hodnotu blízkou 0. [19]

Výsledný algoritmus Q-učení je popsán v algoritmu 1.

Algoritmus 1: Pseudokód Q-učení

Libovolně inicializuj Q(s,a)
repeat pro každou epizodu:
  Inicializuj počáteční stav s
  repeat pro každý krok epizody:
    Vyber akci a pro stav s s pravděpodobností ϵ náhodně, jinak s nejvyšší Q-hodnotou
    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

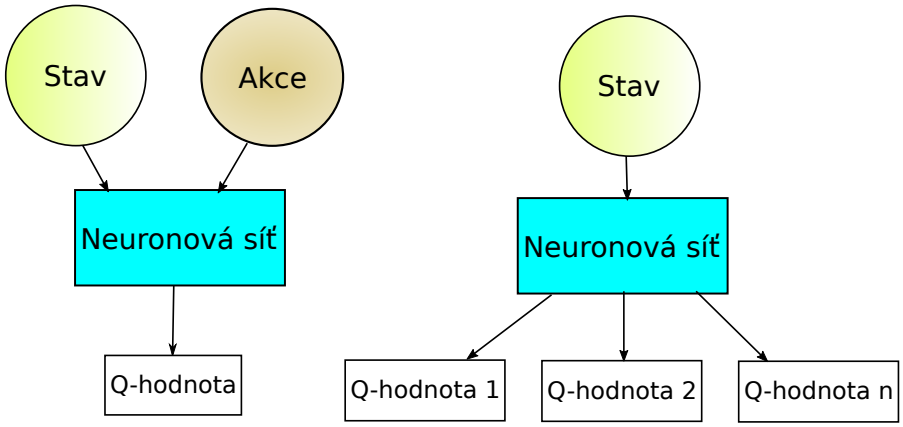
2.3 Hluboké Q-učení

Hlavním problémem Q-učení je nízká použitelnost pro komplexnější problémy. Pro obyčejné piškvorky v poli 3x3 existuje 19 683 různých stavů. Po odečtení nedosažitelných stavů je jich 5478. [26] Q-tabulka by tedy pro poměrně jednoduchý problém musela mít 5478 řádků. Tento způsob je pro počítačové hry, kde je stav popsán pixely na obrazovce, nepoužitelný. Konvergence tak velké tabulky by trvala extrémně dlouho. Z tohoto důvodu se pro odhad Q-hodnot používá hluboká neuronová síť — vzniká tak agent hluboké Q-sítě (dále jen DQN). [13]

Neuronová síť

Neuronová síť je sadou algoritmů modelovaných tak, aby zpracovávaly informace podobně jako lidský mozek. DQN využívá neuronovou síť pro odhad Q-hodnot. Ta je schopná rozpoznat vzorce ve vstupních datech, takže dokáže dobře odhadnout Q-hodnoty neznámých stavů, které jsou podobné jiným již známým stavům. Vstupem sítě by mohl být stav a akce a výstupem Q-hodnota dané akce. Efektivnějším přístupem je vzít jako vstup pouze stav a na výstupu získat Q-hodnoty všech možných akcí (viz obrázek 2.4). Tento přístup má vý-

hodu v tom, že pokud je potřeba provést aktualizaci Q-hodnot nebo vybrat akci s největší Q-hodnotou, stačí provést pouze jeden průchod neuronovou sítí. [13]



Obrázek 2.4: Nalevo je vidět méně efektivní využití neuronové sítě, kdy je pro určení Q-hodnoty každé akce potřeba jeden průchod sítí. Napravo je vidět efektivnější způsob, kdy pro všechny akce v daném stavu stačí pouze jeden průchod sítí.

V DQN algoritmu vytvořenén firmou Deepmind [16] byla použita architektura sítě znázorněná na obrázku 2.5. Vstupem sítě jsou 4 snímky obrazovky o velikosti 84x84 ve stupních šedi. Tento počet snímků je schopen zachytit pohyb, rychlost i zrychlení objektů. Obrázky projdou třemi konvolučními vrstvami, každá vrstva je následovaná aktivační funkcí ReLU [28], která na výstupy aplikuje nelinearitu. Poté jsou zpracovány dvěma plně propojenými vrstvami. Výstupem sítě je Q-hodnota pro každou akci. [13]

Neuronová síť se učí na základě zpětné propagace parametrů. Nejprve se spočítá chyba pomocí zvolené chybové funkce. Poté jsou váhy neuronové sítě zpětně upraveny optimalizačním algoritmem na základě vypočítané chyby. Pro trénování sítí ve hrách se často používá optimalizační algoritmus RMSProp. [15] Modernější alternativou je algoritmus Adam, který kombinuje výhody RMSProp s AdaGrad algoritmem a dosahuje dobrých výsledků rychle. [3] Mezi často používané chybové funkce patří střední kvadratická chyba (MSE), pro kterou platí

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(t_i - y_i)^2, \tag{2.13}$$

kde n je počet trénovacích vzorků, t_i je cílová hodnota a y_i je předpovězená hodnota i-tého vzorku. MSE zajímá pouze průměrná velikost chyby, nikoliv její směr. Odhady, které jsou vzdálené od skutečné hodnoty, jsou kvůli umocnění těžce penalizovány ve srovnání s málo odchýlenými odhady. Podobnou chybovou funkcí je střední absolutní chyba (MAE)

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|t_i - y_i|. \tag{2.14}$$

Význam proměnných je stejný jako v předchozí rovnici. Pro klasifikační problémy se nejčastěji používá logaritmická chyba (Cross Entropy Loss), která je dána vztahem

$$CEL = -(t_i \log(y_i) + (1 - t_i)\log(1 - y_i)). \tag{2.15}$$

Proměnné jsou stejné jako v předchozích rovnicích. Tato chybová funkce těžce penalizuje odhady, které jsou sebevědomé, ale špatné. [17]

Vzpomínková paměť

Pro rychlejší trénování DQN se používá metoda vzpomínkové paměti. V každém kroku se agentova vzpomínka $e_t = (s_t, a_t, r_t, s_{t+1})$ ukládá do vzpomínkové paměti. Učení pak probíhá tak, že se vybere dávka náhodných vzorků ze vzpomínkové paměti a provede se nad nimi aktualizace Q-hodnot. Tento přístup má několik výhod oproti normálnímu Q-učení [16]:

- každý krok vzpomínky může být použit pro více aktualizací vah, což umožňuje lepší efektivitu dat,
- vzorky nejsou po sobě jdoucí, není mezi nimi souvislost, což snižuje odchylku aktualizací,
- vzorky, na kterých se agent učí nezávisí na aktuálních parametrech sítě.

Cílová síť

Další modifikací DQN je využití samostatné neuronové sítě pro generování cílových hodnot y_i pro aktualizaci Q-hodnot. Po určitém počtu aktualizací se neuronová síť naklonuje na cílovou síť, která pak generuje cílové hodnoty. Tato modifikace zvyšuje stabilitu algoritmu, protože použití starších parametrů vytváří zpoždění mezi aktualizací aktivní sítě a aktualizací cílových hodnot, což výrazně snižuje šanci na divergenci a oscilaci parametrů. [16]

Algoritmus DQN s využitím vzpomínkové paměti a cílové sítě je popsán v algoritmu 2.

Algorithm 2: DQN algoritmus se vzpomínkovou pamětí, převzato a upraveno z [16]

Inicializuj vzpomínkovou paměť D
Inicializuj funkci ohodnocení akce Q s náhodnými váhami θ
Inicializuj cílovou funkci ohodnocení akce $\hat{Q}$ s váhami $\theta^- = \theta$
for *episode = 1, M* **do**
  Inicializuj počáteční stav $s1$
  for $t = 1, T$ **do**
    S pravděbodobností ϵ vyber náhodnou akci a_t
    jinak vyber $a_t = \text{argmax}_a Q(s_t, a; \theta)$
    Proveď akci a_t a pozoruj odměnu r_t a stav s_{t+1}
    Ulož přechod (s_t, a_t, r_t, s_{t+1}) do D
    Vezmi náhodnou dávku přechodů (s_j, a_j, r_j, s_{j+1}) z D
    Nastav $y_j = r_j$ pokud epizoda končí krokem $j + 1$
    jinak $y_j = r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \theta^-)$
    Proveď pokles podle gradientu na $(y_j - Q(s_j, a_j, \theta))^2$ vzhledem k síťovým parametrům θ
    Každých C kroků aktualizuj cílovou síť $\hat{Q} = Q$
  end
end

2.4 Alternativy k DQN

Současný pokrok lze rozdělit na dvě odvětví — vylepšení algoritmu DQN, kterým se věnuje další kapitola, a alternativní přístupy. Jednou z možností je neučit se funkci ohodnocení

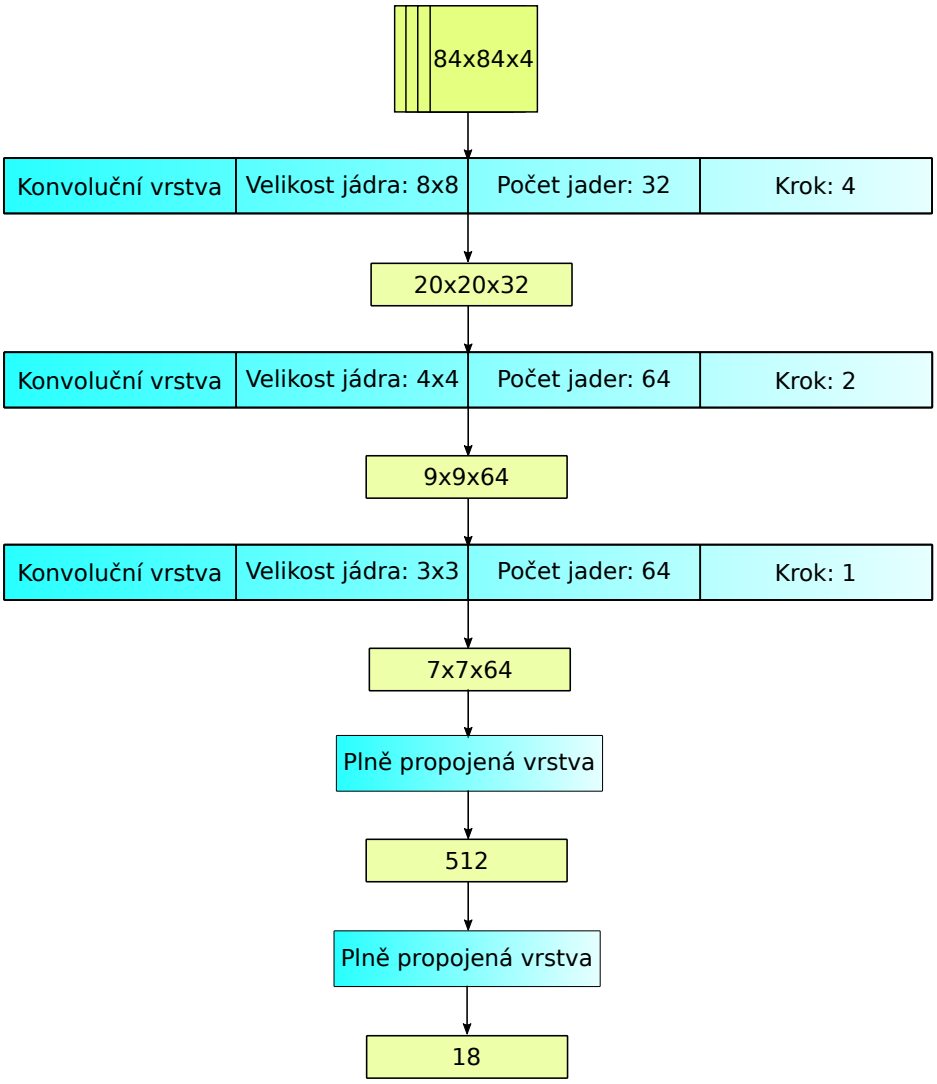
stavu nebo akce, ale učit se přímo funkci strategie (policy function), která mapuje stav na akci. Tato strategie může být buď deterministická, která pro vstupní stav vrací vždy jednu stejnou akci, nebo stochaistická, která pro vstupní stav vrací pravděpodobnost zvolení jednotlivých akcí. Většinou se využívá stochaistická strategie. Výhody těchto metod jsou: [21]

- mají lepší konvergenční vlastnosti,
- fungují lépe v prostředích, kde jsou akce spojité,
- umí se naučit stochaistické strategie bez implementování explorace/exploitace.

