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

SDÍLENÍ ZKUŠENOSTÍ V POSILOVANÉM UČENÍ

EXPERIENCE TRANSFER IN REINFORCEMENT LEARNING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — RADEK MOJŽÍŠ
AUTHOR

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

BRNO 2019

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

Zadání bakalářské práce

22233

Student: **Mojžíš Radek**
Program: Informační technologie
Název: **Sdílení zkušeností v posilovaném učení**
Shared Experience in Reinforcement Learning
Kategorie: Umělá inteligence
Zadání:

1. Prostudujte základy neuronových sítí a posilovaného učení.
2. Vytvořte si přehled o současných metodách využívají neuronové sítě a posilované učení s prvky umožňující sdílení zkušeností a přenos informací mezi příbuznými úlohami (transfer learning).
3. Vyberte konkrétní metodu aplikovatelnou na určitou homogenní třídu starých 2D her.
4. Implementujte navrženou metodu a proveďte experimenty se zaměřením na efektivitu přenosu zkušeností.
5. Porovnejte dosažené výsledky a diskutujte možnosti budoucího vývoje.
6. Vytvořte stručné video prezentující vaši práci, její cíle a výsledky.

Literatura:

- Ian Goodfellow and Yoshua Bengio and Aaron Courville: Deep Learning. MIT Press, 2016.
- Sergey Levine et al.: CS 294: Deep Reinforcement Learning. Berkeley, 2017. http://rll.berkeley.edu/deeprlcourse/
- Fei-Fei Li et al.: CS231n: Convolutional Neural Networks for Visual Recognition. Stanford, 2017. http://cs231n.stanford.edu/

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: 11. ledna 2019

Abstrakt

Cílem této práce je využít metod sdílení zkušeností při učení neuronových sítí na problém posilovaného učení. Jako testovací prostředí používám staré 2D konzolové hry, jako například space invaders nebo Phoenix. Testuji vliv přeučení již natrénovaných modelů na nová podobná prostředí. Dále zkouším metody pro přenášení rysů domén. Nakonec se zaměřuji na možnosti trénování modelu na několika prostředích simultánně. Z výsledků získáme přehled o možnostech sdílení zkušeností při trénování modelů pro algoritmy posilovaného učení.

Abstract

The aim of this thesis is to use methods of transfer learning for training neural network on a reinforcement learning tasks. As test environment, I am using old 2D console games, such as space invaders or phoenix. I am testing the impact of re-purposing already trained models for different environments. Next I use methods for domain feature transfer. Lastly i focus on the topic of multi-task learning. From the results we can gain insight into possibilities of using transfer learning for reinforcement learning algorithms.

Klíčová slova

Posilované učení, DQN, sdílení zkušeností, konvoluční neuronové sítě, rozpoznávání obrazu, strojové učení, umělá inteligence

Keywords

Reinforcement learning, DQN, Transfer learning, convolutional neural network, image recognition, machine learning, artificial Intelligence

Citace

MOJŽÍŠ, Radek. *Sdílení zkušeností v posilovaném učení.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Hradiš, Ph.D.

Sdílení zkušeností v posilovaném učení

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.

. . . . . .
Radek Mojžíš
15. května 2019

Poděkování

Chtěl bych poděkovat vedoucímu mé bakalářské práce Ing. Michalu Hradišovi, Ph.D. za odbornou konzultaci v rámci vypracování této práce.

Obsah

1 Úvod **2**

2 Principy a způsoby sdílení zkušeností **3**
2.1 Definice problému sdílení zkušeností 3
2.2 Strategie sdílení zkušeností 4

3 Principy zpětnovazebního učení **7**
3.1 Markovův rozhodovací proces 7
3.2 zpětnovazební učení 8

4 Algoritmus DQN **10**
4.1 Q učení 10
4.2 DQN 11
4.3 Architektura modelu neuronové sítě pro DQN 12

5 Prostředí starých 2D her **14**
5.1 OpenAi Gym 14
5.2 Úpravy vybraných prostředí 14
5.3 Referenční hodnoty pro jednotlivá prostředí 19

6 Experimenty sdílení zkušeností pro DQN **22**
6.1 Naivní přístup k přenosu zkušeností 22
6.2 Využití metody přenesení rysů domén 24
6.3 Simultánní učení ve více prostředích 24

7 Implementační detaily **26**

8 Závěr **28**

Literatura **29**

Kapitola 1

Úvod

Člověk je schopen učit se velmi rychle. Je to částečně i proto, že dokáže využít svých předchozích znalostí o věcech, které dělal v minulosti. V této práci se zaměřuji na aplikaci principů přenosu zkušeností mezi neuronovými sítěmi. Pokouším se zjistit jaké možnosti tyto principy poskytují v oblasti strojového učení známé pod názvem posilované učení.