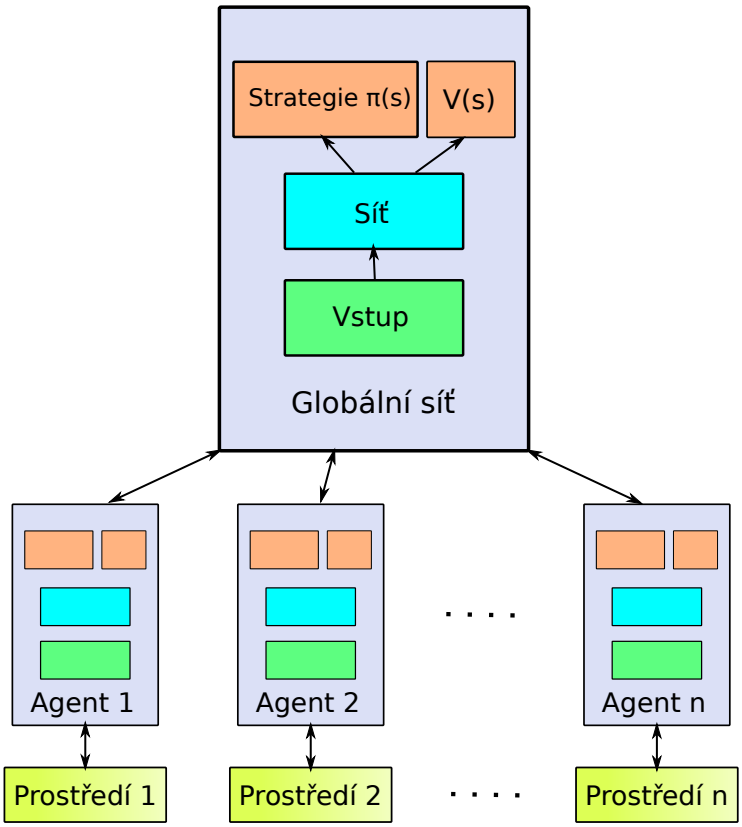
Hlavní nevýhodou je pomalejší konvergence ke globálnímu maximu.

Jedním z algoritmů využívajících tento princip, který dosahuje v oblasti her kvalitních výsledků, je A3C (Asynchronous Advantage Actor-Critic). [14] A3C kombinuje odhad funkce ohodnocení stavů $V(s)$ s přímou optimalizací strategie $\pi(s)$ pomocí gradientů (Policy Gradient). Poslední část neuronové sítě je složena ze dvou plně propojených vrstev, kde jedna dává na výstup hodnotu $V(s)$ (critic) a druhá strategii $\pi(s)$ (actor). Agent pak využije oba výstupy pro aktualizaci strategie. [9]

Hlavní výhodou tohoto algoritmu je rychlost. A3C využívá několik instancí agenta, které se trénujou paralelně. Každý z těchto agentů interaguje s vlastní kopií prostředí. Jejich zkušenosti jsou pak sdíleny prostřednictvím jedné globální sítě, viz obrázek 2.6. [9]



Obrázek 2.5: Architektura sítě DQN [16]. Žluté obdélníky reprezentují vstup/výstup, modré rámečky představují jednotlivé vrstvy neuronové sítě. První dva rozměry vstupů konvolučních vrstev vyjadřují počet pixelů, třetí je počet kanálů každého pixelu, který odpovídá počtu jader dané vrstvy. Krok (stride) vyjadřuje o kolik pixelů se konvoluční jádro posouvá. Výstup plně propojených vrstev odpovídá počtu jejich neuronů.



Obrázek 2.6: Vizualizace algoritmu A3C. Každý agent pracuje s vlastní kopií prostředí, vypočítává vlastní chybové funkce a získává z nich gradienty. Prostřednictvím gradientů aktualizuje globální síť a aktualizuje svoje váhy na váhy globální sítě.

Kapitola 3

Rainbow algoritmus

DQN a A3C jsou dnes považovány za základní a úspěšné algoritmy v oblasti počítačových her. Já jsem se rozhodl zaměřit na agenta DQN a jeho vylepšení. Algoritmus Rainbow [7] je agentem DQN se šesti vylepšeními, který dosáhl vysokých skóre v prostředí Atari her. V této kapitole jsou popsány principy a přínosy těchto rozšíření.

3.1 Dvojité DQN a více-krokové učení

Dvojité DQN a více-krokové učení jsou dvě drobná vylepšení původního DQN algoritmu. Cílem Dvojitého DQN [6] (Double DQN) je snížit nadhodnocování Q-hodnot potenciálních akcí. Tento jev vzniká kvůli použití stejných vah sítě pro výběr akce a ohodnocení této akce. Myšlenkou Dvojitého DQN je pro výpočet cílové Q-hodnoty využít dvě různé sady vah. Cílová hodnota pro DQN je dána vztahem

$$Q_t(s_t, a_t) = r_t + \gamma \max_a Q(s_{t+1}, a; \theta_t^-), \tag{3.1}$$

kde r_t je okamžitá odměna při provedení akce a_t ve stavu s_t, γ je faktor stárnutí a θ_t^- jsou váhy cílové sítě. Rovnici lze přepsat jako

$$Q_t(s_t, a_t) = r_t + \gamma Q(s_{t+1}, \operatorname*{argmax}_a Q(s_{t+1}, a; \theta_t^-); \theta_t^-). \tag{3.2}$$

Pro výběr akce i pro její ohodnocení jsou použity váhy cílové sítě θ_t^-. Ve Dvojitém DQN se pro výběr akce využije parametrů θ_t z primární sítě, výslednou rovnicí je pak [8]

$$Q_t(s_t, a_t) = r_t + \gamma Q(s_{t+1}, \operatorname*{argmax}_a Q(s_{t+1}, a; \theta_t); \theta_t^-). \tag{3.3}$$

Tato modifikace přinesla výrazné zlepšení při učení na Atari hrách s minimální výpočetní režií. [6]

Více-krokové učení

Metody učení, kterými se tato práce zabývá, využívají pro výpočet Q-hodnot či hodnot stavu ($V(s)$) pouze okamžitou odměnu a odhad hodnoty následujícího stavu (rovnice 2.9). Existují i metody Monte Carlo, které pro aktualizaci hodnoty stavu využívají diskontovaný součet všech budoucích odměn. [22] Více-krokové učení používá pro predikci cílových hodnot

n budoucích diskontovaných odměn definovaných jako

$$R_t^{(n)} \equiv \sum_{k=0}^{n-1} \gamma_t^{(k)} r_{t+k+1}, \tag{3.4}$$

kde γ je faktor stárnutí a r_t je okamžitá odměna ve stavu s_t. [7] Aktualizace paramterů DQN je pak dána zmenšováním alternativní chyby

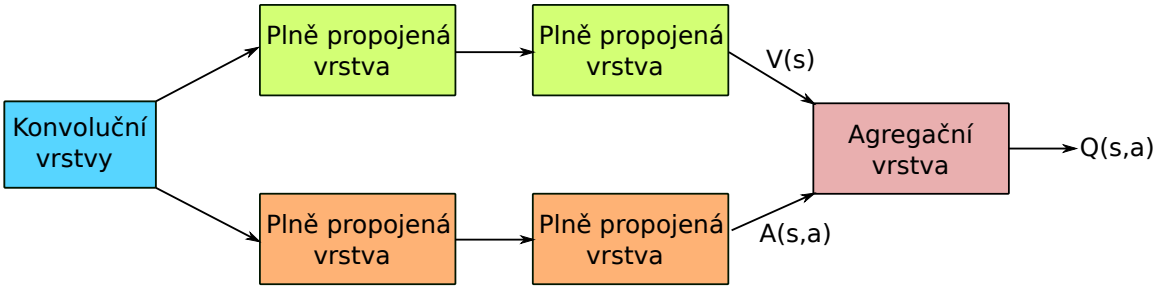
$$(R_t^{(n)} + \gamma_t^{(n)} \max_{a'} Q(s_{t+n}, a'; \theta^-) - Q(s_t, a_t; \theta))^2, \tag{3.5}$$

kde θ a θ^- jsou parametry primární a cílové sítě. Při vhodné volbě n vede tato modifikace ke zrychlení učení. [22]

3.2 Duální architektura neuronové sítě

Pokud se agent nachází ve stavech, ve kterých akce nemají na prosředí žádný relevantní vliv, je pro agenta zbytečné učit se efekt těchto akcí. V takových případech agentovi stačí naučit se hodnotu stavu. Duální architektura rozděluje výpočet funkce $Q(s, a)$ na výpočet hodnoty stavu $V(s)$ a funkci výhody $A(s, a)$, která udává, o kolik je lepší vybrat danou akci oproti ostatním v daném stavu. [8] Q-hodnota je kombinací těchto dvou funkcí, formálně

$$Q(s, a) = V(s) + A(s, a). \tag{3.6}$$



Obrázek 3.1: Duální architektura neuronové sítě. Výstup konvoluční části jde do dvou oddělených větví, které jsou složené z několika plně propojených vrstev. Výstupy větví se poté sečtou a dostaneme Q-hodnoty akcí.

Pro využití této modifikace je potřeba upravit architekturu neuronové sítě, viz obrázek 3.1. Konvoluční část zůstává stejná, ale následná sekvence plně propojených vrstev je rozdělena na dva proudy. Jeden z nich slouží pro výpočet hodnoty stavu a druhý pro výpočet výhody akce. Tyto dva proudy se spojí v poslední agregační vrstvě a výstupem je Q-hodnota. Výstup sítě je popsán rovnicí

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha), \tag{3.7}$$

kde θ jsou parametry společné části sítě, α parametry části sítě pro výpočet výhody akce a β parametry části sítě pro výpočet hodnoty stavu. Problémem této rovnice je, že na základě Q nelze jednoznačně určit V a A, což se projevuje slabým výkonem. [25] Řešením

tohoto problému je nastavit odhadování výhody tak, že zvolená akce bude mít výhodu 0. Rovnici 3.7 pak lze přepsat jako

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + (A(s,a;\theta,\alpha) - \max_{a'} A(s,a';\theta,\alpha)). \tag{3.8}$$

Pro Q hodnotu optimální akce a^* ve stavu s pak platí

$$Q(s,a^*;\theta,\alpha,\beta) = V(s;\theta,\beta). \tag{3.9}$$

To znamená, že jeden proud dodává odhady hodnot stavu, zatímco druhý produkuje odhady výhod akcí.

Alternativou je využít místo výhody optimální akce průměr výhod všech možných akcí [25]

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + (A(s,a;\theta,\alpha) - \frac{1}{|A|}\sum_{a'} A(s,a';\theta,\alpha)). \tag{3.10}$$

Ačkoli V a A ztrácí původní význam, protože jsou tyto hodnoty posunuté o konstantu, zvyšuje tento přístup stabilitu optimalizace. Rovnice 3.10 je použita v implementaci Rainbow agenta [7].

3.3 Prioritní vzpomínková paměť

Při použití obyčejné vzpomínkové paměti jsou vzpomínky při učení vzorkovány náhodně. Prioritní vzpomínková paměť zvyšuje pravděpodobnost vzorkování důležitějších přechodů. Ideálním měřítkem pro důležitost přechodu by bylo množství poznatků, které se agent může z daného přechodu naučit. [18] Tato informace není přímo k dispozici, proto se využívá velikosti rozdílu cílové Q-hodnoty a odhadnuté Q-hodnoty. Priorita se pak spolu s přechodem ukládá do paměti. Při vytváření dávky jsou vybírány přechody s největší prioritou. [20]

Chamtivé upřednostňování má několik problémů. Přechody, které po prvním průchodu mají nízkou prioritu budou vybrané až po dlouhé době, nebo nikdy (kvůli omezené kapacitě paměti budou dřív nahrazeny jiným přechodem). Přechody s vysokou prioritou budou naopak přehrávány mnohokrát, protože rozdíl mezi cílovou a odhadnutou hodnotou se snižuje pomalu. Agent se tak natrénuje jen na malém množství stavů. Proto se využívá stochaistické upřednostňování, které je kompromisem mezi chamtivým upřednostňováním a uniformně náhodným výběrem. [18] Pravděpodobnost výběru přechodu i je

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}, \tag{3.11}$$

kde $p_i > 0$ je priorita přechodu i a exponent α určuje míru upřednostnění.

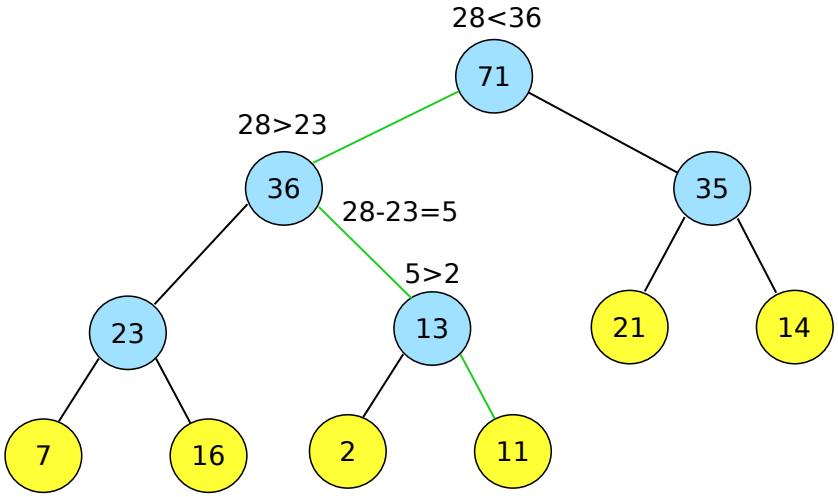
Prioritní paměť mění distribuci aktualizací na neuniformní a tím pádem mění řešení, ke kterému budou odhady konvergovat.[18] Tuto zaujatost k přechodům s vysokou prioritou lze napravit pomocí tzv. vah vzorkování důležitosti (importance-sampling weights)

$$w(i) = (\frac{1}{N} \cdot \frac{1}{P(i)})^\beta, \tag{3.12}$$

kde N je velikost paměti, $P(i)$ pravděpodobnost výběru přechodu i a β je exponent od 0 do 1, který kontroluje jak velký vliv tyto váhy mají na učení. [20] Velikost exponentu β koresponduje s hodnotou exponentu α použitém při výpočtu pravděpodobnosti výběru.

Čím je α větší, tím více jsou přechody upřednostňovány a roste potřeba korigovat zaujatost pomocí β. Nejčastěji se používá β s nastavenou počáteční hodnotou (například 0.4 [7]), která lineárně s průběhem učení roste k hodnotě 1. [18]

Pro efektivní implementaci prioritní vzpomínkové paměti se používá struktura binárního součtového stromu znázorněná na obrázku 3.2. Listy tohoto stromu jsou jednotlivé přechody s prioritou. Asymptotická složitost aktualizování a vzorkování je $O(\log N)$. [20]



Obrázek 3.2: Implementace prioritní vzpomínkové paměti pomocí struktury binárního součtového stromu. Listy (žluté kruhy) reprezentují přechody v paměti, čísla prioritu. Priorita rodičovského uzlu je dána součtem priorit dětí. Příklad výběru přechodu pro náhodnou hodnotu 28 je označen zelenými hranami. Pokud je hodnota větší než priorita levého následovníka, tak se priorita odečte a pokračuje se do pravého podstromu, jinak se pokračuje do levého podstromu.

3.4 Distribuční posilované učení

Myšlenkou distribučního posilovaného učení je učit se odhadovat distribuci návratu (odměn) místo očekávané odměny. Výstupem sítě pak nejsou Q-hodnoty pro jednotlivé akce, ale náhodná proměnná Z, která pro každou akci určuje distribuci možných návratů a její střední hodnota je rovna Q-hodnotě. [2] Funkci Z lze popsat distribuční Bellmanovou rovnicí ve tvaru

$$Z(s,a) \overset{\mathrm{D}}{=} r + \gamma Z(s',a'). \tag{3.13}$$

Distribuce $Z(s,a)$ tedy závisí na odměně r a distribuci $Z(s',a')$ budoucí akce a'.

Pro modelování těchto distribucí slouží diskrétní pravděpodobnostní funkce s parametry $N \in \mathbb{N}$ jakožto počtem *atomů* a $V_{min}, V_{max} \in \mathbb{R}$ označující minimální a maximální hodnotu očekávaného návratu. Nosičem této funkce je množina atomů z_i definovaných jako [7]

$$z_i = V_{min} + i\frac{V_{max} - V_{min}}{N-1} \tag{3.14}$$

pro $0 \leq i < N$. Q-hodnotu pak lze snadno získat rovnicí

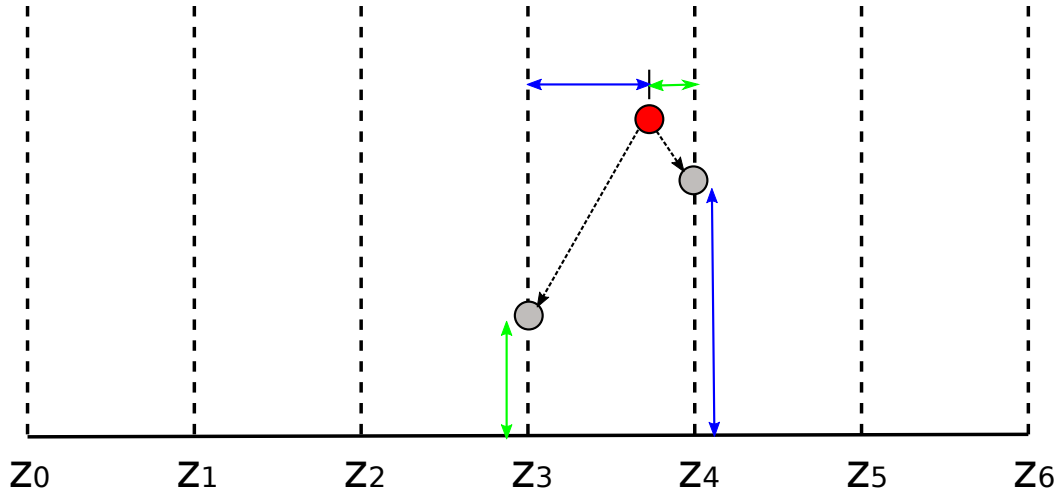
$$Q(s,a) = \mathbb{E}[Z(s,a)] = \sum_{i=0}^{N-1} p_i z_i, \tag{3.15}$$

kde p_i je pravděpodobnost, že se hodnota budoucího stavu bude rovnat z_i.

Pro optimální cílovou hodnotu funkce $Z^*(s, a)$ platí

$$Z^*(s, a) = r + \gamma Z(s', a^*), \tag{3.16}$$

kde a^* je optimální akce. [23] Pravděpodobnostní funkce $p = Z(s', a^*)$ je ve standardní formě, obsahuje tedy dvojice (z_0, p_0), (z_1, p_1),...(z_{n-1}, p_{n-1}). Po vynásobení jednotlivých atomů faktorem stárnutí γ a přičtení okamžité odměny r, se atomy transformují na $(r + \gamma\, z_0,\ p_0)$, $(r + \gamma z_1, p_1)$,...$(r + \gamma z_{N-1}, p_{N-1})$. Nyní jsou atomy vychýlené mimo původní pozice. Tento problém je vyřešen rozdělením výchyleného atomu na dva nejbližší zarovnané atomy a upravením jejich pravděpodobností vzhledem ke vzdálenosti od vychýleného atomu. Příklad je vidět na obrázku 3.3. Pokud byly vychýlené mimo interval $\langle V_{min}, V_{max}\rangle$, jsou nastaveny na hodnotu V_{min} pokud byly vychýleny do nižších hodnot, nebo V_{max} pokud byly vychýleny do vyšších hodnot.



Obrázek 3.3: Ukázka transformace vychýleného atomu na zarovnané atomy. Čím blíž je zarovnaný atom vychýlenému atomu, tím víc je jeho pravděpodobnost navýšena.

Pro výpočet chyby tohoto rozdělení pravděpodobnosti se používá logaritmická chybová funkce (Cross-Entropy loss). Celý postup je popsán v algoritmu 3.