Schopnost neuronových sítí naučit se extrémně komplikované funkce s tisíci až milióny vstupních parametrů je naprosto fascinující. Pro využití jejich síly je ale potřeba obrovský výpočetní výkon. Metody zabývající se přenosem zkušeností se snaží využít informace, které se již naučily jiné neuronové sítě a aplikovat je v nových modelech. Přenesení zkušeností z předchozího modelu do nového modelu, nám můžeme ušetřit čas při trénování. Tento čas potom můžeme využít pro trénování lepšího nebo přesnějšího modelu.

Přenos zkušeností se většinou používá v kontextu klasifikačních úloh. Jako příklad mohu uvést třeba rozpoznávání obrazu nebo spam filtry. Tyto metody předpokládají že máme k dispozici trénovací sadu, což je množina vstupů a správných výstupů/štítků těchto vstupů. Neuronová síť se potom učí aby její výstupy odpovídaly vzorovým výstupům. Oproti tomu v oblasti posilovaného učení (en, Reinforcement learning) žádná takováto trénovací sada neexistuje, a síť se učí pouze ze své interakce s prostředím.

Mým cílem v této práci je implementovat model který je schopný učit se pomocí interakce se svým prostředím. Následně potom na tento model aplikovat metody a principy přenosu a sdílení zkušeností.

V této práci nejprve vysvětluji základní principy přenosu zkušeností, následně popisuji úvod do problematiky zpětnovazebního učení. Dále se zaměřuji na algoritmus použitý pro experimenty, kterým je DQN. Popisuji také jaké hry používám jako prostředí pro provádění experimentů. Poté prezentuji výsledky těchto experimentů a hodnotím jejich úspěšnost. Poslední kapitola je potom zaměřená na implementační detaily praktické části této práce.

Kapitola 2

Principy a způsoby sdílení zkušeností

Při tradičním učení se modely učí jeden specifický úkol. Nepředpokládáme žádnou předchozí znalost úkolu. Pokud chceme algoritmus s přenosem zkušeností, můžeme sdílet společné poznatky o jednotlivých úkolech mezi modely. Díky tomu jsme schopni například natrénovat model plnící úkol pro který máme omezené množství dat tím, že využijeme poznatků z jiného podobného modelu řešící podobný úkol. Tímto se může zvýší efektivita učení našeho nového modelu.

2.1 Definice problému sdílení zkušeností

Problém sdílení zkušeností vyjadřujeme pomocí domény a úkolu [11]. Doména obsahuje množinu rysů χ a marginální distribuci pravděpodobností $P(X)$, kde $X = \{x_1, ..., x_n\} \in \chi$. X je jeden konkrétní vzorek z domény (například obrázek) který má nějaké rysy, $P(X)$ potom říká, jaké rysy mají vzorky v této doméně. Tuto definici lze intuitivně pochopit následovně: pokud jsou domény rozdílné, mají buď jiné rysy nebo mají jinou distribuci rysů ve svých jednotlivých vzorcích.

Pro doménu $D = \{X, P(X)\}$, definujeme úkol jako uspořádanou dvojici $T = \{y, f(\cdot)\}$, kde y je množina štítků, a $f(\cdot)$ je predikční funkce. Tuto funkci je třeba naučit podle trénovacích dat, což jsou dvojice $\{x_i, y_i\}$ kde $x_i \in X$ a $y_i \in y$. Potom funkce $f(\cdot)$ může být použita pro předpověď štítku $f(x)$ i pro nové hodnoty x.

S takto definovanými doménami můžeme při pokusu o přenos zkušeností z domény D_s s úkolem T_s do domény D_t s úkolem T_t narazit na jeden z následujících případů. [15] Jako příklady pro názornost uvažuji kontext práce s textem.

- $\chi_s \neq \chi_t$, rysy domén jsou rozdílné. Toto by bylo ekvivalentní odlišnému jazyku zpracovaných textů.
- $P(X_s) \neq P(X_t)$, distribuce rysů jsou rozdílné. Texty se zabývají jinými tématy.
- $y_s \neq y_t$, množiny štítků mezi úkoly jsou rozdílné. Texty řadíme do jiného počtu jiných kategorií.
- $P(y_s|X_s) \neq P(y_t|X_t)$, Distribuce podmíněných pravděpodobností dvou úkolů jsou rozdílné. To nastane ve chvíli, kdy zdrojová a cílová doména mají nevyvážený poměr počtu štítků v jednotlivých třídách třídách.

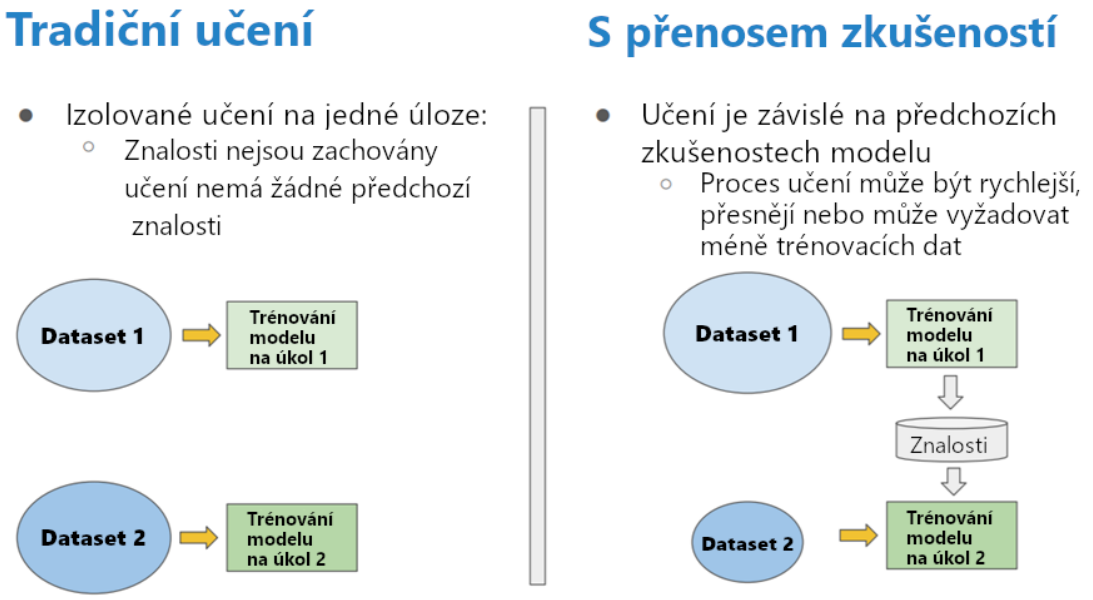
Tyto definice nám pomohou lépe pochopit podstatu strategií popsaných v následující sekci.

2.2 Strategie sdílení zkušeností

Přístupy ke sdílení zkušeností se dělí do několika základních kategorií, podle toho, co mají přenášené domény společné, nebo co mají přenášené úkoly společné. doméně.[15]

Znovupoužití modelu(Přenášení vah) Nejčastější přístup je využití modelu, který je natrénovaný ve zdrojové doméně, pro řešení úkolů v cílové doméně. Sdílení zkušeností pomocí přenosu modelu je pravděpodobně nejčastěji využívaná metoda přenosu zkušeností.

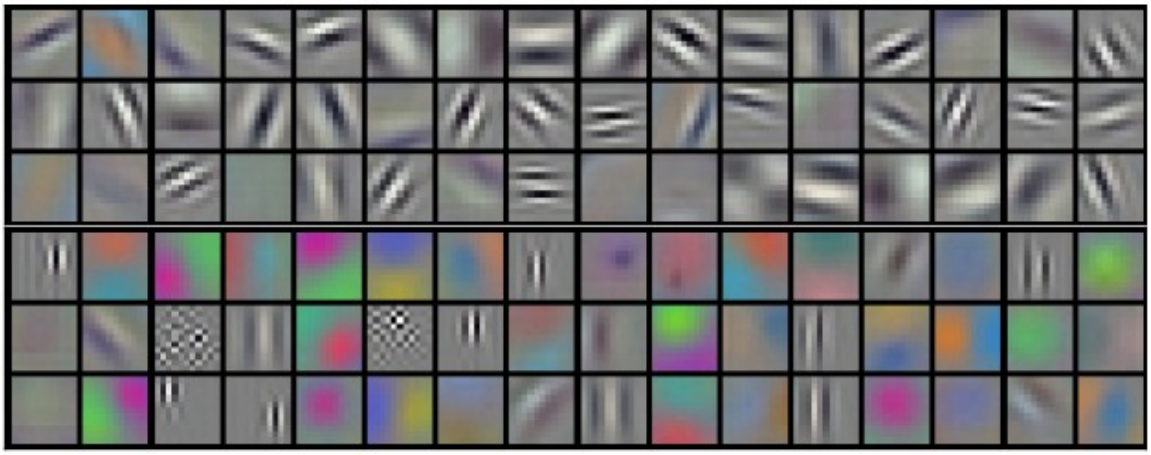
Znovupoužitím modelu rozumíme natrénování modelu na úkolu ve zdrojové doméně a následně poté použít takto natrénovaný model v cílové doméně. Takto použitý model může využít související zkušenosti které získal v předchozím trénování a zlepšit tak svoje vlastnosti.