3.5 Šumové sítě

Šumová síť (NoisyNet) je lineární vrstva neuronové sítě, která přináší alternativní přístup pro exploraci. [7] ϵ-chamtivá explorace dodává do strategie šum v každém kroku učení, který je ale zcela nezávislý na stavu. Cílem šumové sítě je upravit vektor vah tak, aby se strategie změnila v závislosti na stavu. Toho dosahuje pomocí parametrů šumu, které narušují váhy a zkreslení (bias) neuronové sítě. Tyto parametry se pak učí prostřednictvím sestupu gradientů. Efekt šumových sítí se nejvíce projevuje u úloh, kde první odměnu agent získá až po mnoha akcích. [4]

Lineární vrstva neuronové sítě s p vstupy a q výstupy je reprezentovaná rovnicí

$$y = wx + b, \tag{3.17}$$

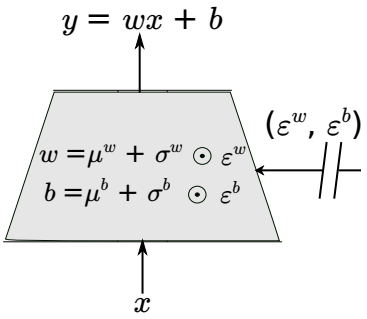
Algorithm 3: Algoritmus distribučního učení (převzato z [2])

Vstup : Přechod $s_t, a_t, r_t, s_{t+1}, \gamma_t \in [0, 1]$
$Q(s_{t+1}, a) = \sum_i z_i p_i(x_{t+1}, a)$
$a^* = \text{argmax}_a\, Q(x_{t+1}, a)$
$m_i = 0,\ i \in 0, ..., N-1$
for $j \in 0, ..., N-1$ **do**
`# Vypočítej projekci vychýlených atomů` $\hat{\mathcal{T}} z_j$ `na nosič` $\{z_i\}$
$\hat{\mathcal{T}} z_j = [r_t + \gamma_t z_j]_{V_{min}}^{V_{max}}$
$b_j = (\hat{\mathcal{T}} z_j - V_{min})/\Delta z$
$l = \lfloor b_j \rfloor,\ u = \lceil b_j \rceil$
`# Rozděl pravděpodobnosti na zarovnané atomy`
$m_l = m_l + p_j(x_{t+1}, a^*)(u - b_j)$
$m_u = m_u + p_j(x_{t+1}, a^*)(b_j - l)$
end
`# Vypočítej logaritmickou chybu`
Výstup: $-\sum_i m_i \log p_i(x_t, a_t)$

kde $x \in \mathbb{R}^p$ je vstupem vrstvy, $w \in \mathbb{R}^{q \times p}$ matice vah a $b \in \mathbb{R}^q$ zkreslení. Odpovídající šumová lineární vrstva je definována jako

$$y \overset{\text{def}}{=} (\mu^w + \sigma^w \odot \epsilon^w)x + \mu^b + \sigma^b \odot \epsilon^b, \tag{3.18}$$

kde parametry $\mu^w + \sigma^w \odot \epsilon^w$ a $\mu^b + \sigma^b \odot \epsilon^b$ nahrazují w a b. $\mu^w \in \mathbb{R}^{q \times p}$, $\mu^b \in \mathbb{R}^q$, $\sigma^w \in \mathbb{R}^{q \times p}$ a $\sigma^b \in \mathbb{R}^q$ jsou učené parametry, kdežto $\epsilon^w \in \mathbb{R}^{q \times p}$ a $\epsilon^b \in \mathbb{R}^q$ jsou náhodné proměnné šumu. Vizualizace šumové lineární vrstvy je znázorněna na obrázku 3.4. [4]



Obrázek 3.4: Vrstva šumové sítě se vstupem x, výstupem y a parametry šumu μ, σ a ϵ.

Existují dvě možnosti distribuce šumu. První z nich je nezávislý Gaussův šum, který používá jeden nezávislý šum pro každou váhu. Druhou možností je faktorizovaný Gaussův šum, který využívá jeden nezávislý šum pro každý výstup a jiný nezávislý šum pro každý vstup. Pro DQN se využívá druhá možnost, protože redukuje výpočetní čas generování náhodných čísel. Počáteční hodnoty parametrů μ a σ ve faktorizované šumové síti jsou následující. Každý prvek $\mu_{i,j}$ je vzorkován z náhodného uniformního rozložení $\mathcal{U}[-\frac{1}{\sqrt{p}}, +\frac{1}{\sqrt{p}}]$, kde p je počet vstupů lineární vrstvy a každý prvek $\sigma_{i,j}$ je nastaven na konstantu $\frac{\sigma_0}{\sqrt{p}}$. Hyperparametr σ_0 je nastaven na 0.5.

Kapitola 4

Návrh agenta

Algoritmus Rainbow dosáhl při hraní 2D her solidních výsledků. To ale neznamená, že se dopracuje podobných úspěchů i ve 3D hrách. Prostředí těchto her je podobnější reálnému světu a často se řídí podobnými fyzikálními zákony. Jedním z hlavních rozdílů pro posilované učení je, že agent vidí jen určitou výseč toho, co se kolem něj děje. V této kapitole navrhuji implementaci agenta Rainbow, se kterou budu dále experimentovat v komplexnějším 3D prostředí.

Pro provádění experimentů ve 3D hře jsem se rozhodl využít platfomu ViZDoom [29]. Toto prostředí dodává rozhraní pro vytváření umělé inteligence, jež hraje střílečku Doom na základě vizuální informace. Pro modelování a trénování neuronové sítě jsem použil knihovnu PyTorch[1]. Jako základ agenta Rainbow pro ViZDoom jsem použil implementaci pro Atari hry[2].

4.1 Prostředí ViZDoom

Protože se další část bude věnovat prostředí ze hry Doom, vysvětlím nyní základní principy této hry. Doom je počítačová hra z roku 1993, kde je hlavním úkolem hráče najít a stisknout tlačítko, které znamená úspěšné dokončení úrovně. V cestě mu brání několik různých druhů nepřátel, kteří hráči ubírají zdraví. Pokud zdraví hráče klesne na 0, hráč zemřel a musí danou úroveň začít znovu. Hráč může po nepřátelích střílet několika typy zbraní, které lze nalézt někde v úrovni. Každá zbraň má omezený počet nábojů. V úrovni se mimo zbraně a náboje nachází i další objekty, které hráči doplňují zdraví či brnění, které blokuje část zranění.

ViZDoom je platforma, která umožňuje vzájemnou komunikaci mezi hrou Doom a agentem. ViZDoom obsahuje několik přednastavených trénovacích úloh, které mají za úkol naučit agenta nějaký princip. Jeden průběh úlohy se nazývá epizoda. Prostředí na základě vybrané úlohy určuje, jaké akce může agent provádět, dává agentovi zpětnou vazbu na akci v podobě odměn a poskytuje agentovi popis stavu v zadaném formátu. Epizoda je ukončena pokud agent dosáhl maximálního počtu kroků, splnil zadaný cíl nebo jeho zdraví kleslo na 0. Prostředí očekává od agenta jako vstup akci v podobě vektoru „stisknutých tlačítek“. Agent tedy může v jednom kroku provést kombinovanou akci, například vystřelit a zároveň se pohnout doleva.

[1] https://pytorch.org/
[2] https://github.com/Kaixhin/Rainbow

Trénovací úlohy

V této části jsou popsány jednotlivé trénovací úlohy, se kterými se bude experimentovat.

Basic, Simpler basic a Rocket basic. Agent se nachází na jedné straně čtvercové místnosti. Na druhé straně se nachází neškodný a nepohyblivý nepřítel. Cílem agenta je co nejrychleji nepřítele jednou střelit. Agent se může pouze pohybovat doleva nebo doprava a střílet. V Simpler basic je zadní stěna výrazně světlejší než nepřítel a v Rocket basic má agent k dispozici místo pistole raketomet, jehož střely letí pomaleji.

Health gathering a Health gathering supreme. Zde se agent nachází v místnosti, kde agentovi neustále ubývá zdraví. Jeho cílem je přežít co nejdéle pomocí sbírání lékárniček, které se na zemi náhodně objevují. V Health gathering supreme je v místnosti několik stěn, což dělá přežití mnohem náročnější.

Defend the center. Agent se nachází uprostřed kruhové místnosti. Na obvodu se objevují nepřátelé, kteří se postupně k agentovi přibližují. Cílem je zabít co nejvíc nepřátel s omezeným počtem nábojů. Agent se nemůže pohybovat, může se pouze otáčet a střílet.

Defend the line. Tato úloha je podobná úloze Defend the center, ale probíhá ve čtvercové místnosti, kde se nepřátele objevují na druhé straně než agent. Dalším rozdílem je, že jsou nepřátelé postupně odolnější a je potřeba je střelit vícekrát.

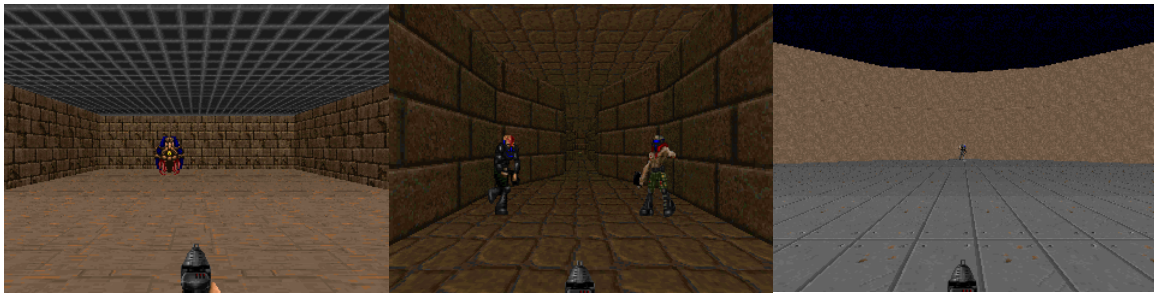
My way home. Úroveň se skládá z několika propojených místností a jedné chodby se slepým koncem. Každá místnost má jinou barvu a v jedné z nich se nachází zelené brnění. Cílem agenta je dojít k této vestě před skončením časového limitu.

Predict position. Agent se nachází ve čtvercové místnosti na jedné straně. Na druhé se vyskytuje nepřítel, který chodí zleva doprava. Cílem agenta je co nejdřív trefit nepřítele. K dispozici má jednu střelu z raketometu. Úloha končí po vypršení časového limitu nebo pokud raketa trefí nepřítele či stěnu.

Take cover. V této úloze je agent na jedné straně čtvercové místnosti. Na druhé straně jsou nepřátelé, kteří po agentovi vrhají ohnivé koule. Cílem agenta je se těmto koulím pomocí pohybu doleva a doprava vyhýbat a přežít co nejdéle.

Deadly corridor. Agent se nachází na začátku úzké chodby. Cílem je dostat se co nejblíže k brnění na druhém konci chodby. Po obou stranách této chodby je celkem 6 nepřátel. Pokud by agent nepřátele ignoroval a šel přímo k vestě, tak bude někde po cestě zabit.

Ukázky snímků obrazovky jsou k vidění na obrázku 4.1.



Obrázek 4.1: Ukázky snímků z trénovacích úloh. Jedná se o úlohy (zleva) Basic, Deadly corridor a Defend the center. Rozlišení snímků je 320x240.

Každá úloha má jinou maximální a minimální možnou celkovou odměnu. Abych mohl učení na různých úlohách rozumně porovnávat, upravil jsem u všech scénářů velikosti odměn, které agent získává. Většinu odměn lze snadno upravit v konfiguračních souborech trénovacích úloh. Některé odměny jsou zabudované přímo v souborech s mapou, které nelze

jednoduše upravovat. Tento problém jsem vyřešil tak, že jsem každé úloze přiřadil dělitele odměn. Pokud mohl agent původně získat prostřednictvím zabudovaných odměn například celkovou odměnu 2000 a není žádná další pozitivní odměna, dělitel této úlohy je 2000. Odměny v konfiguračních souborech jsem pak náležitě upravil. Minimální možná celková odměna v modifikovaných scénářích se pohybuje kolem -1 a maximální kolem +1. V některých případech může agent dosáhnout i odměn mimo tento interval, což ničemu nevadí a lze to interpretovat jako nečekaný úspěch či neúspěch. Upravené odměny a povolené akce v jednotlivých úlohách jsou shrnuty v tabulce 4.1.