Obrázek 2.1: Znázornění metody znovupoužití modelu

přenášení vzorků Tato metoda se snaží obohatit sadu dat cílového úkolu o data ze zdrojového úkolu. Přímé znovupoužití vzorků ze zdrojové domény na úkol v cílové doméně je ideální scénář. Ve většině případů ale zdrojová data nemůžeme použít přímo. Zpravidla je třeba vybrat specificky vzorky ze zdrojové domény které mohou být použity spolu s daty pro úkol z cílové domény. Například v práci "Instance-based Deep Transfer Learning"[18] je použit model který je předtrénovaný ve zdrojové doméně. Zkušenosti z trénování ve zdrojové doméně potom aplikuje na úkol v cílové doméně, aby zjistil vliv trénovacích vzorků na svou přesnost v cílové doméně. Následně se optimalizuje trénovací sada odebráním vzorků které by snížili přesnost tohoto modelu. Tím zlepšují vlastnosti nových modelů na cílovém úkolu.

Přenášení rysů Tento přístup se snaží minimalizovat rozdíly mezi doménami tím, že se snaží využít rysů které mají domény společné aby zrychlil nebo zpřesnil učení modelu v cílové doméně.[5] V praxi se s tímto přístupem můžeme setkat například při rozpoznávání obrazu, konvoluční vrstvy modelů trénovaných na komplexní sadě dat a složitém úkolu, se dokáží naučit konvoluční filtry, které jsou relevantní v podstatě ve všech odvětvích strojového učení využívající strojové vidění.



Obrázek 2.2: příklady filtrů které se naučil AlexNet (Krizhevsky et al., 2012)

Tuto metodu můžeme například použít pro rozpoznávání obrázků tak, že využijeme již naučené konvoluční vrstvy nejlepších volně dostupných předtrénovaných sítí trénovaných pro ImmageNet. Při učení potom parametry těchto vrstev vůbec neměníme, nebo je učíme s malou rychlostí učení, aby model nezapomněl předchozí znalosti. Modely trénované na úkolu ImmageNet jsou dobrým základem pro obecné problémy v oblasti strojového vidění, protože znalosti potřebné pro úspěšnost v tomto úkolu jsou relevantní pro mnoho jiných úkolů.

Učení rysů které se s doménou nemění Dalším způsobem jak přenášet zkušeností je to, že se učí reprezentace a rysy které se nemění s ohledem na naše domény. Tento princip je podobný konceptu přenášení rysů. Hlavní rozdíl je ten, že vytváření reprezentací které se nemění na doméně je daleko levnější, a často o mnoho flexibilnější, než tvořit model který je dobrý na všechny úkoly. Toto je ještě významnější pokud bereme v potaz i jiné oblasti strojového učení než počítačové vidění.

Zmatení domény Pro zaručení podobnosti reprezentací zdrojových a cílových domén se také používá metoda zmatení domén[14]. Tato metoda je založena na představení nového cíle do již existujícího modelu. Tento cíl zabraňuje modelu učit se rysy, které by mu umožnili rozeznat domény mezi sebou. Jde o běžný klasifikační úkol, kde se model snaží předpovědět doménu vstupu. Funkce ztráty tohoto výstupu je standardní ztráta používaná při klasifikaci, nicméně gradient určený touto ztrátou je pro zbytek sítě převrácený. Intuitivně tato metoda funguje tak, že se model snaží minimalizovat ztrátu pro nový úkol přičemž nedokáže rozlišovat mezi doménami. Cílem této metody je, aby se model učil jen rysy, které mají domény společné.

Učení na několika prostředích Učení na několika prostředích je metoda jejíž cílem je zlepšit generalizační schopnosti modelu.[13] Snaží se toho dosáhnout použitím informací obsažených v trénovacích signálech souvisejících úkolů aby se dokázal rozhodnout v nových stavech. Učení probíhá paralelně za použití společné reprezentace jednotlivých domén. Co se síť naučí v jednom úkolu jí může pomoci dělat ostatní úkoly lépe.

Negativní přenos Při použití těchto metod je třeba dávat si pozor, k čemu je která z nich užitečná a jaké jsou jejich slabiny. Pokud jsou například zdrojová a cílová doména příliš rozdílné, může dojít ke zhoršení modelu až do bodu, kdy by náhodný nový model byl lepší. Takovému případu se říká negativní přenos zkušeností. Může k němu dojít třeba tak, že i přesto, že jsou domény podobné, mohou být množiny rysů které jsou důležité pro plnění úkolů disjunktní, popřípadě mít jen velmi málo společné. V tu chvíli se model nejdříve musí odnaučit zapamatované rysy. A potom se znovu naučit nové rysy z nové domény. Existují metody jak tomuto fenoménu do značné míry zabránit, nicméně jsou složité a v této práci se jimi nebudu zabývat. [20]

Kapitola 3

Principy zpětnovazebního učení

Posilované učení, též "zpětnovazebí učení", je oblast strojového učení zabývající se chováním agentů v různých prostředích.[3] Cílem algoritmů této třídy je trénovat agenta, který provádí akce tak, aby maximalizoval odměnu dosaženou během průchodu prostředím. Posilované učení je bráno jako jeden ze tří hlavních přístupů ke strojovému učení, spolu s učením s učitelem (en:*supervised learning*) a učením bez učitele (en:*unsupervised learning*).

Pokud provádíme učení s učitelem [1], předpokládáme, že máme k dispozici takzvanou trénovací sadu. Tato sada obsahuje data a jejich patřičné štítky. Příkladem takového učení může být například základní úkol klasifikace obrázků, kdy máme k dispozici zpravidla velké množství obrázků a ke každému obrázku máme v trénovací sadě i štítek, který říká, do jaké kategorie tento obrázek patří. V posilovaném učení žádná takováto sada neexistuje. Proto je třeba z dat které agent shromáždí při interakci s prostředím zjistit, jaké akce vedly k dobrým odměnám, a jaké akce vedly ke špatným odměnám.

Tato interakce agenta a prostředí se v kontextu posilovaného učení matematicky popisuje pomocí takzvaného Markovova rozhodovacího procesu [19]. Pomocí tohoto matematického popisu potom můžeme definovat algoritmy které k učení žádné štítky nepotřebují, nebo si je dokáží obstarat samy.

Metody které se snaží řešit problém posilovaného učení se stýkají s mnohými problémy. Některé plynou ze samotné podstaty neuronových sítí a některé z podstaty představeného úkolu.

3.1 Markovův rozhodovací proces

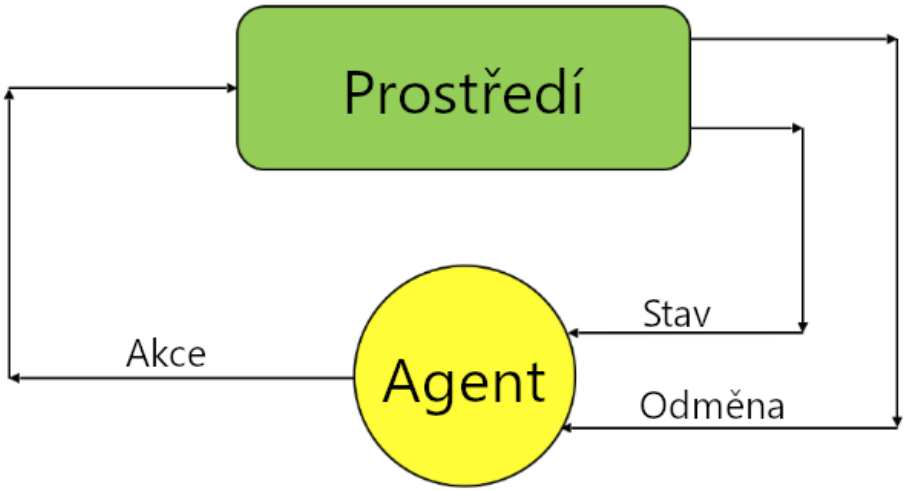
Markovův rozhodovací proces [19] je stochastický proces ovládání s diskrétním časem. Matematicky je vhodný pro modelování rozhodování agenta v prostředích kdy výsledek akce je částečně náhodný a částečně závislý na akci tohoto agenta.