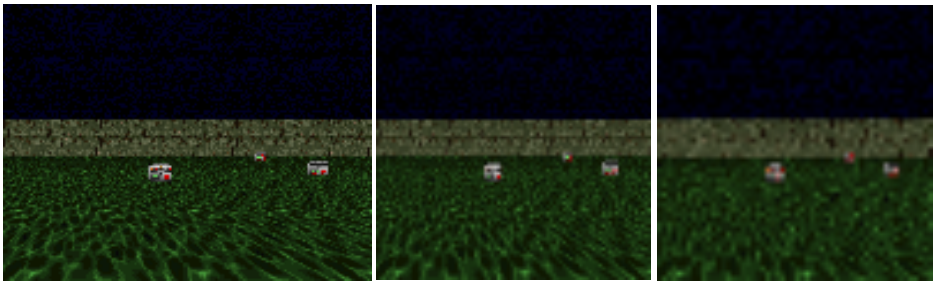
Název úlohy	Upravené odměny	Povolené akce
Basic	-0.0016 za každý krok +1 za trefení nepřítele -0.05 za střelu mimo	PL,PR,Ú
Health gathering	+0.001 za každý krok -1 za smrt	PF, OL, OP
Defend the center/line	+0.125 za zabití nepřítele -1 za smrt	OL, OR, Ú
My way home	-0.0005 za každý krok +1 za dosažení brnění	PF, OL, OR
Predict postition	-0.0033 za každý krok +1 za trefení nepřítele	OL, OR, Ú
Take cover	+0.001 za každý krok -1 za smrt	PL, PR
Deadly corridor	+0.0005 za přiblížení se k brnění -0.0005 za vzdálení se od brnění -1 za smrt	PL, PR, PB, PF, OL, OR, Ú

Tabulka 4.1: Tabulka upravených odměn a povolených akcí. Význam zkratek u povolených akcí: PL – pohyb doleva, PR – pohyb doprava, PB – pohyb dozadu, PF – pohyb dopředu, Ú – útok, OL – otočení vlevo, OP – otočení vpravo.

4.2 Architektura sítě

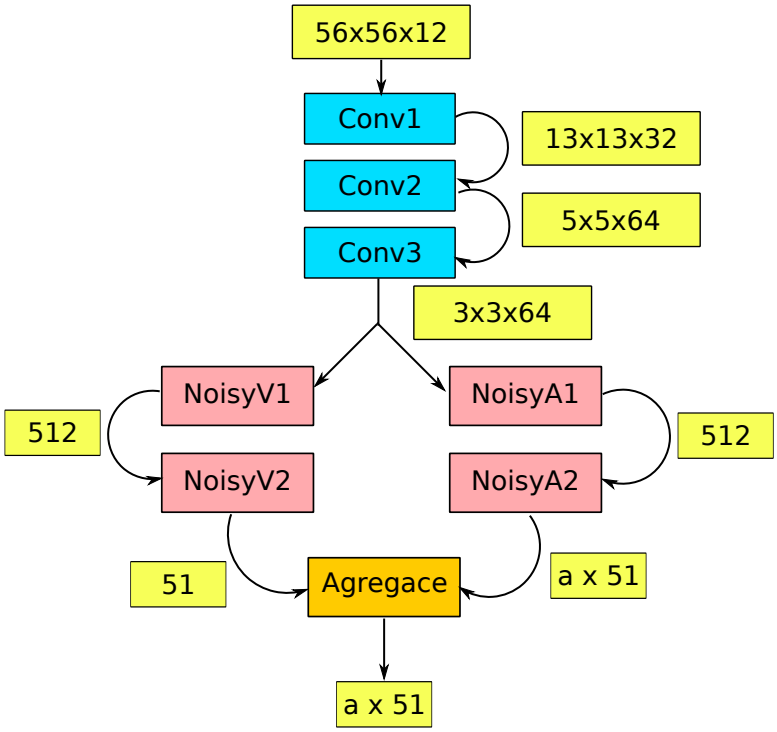
Základem neuronové sítě, kterou budu používat pro experimenty, je původní architektura sítě DQN. Vstupem sítě DQN jsou 4 obrázky o velikost 84x84 ve stupních šedi. V prostředí Doomu jsou objekty často výrazně barevně odlišeny. Kdybych používal šedotónový vstup tak se tato velice výhodná vlastnost ztratí. To může vést k pomalejšímu učení [11]. Proto budou vstupem mé sítě 4 barevné obrázky, které jsou výrazně výpočetně náročnější na zpracování, než šedotónové. Aby se doba zpracování vstupu příliš nezvýšila, zmenšil jsem rozlišení obrázků na 56x56. Na obrázku 4.2 je příklad vstupu v různých rozlišení. Lze postřehnout, že i drobné objekty jsou při rozlišení 56x56 stále viditelné, což je pro správné fungování neuronové sítě klíčové.

Konvoluční část neuronové sítě je složena ze tří konvolučních vrstev. První vrstva má 32 kanálů, další dvě 64, podobně jako u agenta DQN (viz obrázek 2.5). Vstupem první vrstvy jsou 4 barevné obrázky ve formátu RGB s rozlišením 56x56. Obrázky jsou nejprve normalizovány — hodnoty kanálů jednotlivých pixelů jsou vyděleny 255. Vrstvy používají jádra o velikosti 8×8, 4×4 a 3×3 s velikostí kroku (stride) 4, 2 a 1. Plně propojená



Obrázek 4.2: Porovnání rozlišení stejného snímku, s rozlišeními (zleva) 160 x 120, 84 x 84, 56 x 56.

část je rozdělena na dvě samostatné větve, jednu pro výpočet hodnoty stavu a druhou pro výhody akcí. Každá větev obsahuje jednu skrytou šumovou lineární vrstvu s 512 neurony. První větev vyprodukuje distribuci pravděpodobnosti hodnot stavů. Druhá větev vypočítá distribuce pravděpodobností výhod akcí. Výstupní šumová vrstva první větve má počet neuronů roven počtu atomů distribuce pravděpodobnosti, druhá má počet neuronů roven počtu možných akcí násobený počtem atomů. Výstupy jsou následně sečteny a normalizovány a výsledkem je distribuce pravděpodobnosti Q-hodnot pro každou akci. Vizualizace sítě je na obrázku 4.3



Obrázek 4.3: Vizualizace sítě Rainbow agenta použitého v experimentech. Vstupem sítě jsou 4 barevné snímky v rozlišení 56x56, výstupem je distribuční pravděpodobnost Q-hodnot pro každou akci. Žluté obdélníky označují velikost výstupu vrstev, a označuje počet povolených akcí. Modré obdélníky jsou konvoluční vrstvy, červené jsou plně propojené lineární vrstvy s šumem.

V implementaci využívám dvě modifikace, které ještě nebyly zmíněny. Na začátku učení se nejdříve naplňuje prioritní vzpomínková paměť a až pak se začne učit na vzorcích z paměti. Počet kroků před začátkem učení jsem nastavil na 20 000. Druhé malé vylepšení se odvíjí od toho, že hry jsou nastavené pro lidi, a člověk není schopen stisknout tlačítko na dobu jednoho snímku (většinou to ani není nijak výhodné). Proto je akce vybraná agentem několik snímků opakována. Experimentálně se ukázalo, že nejvýhodnější je opakovat akci po dobu čtyř až deseti snímků. [29]

U nastavení hyperparametrů algoritmu Rainbow jsem se řídil převážně doporučenými hodnotami, které byly použity při učení na Atari hrách. Jako optimalizační algoritmus jsem zvolil Adam, který je méně náchylný k volbě míry učení (learning rate).[7] Hodnoty hyperparametrů lze vidět v tabulce 4.2.

Parametr	Hodnota
Počet kroků před začátkem učení	20 000 snímků
Míra učení algoritmu Adam	0.0000625
Počáteční hodnota šumu σ_0	0.1
Interval aktualizace cílové sítě	20 000 snímků
Parametr ϵ algoritmu Adam	0.00015
Exponent prioritní paměti α	0.5
Exponent vah vzorkování důležitosti β	$0.4 \rightarrow 1$
Počet kroků více-krokového učení n	3
Počet distribučních atomů z	51
Distribuční hodnoty V_{min} a V_{max}	[-10, 10]
Faktor stárnutí γ	0.99
Velikost dávky	32
Velikost paměti	300 000
Interval učení z paměti	4 kroky
Opakování akcí	4 snímky

Tabulka 4.2: Tabulka hyperparametrů

4.3 Učení na více úlohách

Trénovací úlohy mají mnoho společných prvků, protože jsou z prostředí jedné hry. Nepřátelé vypadají stejně, akce útok znamená to samé ve všech úlohách, pokud agent trefí nepřítele, tak zpravidla dostane odměnu. Agent by těchto informací mohl využít a při vhodném návrhu by se mohl naučit plnit několik různých úloh. Cílem této části je takového agenta navrhnout.

Nejdříve je potřeba zařídit, aby prostředí předkládalo agentovi různé úlohy. Při učení na jedné úloze po konci epizody ihned začala nová. V případě více úloh se po skončení epizody náhodně vybere jedna z trénovacích úloh a načte se příslušný konfigurační soubor, obsahující informace o mapě a odměnách, a zahájí se nová epizoda. Vzpomínky ze všech úloh se ukládají do jedné společné paměti. Díky tomu jedna dávka pro učení obsahuje vzpomínky z různých trénovacích úloh.

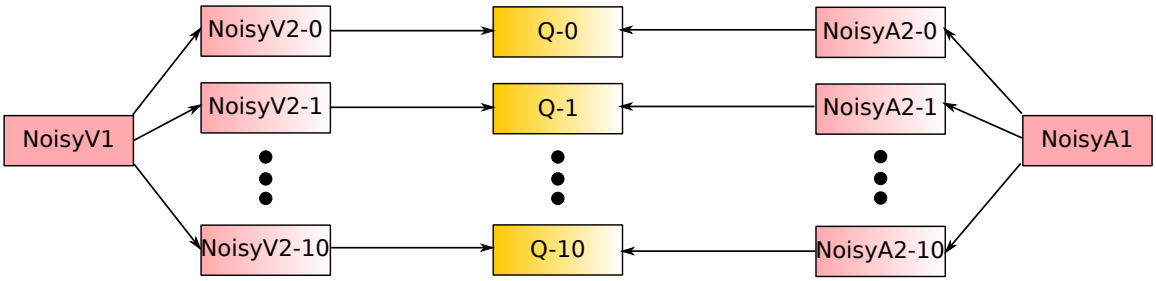
Důležitou informací pro takového agenta je, z jaké úlohy vstupní stav pochází. Bude-li agent disponovat touto informací, umožní mu to lépe určit kvality akcí v tomto stavu. Přechod pak bude kromě aktuálního stavu, provedené akce, odměny a následujícího stavu

obsahovat i hodnotu pro identifikaci úlohy. Upravený přechod je stejně jako původní ukládán do paměti a vzorkován při učení.

Dále je potřeba upravit neuronovou síť tak, aby pro učení efektivně využívala identifikátor úlohy. Úpravy se opírají o myšlenku, že nižší vrstvy neuronových sítí se učí primárně obecnější principy prostředí a na konkrétní úkol se trénují hlavně vyšší, většinou plně propojené vrstvy. První možností jak do sítě dodat identifikátor úlohy, je přidat vstupnímu snímku jeden kanál, který přiřadí ke každému pixelu z jaké úlohy pochází. Tento způsob nedává příliš smysl, protože přímo upravuje vstupní snímek, což může vést k neočekávanému chování. Navíc nemá smysl dodávat informaci o úloze v nižších vrstvách sítě, lze ji přidat až u vyšších.