Markovův rozhodovací proces je vždy v nějakém stavu s, v tomto stavu je možné zvolit nějakou z akcí a a následně proces přejde do stavu s' přičemž vrací odměnu za tento přechod r.

Formálně jde o uspořádanou čtveřici (S, A, P_a, R_a) kde

- S je množina stavů ve kterých se může proces nacházet,
- A je množina akcí které je možné zvolit,
- $P_a(s, s') = Pr(s_t + 1 = s'|s_t = s, a_t = a)$ je pravděpodobnost že proces po zvolení akce a ve stavu s v čase t přejde do stavu s' v čase $t + 1$,

- $R_a(s, s')$ je odměna dosažená zvolením akce a a následovným přechodem ze stavu s do stavu s'.



Obrázek 3.1: Znázornění interakce mezi agentem a prostředím

Algoritmy používající takovouto definici Markovova rozhodovacího procesu potom ještě většinou předpokládají, že velikost stavového prostoru S a velikost množiny akcí A jsou konečné. Tyto algoritmy také definují několik konceptů které jsou klíčové pro praktické využití posilovaného učení.

3.2 zpětnovazební učení

Ve zpětnovazebném učení předpokládáme existenci agenta. Tento agent se pohybuje v nějakém prostředí které je popsáno pomocí Markovského rozhodovacího proceseu. Agentovo rozhodování o tom, jakou akci v daném stavu zvolit je obecně modelované jako matematické zobrazení z množiny stavů do množiny akcí. Tomuto rozhodování se také říká politika a označuje se $\pi : S \to A$. Algoritmy posilovaného učení se potom snaží najít takovou politiku která maximalizuje celkovou odměnu posbíranou při průchodu prostředím.

Funkce hodnoty stavu Funkce hodnoty stavu je funkce, která se snaží zjistit, jak dobré je být v nějakém stavu. Tuto funkci označujeme jako $V_\pi(S) = E(R)$[6], kde

$$E(R) = r_t + r_{t+1} + r_{t+3} + r_{t+3} + ... + r_{t+n}$$

je očekávaná celková budoucí odměna kterou dostaneme v případě, že začneme ve stavu S a budeme až do konce epizody vybírat akce podle politiky π. Očekávanou odměnu používáme proto, že po vykonání akce odměna nemusí být přidělena ihned. Někdy může trvat i velmi dlouho, než se projeví odměna za danou akci. Tento problém je známý pod anglickým označením "Credit assignment problem", nebo "delayed reward problem".

Takovýto přístup k očekávané odměně má ale také několik vad kvůli kterým se nepoužívá. Jednou z nich je, že kvůli stochastické podstatě Markovských rozhodovacích procesů nemůžeme přesně určit budoucí odměny. Čím dále v budoucnosti odměna je, tím menší je

pravděpodobnost, že se k ní vůbec agent dostane. Tento problém se spolu s dalším problémem - problémem potenciálně nekonečného horizontu - řeší zavedením konceptu stárnutí. [12] Problém nekonečného horizontu říká, to že pokud je časový horizont daného prostředí potenciálně nekonečný, může i očekávaná odměna být potenciálně nekonečná. Potenciální nekonečna a obecně velká čísla v neuronových sítích zavádějí velké chyby a snažíme se jim proto pokud možno vyvarovat. Oba tyto problémy, jak jsem již zmiňoval, se řeší zavedením konceptu stárnutí. Koncept stárnutí označujeme jako γ a platí, že $\gamma \leq 0$, vzporec pro očekávanou budoucí odměnu potom můžeme zapsat jako [9] [6]

$$E(R) = r_t + r_{t+1}\gamma + r_{t+2}\gamma^2 + r_{t+3}\gamma^3 + ... + r_{t+n}\gamma^n$$

Díky tomto přístupu mají odměny, které očekáváme později v budoucnosti menší váhu, než okamžité odměny. Snižováním hodnoty γ můžeme ovlivňovat jak daleko do budoucnosti se náš agent dívá. Nastavení této hodnoty na 0 by potom znamenalo, že ho bude zajímat pouze okamžitá odměna. Nejčastěji se používá hodnota blízká 1. Například společnost DeepMind použila hodnotu 0.99 pro hraní her konzole Atari 2600 [10].

Metody, které se snaží najít hodnotící funkci, závisí na teorii o Markovových rozhodovacích procesech, kde politiku můžeme nazvat optimální v případě, že dosahuje nejvyšší možné očekávané odměny z jakéhokoliv počátečního stavu. Takováto politika vždy existuje, jen může být velmi nepraktické ji přímo hledat.[3] Formálně potom můžeme politiku politiku π obecně definovat jako

$$V^{\pi}(S) = E(R|s, \pi)$$

kde R je očekávaná odměna za předpokladu, že začínáme ve stavu S a provádíme akce podle politiky π. Optimální politiku potom definujeme jako $V^*(s)$ a má maximální hodnotu funcke $V^{\pi}(s)$ v závislosti na π. Neboli

$$V^*(S) = max_{\pi}(V^{\pi}(S)).$$

Politika která dosahuje těchto maximálních hodnot se nazývá optimální, a dosahuje nejlepších možných výsledků pro všechny stavy.

I přesto, že pro definici optimality je dostatečné použití funkce hodnoty stavu, je vhodné představit i funkci Q hodnotící jednotlivé akce ve stavech, neboť je praktičtější ji aproximovat. Tuto funkci potom můžeme definovat jako

$$Q^{\pi}(s, a) = E(R|s, a, \pi))$$

Kde R nyní značí odměnu dosaženou použitím akce a ve stavu s za předpokladu že následujeme politiku π až do konce epizody. Podobně jako funkce hodnoty stavů potom definujeme funkci $Q^*(s, a)$, která obsahuje očekávané odměny za akce v jednotlivých stavech pro optimální politiku. Pokud známe $Q^*(s, a)$, můžeme z ní odvodit optimální politiku tak, že vždy vykonáváme akci, pro kterou funkce $Q^*(s, a)$ nabývá nejvyšší hodnoty. Metoda získání takovéto funkce se potom nazývá Q učení, a zabýváme se ji v následující kapitole. [3]

Kapitola 4

Algoritmus DQN

Algoritmus DQN je nejběžněji používaný algoritmus pro posilované učení. Je velmi flexibilní a dosahuje vynikajících výsledků i na velmi složitých úlohách. Tento algoritmus nevyžaduje model prostředí a dokáže si poradit i s problémy které vyžadují zvažování stochastických odměn. Je snadné jej paralelizovat a nevyžaduje žádné předchozí informace o doménách se kterými pracuje. Pro vysvětlení algoritmu DQN je ale nejprve třeba vysvětlit princip Q učení, o kterém píši v následující sekci. 1

4.1 Q učení

Cílem Q učení je najít ideálně funkci $Q^*(s, a)$, pokud takovouto funkci máme, máme i přístup k optimální politice vybírání akcí v daném prostředí. A to tak, že vždy zvolíme akci jako $a = argmax_a(Q^*(s, a))$.

Pro jakýkoliv konečný Markovský rozhodovací proces dokáže Q učení teoreticky najít optimální Q funkci, to za předpokladu, že máme nekonečnou paměť a nekonečný čas na učení. [7]

V praxi nekonečný čas ani paměť nemáme, takže pro jednoduchost prozatím předpokládejme, že máme prostředí s malým počtem stavů a malým počtem akcí. Q funkci pro takové prostředí můžeme potom zapsat do tabulky. Tato tabulka bude mít řádek pro každý možný stav tohoto prostředí a sloupec pro každou akci kterou lze v daném stavu zvolit. [6] Jednoduchý online algoritmus Q učení potom bude vypadat následovně:

Algorithm 1 Q learning

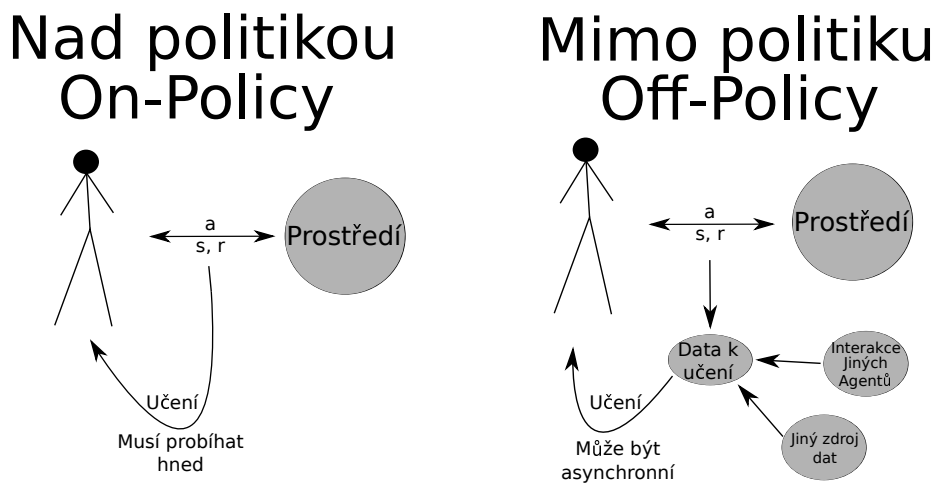
```
Libovolně inicializuj tabulku Q hodnot
while Učení není dokončeno do
    Restartuj prostředí
    s ← Počáteční stav prostředí
    repeat
        Vyber akci podle Q(s)
        Proveď akci r ← odmena, s' ← newstate
        Q(s,a) ← Q(s,a) + α(r + γ argmax_a Q(s',a) − Q(s,a))
    until Konec epizody
```