Návrh 1

Rozumnější možností je vypočítat Q-hodnoty akcí daného stavu pro všechny úlohy, a poté na základě identifikátoru úlohy vybrat Q-hodnoty odpovídající dané úloze. Poslední vrstva je rozdělena tak, aby každá úloha měla vlastní plně propojenou vrstvu, jejímž výstupem jsou Q-hodnoty akcí pro odpovídající úlohu. Rainbow využívá duální architektury, takže pro každou úlohu existují ve skutečnosti dvě vrstvy, jedna pro výpočet hodnoty stavu a druhá pro produkování výhody akcí, které se následně sečtou. Z těchto sad Q-hodnot se pak vybere sada odpovídající dané úloze. Poslední vrstvy se učí pouze na konkrétní úkol, zatímco ostatní se trénují na všechny úlohy. Vizualizace této architektury je na obrázku 4.4.

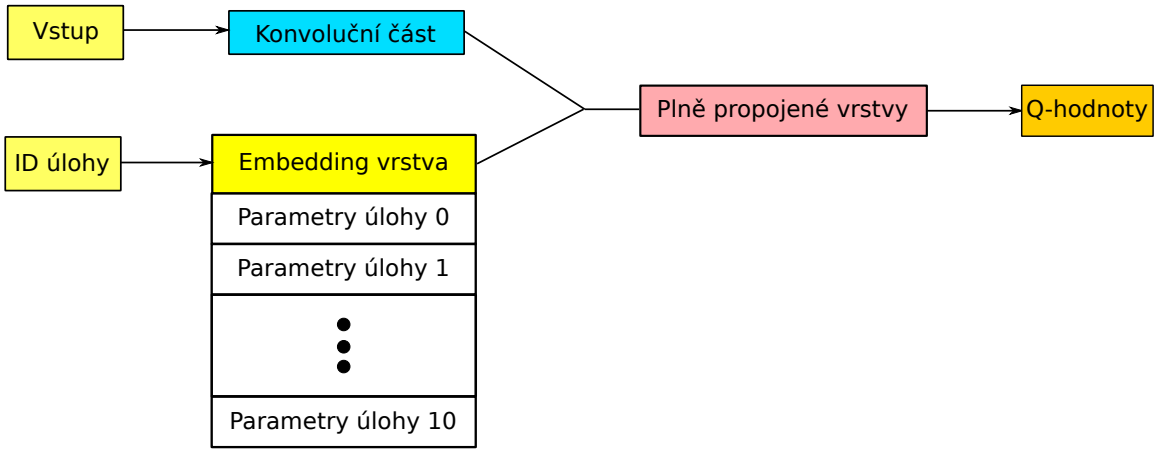


Obrázek 4.4: První návrh architektury sítě pro učení na jedenácti úlohách. Pro vstup se vygeneruje sada Q-hodnot pro každou úlohu. Na základě identifikátoru úlohy pak síť vybere vhodnou Q-hodnotu, kterou pošle na výstup.

Návrh 2

Druhý způsob, který jsem navrhnul, je použití Embedding vrstvy pro parametrický popis vlastností trénovací úlohy. Embedding vrstva je jednoduchá vyhledávací tabulka, která v tomto případě pro zadaný identifikátor úlohy vrátí vektor vlastností prostředí. Tento vektor konkatenuji ke vstupu prvních plně propojených vrstev. Embedding vrstva se stejně jako zbytek sítě učí na základě gradientů, počáteční hodnoty jsou inicializovány náhodně. Počet řádků Embedding vrstvy odpovídá počtu scénářů. Jako počet parametrů pro popis úlohy jsem zvolil 128. Vizualizace tohoto návrhu je na obrázku 4.5

Ke správnému fungování výše zmíněných architektur je potřeba, aby všechny úlohy měly stejný počet akcí, jinak by vrstvy úloh s větším počtem akcí počítaly Q-hodnoty neexistujících akcí. Agent by se pak takové akce snažil provést, což by vedlo k chybě. Vymyslel jsem dva způsoby, jak tento problém vyřešit. Pro oba platí, že počet možných



Obrázek 4.5: Druhý návrh architektury sítě pro učení na jedenácti úlohách. Vstupní snímky jsou zpracovány konvoluční částí a výstupu je připojen parametrický popis úlohy získaný z embedding vrstvy. Tento vektor je zpracován plně propojenými vrstvami a výstupem jsou Q-hodnoty pro danou úlohu.

akcí všech úloh bude roven maximálnímu počtu povolených akcí. První možností je nahradit nepovolené akce prázdnou akcí. Provedením této akce agent v prostředí nic neudělá. To je zpravidla z hlediska odměny nevýhodné, a tak se rychle naučí tyto akce nevybírat. Druhou možností je přemapovat nepovolené akce na jiné povolené akce. Akce otočení doleva by v jiné úloze znamenala například akci pohyb vpravo. Tato změna významu akce by ale mohla mít negativní dopad na učení více úloh. Některé akce by navíc byly mapovány vícekrát, než jiné, takže by je agent ze začátku učení prováděl častěji. Proto jsem se rozhodl použít pro experimenty první možnost.

U některých úloh lze povolit i akce, které nejsou přístupné v původním nastavení úloh, aniž by se výrazně změnil princip trénovací úlohy. Jedná se o úlohy, kde se agent pohybuje volně v prostoru, tedy Health gathering, Health gathering supreme a My way home. V těchto úlohách jsem dovolil agentovi kromě pohybu vpřed a otáčení i pohyby doleva, doprava a dozadu. To by mělo vést k lepšímu celkovému pochopení významu zmíněných akcí. Prostředí navíc umožňuje simulovat pohyb myši pomocí otočení o více stupňů. Do úloh, kde se agent může otáčet, jsem přidal akci otočení doleva o 10 stupňů a akci otočení doprava o 10 stupňů. Protože se akce čtyřikrát opakuje, dojde k otočení o 40 stupňů. Tato úprava by měla dovolit agentovi rychleji reagovat na situaci a tím zlepšit jeho výkon. Celkově má agent na výběr z devíti akcí.

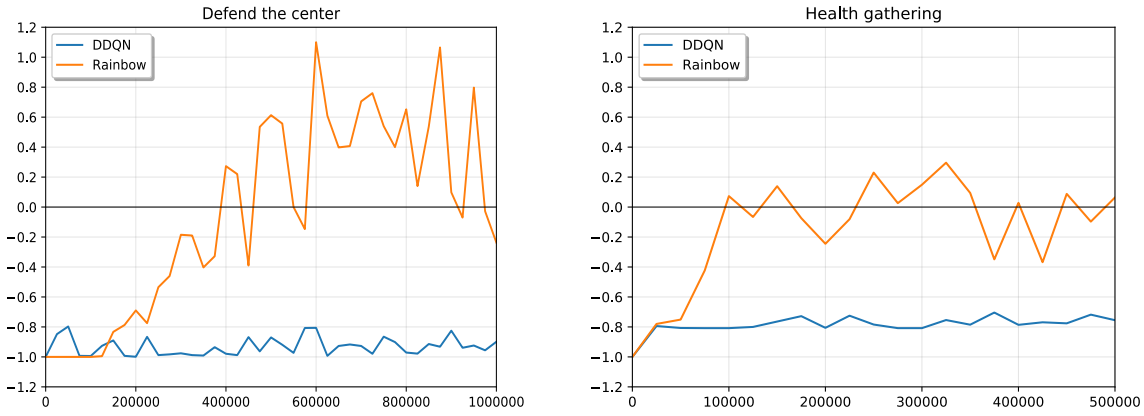
Kapitola 5

Experimenty

V experimentech testuji výkon agenta Rainbow navrženého v předchozí kapitole na trénovacích úlohách. Hlavním měřítkem jsou odměny, kterých agent dosahuje po určité době. Kvůli výpočetní a časové náročnosti byly experimenty provedeny pouze jednou. Pro trénování agentů jsem využil vlastní počítač a cloud v metacentru firmy Google[1].

5.1 Porovnání Rainbow s DQN

V experimentu srovnávám rychlost učení agenta Rainbow s implementací Duálního DQN[2]. Porovnání jsem provedl na dvou trénovacích úlohách, konkrétně Health gathering a Defend the center. Používám vlastní verze těchto úloh, tedy s upravenými odměnami a akcemi. Počet trénovacích kroků jsem nastavil na 500 000 pro Health gathering a 1 000 000 pro Defend the center. Po každých 25 000 krocích jsou modely testovány na 100 epizodách dané úlohy. Celkové odměny ze všech testovacích epizod jsou zprůměrovány a promítnuty do grafů na obrázku 5.1.



Obrázek 5.1: Porovnání učení agenta Rainbow s Dueling DQN. Osa x označuje počet kroků agenta a hodnoty na ose y označují průmer odměn získaných ve 100 testovacích epizodách.

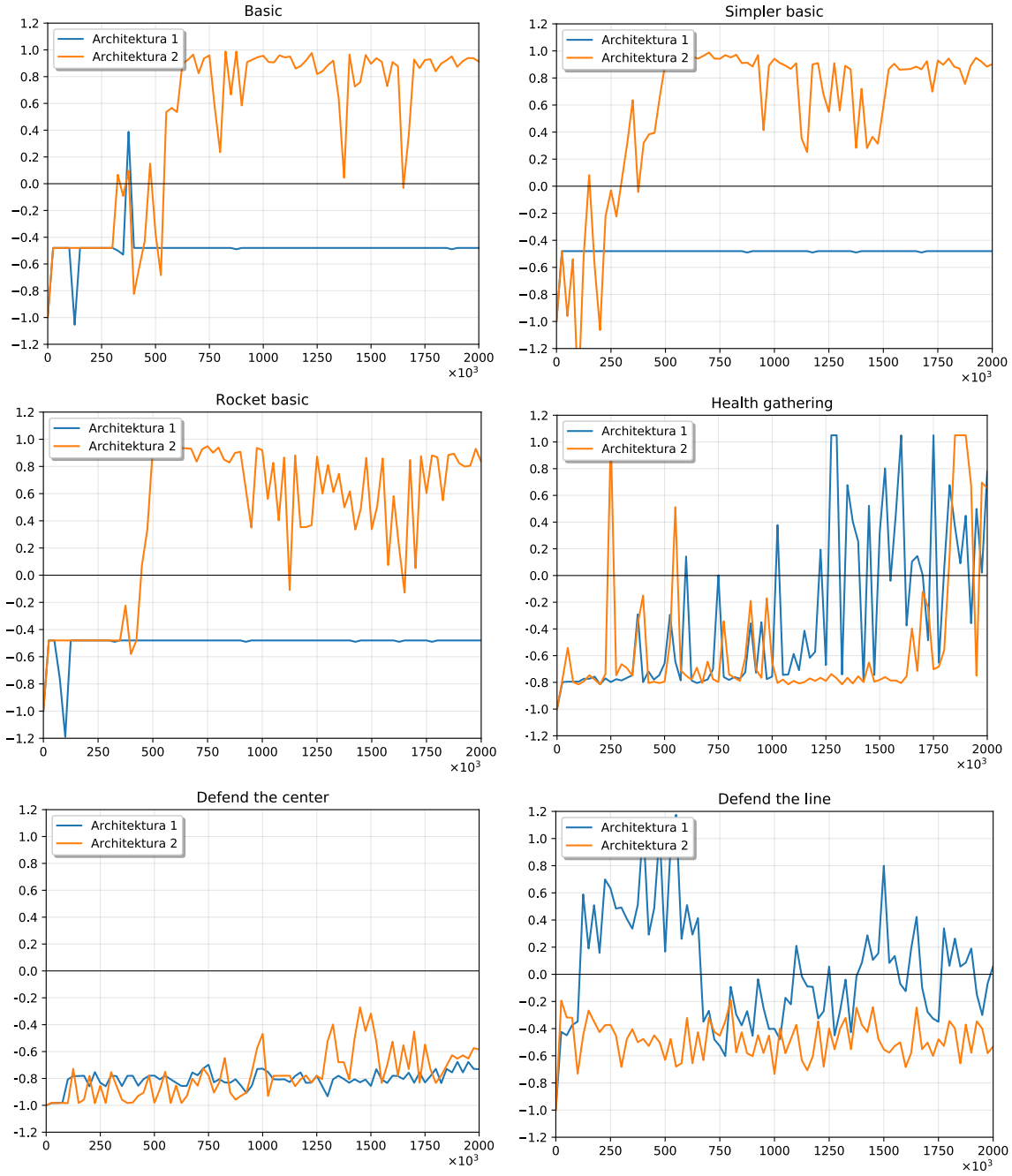
Z výsledků lze usoudit, že vylepšení agenta Rainbow mají na výkon při učení na těchto úlohách velmi pozitivní vliv. Agent DDQN se za prvních 1 000 000 kroků učení v úloze

[1] https://console.cloud.google.com/
[2] https://github.com/mihahauke/deep_rl_vizdoom

Defend the center nedostal přes průměrnou hodnotu -0.8. U agenta Rainbow můžeme pozorovat i případy, kdy dosáhl průměrného skóre přes 1, které odpovídá zabití šestnácti nepřátel. Podobně je tomu i u druhé trénovací úlohy. Už po 100 000 krocích se agent naučil strategii, která mu přinesla průměrné odměny větší než 0. To znamená, že v některých epizodách přežil až do konce časového limitu a nezískal zápornou odměnu za smrt, tedy splnil úkol. Lze předpokládat, že při delším trénování by agentovo průměrné skóre rostlo.

Důvodů slabého výkonu DDQN oproti Rainbow může být mnoho. Jako optimalizační algoritmus je použit RMSProp, který konverguje pomaleji než Adam, což mohlo ovlivnit rychlost učení. Další možností je, že ϵ-chamtivá strategie není pro tento typ úkolů optimální. Pokud agent dělá akce zcela náhodně a provede pohyb dopředu a pohyb dozadu, dostane se do stavu, ve kterém byl, aniž by se něco naučil. V původní nemodifikované úloze se mohl agent pouze pohybovat dopředu a otáčet, takže taková situace nemohla snadno nastat. Poslední možný důvod jenž zmíním, je nevhodné nastavení hyperparametrů.

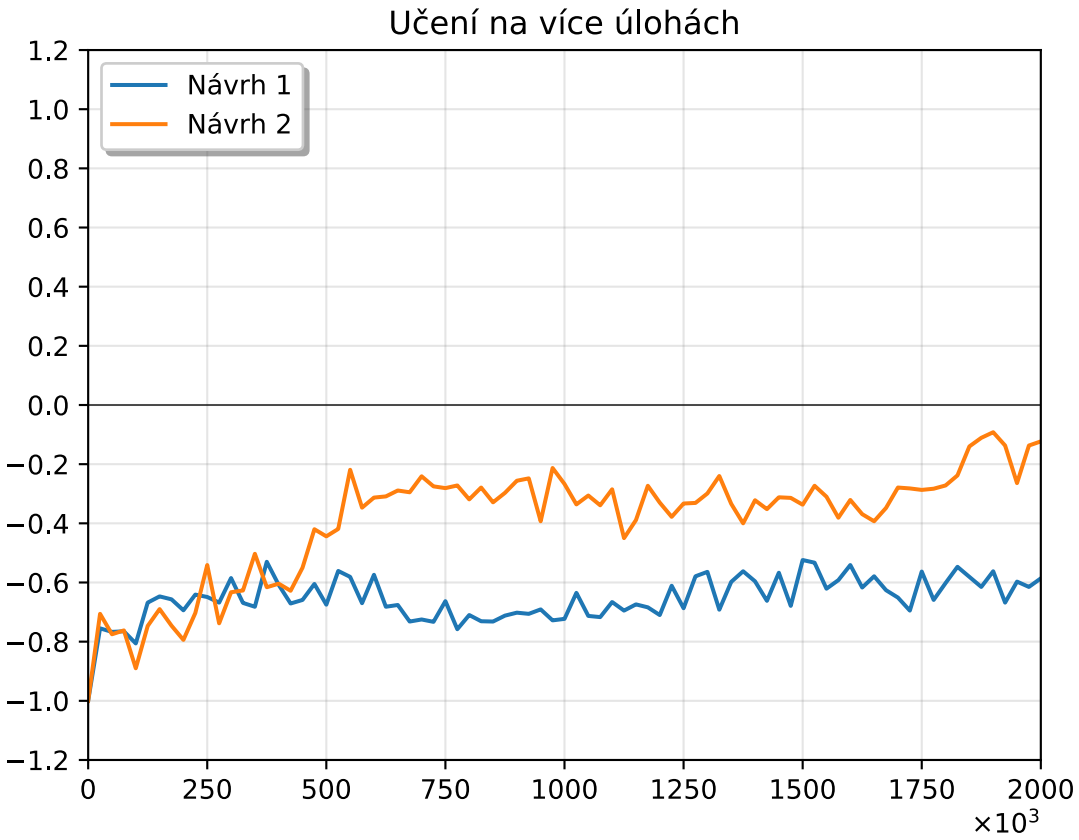
Nejdůležitější informací, která z tohoto experimentu vyplývá je, že se agent Rainbow učí na jednotlivých úlohách se stejným nastavením hyperparametrů. V obou úlohách agent hojně využívá akci otáčení o více stupňů, což potrvzuje moji hypotézu, že přidání těchto akcí umožní agentovi získávat vyšší odměny.



Obrázek 5.2: Srovnání výkonu architektur v jednotlivých úlohách. Osa x označuje počet kroků v tisících.

5.2 Experimenty na více úlohách

V této části experimentuji s agenty, kteří se učí na všech jedenácti úlohách zároveň. Pozoruji nejen jejich celkový výkon, ale i rychlost učení na jednotlivých úlohách. Počet kroků agenta je 2 000 000, testování modelů probíhá na pěti epizodách z každé úlohy, celkový počet testovaných epizod při každém vyhodnocení je 55. V grafu na obrázku 5.3 je srovnání učení dvou navrhnutých architektur pro učení na více úlohách.



Obrázek 5.3: Porovnání architektur agenta Rainbow pro učení na více scénářích. Osa x odpovídá počtu kroků agenta v tisících, osa y průměrnou odměnu získanou při testování na pěti epizodách z každé úlohy.

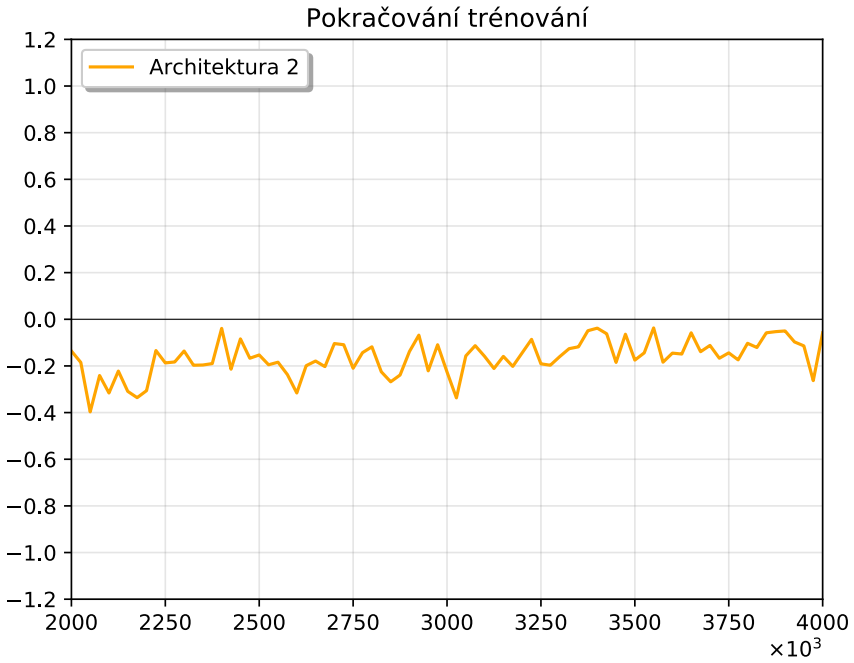
Z grafu lze vypozorovat, že agent využívající Embedding vrstvu se celkově učí výrazně rychleji než architektura s rozvětvenou poslední vrstvou. Pro lepší pochopení rozdílného výkonu je v grafech na obrázku 5.2 porovnání výsledků ve vybraných úlohách.

Hlavním zdrojem rozdílných výsledků mezi agenty byl scénář Basic a jeho alternativy. Agent 2 se po 250 000 krocích začal zlepšovat v úloze Simpler basic. Poznatky získané v této úloze mu pomohly získávat vyšší odměny i v Rocket basic, kde vidíme prudký nárůst získaných odměn kolem 500 000 kroků. Hned poté se agent zlepšil i v úloze Basic. Je vidět, že sdílení informací z jedné úlohy v tomto případě pomohlo agentovi k zlepšení na jiné úloze. Agent 1 se na tento typ úloh naučil strategii „nic nedělej“, protože za každý výstřel mimo je penalizován. To vede ke konstatntní odměně -0.48.

Agent 1 se naučil strategie v úlohách Defend the line a Health gathering, které mu přinášely vyšší průměrnou odměnu. Agent 2 se ke konci trénování naučil plnit úlohu Health

gathering, což vedlo ke vzrůstu celkových průměrných odměn. V úloze Defend the center byl o trochu úspěšnější agent 2. Ostatní úlohy byly komplexnější, takže se v nich agenti příliš neučili, ale z pohledu dosažených odměn byl úspěšnější agent 2.

Rozhodl jsem se učit model agenta 2 ještě dalších 2 000 000 kroků, abych zjistil, jestli se bude agent dále zlepšovat. Průběh učení je zachycen v grafu na obrázku 5.4. Agent se mírně zlepšil, několikrát dosáhl průměrné odměny -0.05. Největší pokrok nastal v úlohách Defend the center a Defend the line.



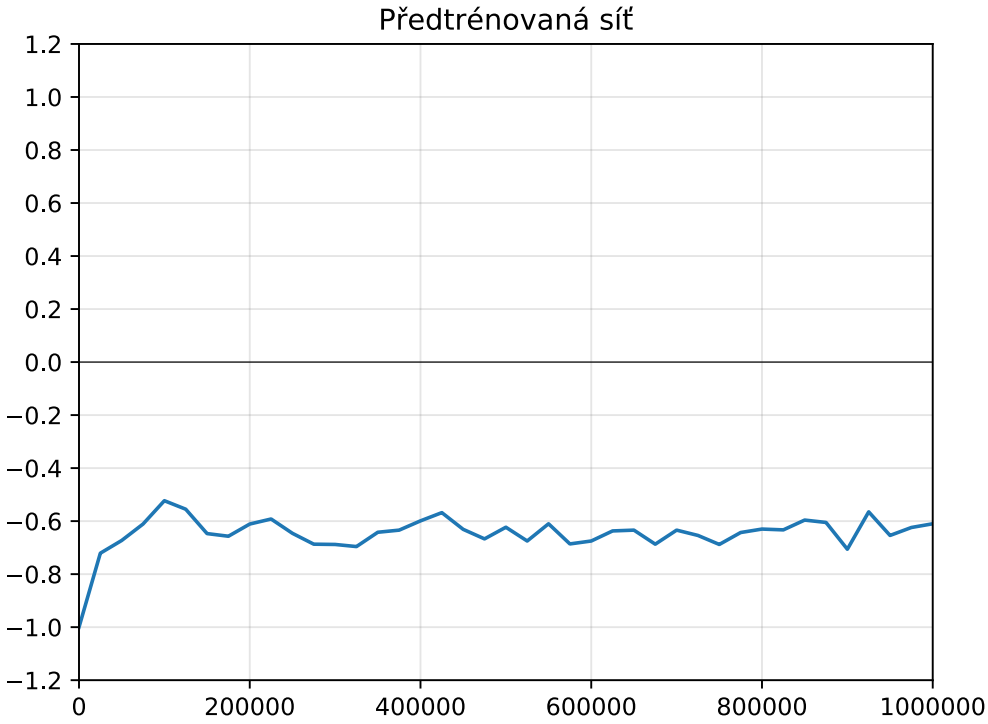
Obrázek 5.4: Pokračování tréninku agenta 2.

5.3 Učení s využitím předtrénované sítě

Posledním provedeným experimentem je učení na více úlohách s použitím předtrénované konvoluční části neuronové sítě. Pro tento experiment jsem využil předtrénované sítě pro klasifikaci obrázků SqueezeNet. Úkol mého agenta je sice velmi odlišný od úkolu SqueezeNetu, ale to by při použití konvoluční části nemuselo vadit. Pro vstup o velikosti 56 x 56 je výstupem konvoluční části sítě aktivační mapa 3 x 3, kde je každý pixel popsaný 512 kanály. Vstupem SqueezeNetu je pouze jeden barevný obrázek, tudíž stav v tomto experimentu není popsán čtyřmi snímky, ale jedním normalizovaným[3] obrázkem. Pro učení na více scénářích jsem využil architekturu s Embedding vrstvou, která se učila rychleji. Ačkoli Squeezenet patří mezi sítě pro klasifikaci obrazu s menším počtem parametrů, stále má mnohem více vrstev než jednoduchá síť složená ze tří vrstev. To se projevilo na výpočetní náročnosti, proto jsem nastavil počet kroků agenta na 1 000 000. Parametry předtrénované části jsou zmražené, takže se netrénují.

Z výsledků v grafu 5.5 lze vypozorovat, že se agent při použití předtrénované konvoluční části neučí, nebo se učí extrémně pomalu. Příčinou může být příliš velká odlišnost původní

[3] https://pytorch.org/docs/master/torchvision/models.html



Obrázek 5.5: Učení na více úlohách s použitím předtrénované sítě.

funkce sítě od aktuálního úkolu. U klasifikace například nezáleží na poloze objektů, která je pro hraní Doomu klíčová. Mezi vrstvami používá Max-pooling, kvůli kterému síť přestane vnímat polohu objektu. [13] Dalším důležitým rozdílem jsou vstupní data. SqueezeNet je trénovaný na obrázcích o velikosti 224 x 224 z reálného světa, což se výrazně liší od snímků ze hry ve formátu 56 x 56. V každém případě je použití předtrénované sítě SqueezNet pro hraní Doomu nepoužitelné.

Kapitola 6

Závěr

Cílem práce bylo naučit neuronovou síť plnit několik jednoduchých úloh ve 3D prostředí. Pro dosažení tohoto cíle byl vybrán algoritmus Rainbow, což je algoritmus DQN obohacený o několik vylepšení. Jako herní prostředí byla použita platforma ViZDoom. Navrhl a implementoval jsem dvě varianty agenta Rainbow, které umožňují učení na více úlohách zároveň a poté jsem s nimi provedl experimenty na 11 trénovacích úlohách z prostředí ViZDoom. Úlohy jsem upravil tak, aby se celkové odměny pohybovaly zhruba od -1 do 1. Kvůli výpočetní náročnosti byly experimenty provedeny pouze jednou.

Nejdříve jsem porovnával rychlost učení agenta Rainbow s DDQN na dvou různých úlohách. V obou úlohách byl Rainbow výrazně úspěšnější. V první úloze dosahoval po 1 000 000 trénovacích kroků průměrných odměn blízko k 1, zatímco DDQN dosahovalo průměru kolem -0.9. Ve druhé úloze Rainbow už po 100 000 krocích získával odměny zhruba o 0.8 vyšší než DDQN.

V experimentech s učením na více úlohách byla úspěšnější architektura s Embedding vrstvou. Agent se byl schopen po 2 000 000 krocích naučit plnit 4 úlohy tak, že v nich dosahoval průměrných odměn blízkých 1. Průměrná odměna ze všech úloh se pohybovala kolem -0.1, tedy téměř 50 % z maxima. V dalších 2 000 000 krocích nastal viditelný pokrok i v některých dalších úlohách. Lze tedy předpokládat, že by se agent při delším trénování naučil plnit i ostatní úlohy. Druhá varianta agenta byla méně úspěšná, agent se za stejnou dobu naučil rozumně hrát pouze dvě úlohy. Při nahrazení konvoluční části sítě předtrénovanou částí sítě SqueezeNet se agent vůbec neučil.

Z pohledu budoucího vývoje by bylo vhodné experimenty několikrát zopakovat a potvrdit, že dosažené výsledky nebyly náhoda. Z hlediska učení na více úlohách zároveň by mohla být zajímavá kombinace architektury s Embedding vrstvou a algoritmu A3C. Agenti by se asynchroně učili na různých úlohách a gradienty propagovaly do globální sítě, která by obsahovala Embedding vrstvu pro popis prostředí. Pokud je agent A3C schopný učit se na jednotlivých úlohách, mohl by být tento přístup efektivnější.

Literatura

[1] Ashraf, M.: Reinforcement Learning Demystified: Markov Decision Processes (Part 1). Duben 2018, [Online; navštíveno 30.4.2019].
URL https://towardsdatascience.com/reinforcement-learning-demystified-markov-decision-processes-part-1-bf00dda41690

[2] Bellemare, M. G.; Dabney, W.; Munos, R.: A Distributional Perspective on Reinforcement Learning. 2017, arXiv:1707.06887.

[3] Brownlee, J.: Gentle Introduction to the Adam Optimization Algorithm for Deep Learning. Červenec 2017, [Online; navštíveno 2.5.2019].
URL https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[4] Fortunato, M.; Azar, M. G.; Piot, B.; aj.: Noisy Networks for Exploration. 2017, arXiv:1706.10295.

[5] Geerinck, X.: Bellman Equations. Květen 2018, [Online; navštíveno 30.4.2019].
URL https://xaviergeerinck.com/bellman-equations

[6] van Hasselt, H.; Guez, A.; Silver, D.: Deep Reinforcement Learning with Double Q-learning. 2015, arXiv:1509.06461.

[7] Hessel, M.; Modayil, J.; van Hasselt, H.; aj.: Rainbow: Combining Improvements in Deep Reinforcement Learning. 2017, arXiv:1710.02298.

[8] Juliani, A.: Simple Reinforcement Learning with Tensorflow Part 4: Deep Q-Networks and Beyond. Září 2016, [Online; navštíveno 3.5.2019].
URL https://medium.com/@awjuliani/simple-reinforcement-learning-with-tensorflow-part-4-deep-q-networks-and-beyond-8438a3e2b8df

[9] Juliani, A.: Simple Reinforcement Learning with Tensorflow Part 8: Asynchronous Actor-Critic Agents (A3C). Prosinec 2016, [Online; navštíveno 4.5.2019].
URL
https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2

[10] Kaelbling, L. P.; Littman, M. L.; Moore, A. W.: Reinforcement Learning: A Survey. 1996, arXiv:cs/9605103.

[11] Lample, G.; Chaplot, D. S.: Playing FPS Games with Deep Reinforcement Learning. 2016, arXiv:1609.05521.

[12] Li, Y.: Deep Reinforcement Learning. 2018, arXiv:1810.06339.

[13] Matiisen, T.: *Demystifying Deep Reinforcement Learning*. Prosinec 2015, [Online; navštíveno 30.4.2019].
URL https://www.intel.ai/demystifying-deep-reinforcement-learning/

[14] Mnih, V.; Badia, A. P.; Mirza, M.; aj.: Asynchronous Methods for Deep Reinforcement Learning. 2016, arXiv:1602.01783.

[15] Mnih, V.; Kavukcuoglu, K.; Silver, D.; aj.: Playing Atari with Deep Reinforcement Learning. 2013, arXiv:1312.5602.

[16] Mnih, V.; aj.: Human-level control through deep reinforcement learning. *Nature*, Únor 2015.
URL https://doi.org/10.1038/nature14236

[17] Parmar, R.: Common Loss functions in machine learning. Září 2018, [Online; navštíveno 2.5.2019].
URL https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23

[18] Schaul, T.; Quan, J.; Antonoglou, I.; aj.: Prioritized Experience Replay. 2015, arXiv:1511.05952.

[19] Simonini, T.: Diving deeper into Reinforcement Learning with Q-Learning. Duben 2018, [Online; navštíveno 1.5.2019].
URL https://medium.freecodecamp.org/diving-deeper-into-reinforcement-learning-with-q-learning-c18d0db58efe

[20] Simonini, T.: Improvements in Deep Q Learning: Dueling Double DQN, Prioritized Experience Replay, and fixed Q-targets. Červenec 2018, [Online; navštíveno 5.5.2019].
URL https://medium.freecodecamp.org/improvements-in-deep-q-learning-dueling-double-dqn-prioritized-experience-replay-and-fixed-58b130cc5682

[21] Simonini, T.: An introduction to Policy Gradients with Cartpole and Doom. Květen 2018, [Online; navštíveno 3.5.2019].
URL https://medium.freecodecamp.org/an-introduction-to-policy-gradients-with-cartpole-and-doom-495b5ef2207f

[22] Sutton, R. S.; Barto, A. G.; aj.: *Introduction to reinforcement learning*, ročník 135. MIT press Cambridge, 1998.

[23] Tomassoli, M.: Distributional RLs. Prosinec 2017, [Online; navštíveno 5.5.2019].
URL https://mtomassoli.github.io/2017/12/08/distributional_rl/

[24] Violante, A.: Simple Reinforcement Learning: Q-learning. Březen 2018, [Online; navštíveno 1.5.2019].
URL https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56

[25] Wang, Z.; Schaul, T.; Hessel, M.; aj.: Dueling Network Architectures for Deep Reinforcement Learning. 2015, arXiv:1511.06581.

[26] Wikipedia: Game complexity — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Game%20complexity&oldid=882151381, 2019, [Online; navštíveno 8.5.2019].

[27] Wikipedia: Markov decision process — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Markov%20decision%20process&oldid=893936705, 2019, [Online; navštíveno 6.5.2019].

[28] Wikipedia: Rectifier (neural networks) — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Rectifier%20(neural%20networks)&oldid=890629677, 2019, [Online; navštíveno 8.5.2019].

[29] Wydmuch, M.; Kempka, M.; Jaśkowski, W.: ViZDoom Competitions: Playing Doom from Pixels. *IEEE Transactions on Games*, 2018.

Příloha A

Obsah přiloženého CD

doc/ Zdrojové soubory technické zprávy a přeložené PDF.

src/ Zdrojové soubory použité pro experimenty.

videos/ Video obsahující výsledky práce.

README.txt Soubor s popisem jednotlivých souborů.