Tento algoritmus bohužel trpí několika nedostatky. Nikde není přímo uvedeno, jak by se měla vybírat akce v době učení. Toto je poměrně zásadní informace, neboť v rámci učení

by bylo dobré, aby agent prozkoumal co nejvíce různých stavů. Využít pro tento případ Greedy politiku by mohlo způsobit, že by agent po najití nějakého lokálního maxima nikdy nezkoušel jiné akce, které by dlouhodobě mohly přinést i větší odměnu. [6]

otázka objevování a vykořisťování prostředí. Toto je jedna ze základních otázek kterou se posilované učení zaobírá. Většinou se řeší zavedením proměnné $\epsilon \leq 1$, která vyjadřuje pravděpodobnost s jakou zvolíme náhodnou akci namísto optimální akce podle naší politiky. Této proměnné říkáme faktor prozkoumávání. V praxi na začátku nastavíme $\epsilon = 1$, takže náš agent provádí náhodné akce a postupně snižujeme hodnotu ϵ, díky tomu se agent začne postupně chovat tak, jak mu říká politika, která získává stále lepší přehled o prostředí.

Dalším nedostatkem algoritmu Q učení je, že nevyužívá největší sílu Q učení, kterou je to, že jej není třeba provádět nad politikou (en, *On policy*) kterou zrovna trénujeme.[6] Při Q učení nám nezáleží odkud získáváme trénovací vzorky, jediné co je podstatné je, aby tyto vzorky navštěvovaly stavy, které by navštěvoval agent řízený naučenou Q funkcí. Díky této vlastnosti můžeme v praxi lépe využít data získaná při průchodu agenta prostředím a můžeme využít paralelizaci.



Obrázek 4.1: Znázornění principu nad politikou (on policy) a mimo politiku (off policy).

Důvodem, proč se metoda Q učení nepoužívala až donedávna, je to, že je složité reprezentovat Q funkci v počítači. Pro jednoduchá prostředí lze zapsat její hodnoty do tabulky, nicméně pro složitější prostředí se velmi rychle začneme dostávat do problémů s pamětí. Tomuto fenoménu se také říká prokletí mnohodimensionality (en, The curse of dimensionality). S rostoucím počtem dimenzí stavového prostoru roste totiž počet jednotlivých stavů exponenciálně. Pokud bychom například zvážili barevný obrázek o velikosti 240x240 pixelů, tak zjistíme, že takových obrázků existuje $255^{3^{240^{240}}}$. Proto v praxi využíváme univerzální funkční aproximátory, jako například neuronové sítě. [6]

4.2 DQN

Algoritmus DQN Je vylepšenou verzí Q učení která využívá neuronové sítě. Díky jejich schopnosti generalizovat, lze poměrně jednoduchou síť naučit Q funkci i pro složité úkoly jako například hraní her. Oproti Q učení 1 je algoritmus DQN jen málo odlišný. Poprvé

byl představen ve článku "Playing Atari with Deep Reinforcement Learning"[10]. Oproti základnímu Q učení používá ale tato implementace několik vylepšení.

Jedním z použitých vylepšení je vzpomínková paměť (en: Replay memory), do které se ukládají informace o interakci agenta s prostředím. V podstatě jde o datovou strukturu do níž ukládáme čtveřice (s, r, a, s'), což jsou všechny informace potřebné k pozdějšímu trénování Q funkce.

Algorithm 2 Algoritmus DQN

Inicializuj Vzpomínkovou paměť na plnou kapacitu
for Všechny epizody **do**
    Restartuj prostředí
    $s \leftarrow$ počáteční stav prostředí
    repeat
        S pravděpodobností ϵ vyber náhodnou akci a,
        jinak vyber akci jako $a = argmax(Q(s))$
        Proveď akci s a získej odměnu r a nový stav s'
        Ulož čtveřici(s, a, r, s') do Paměti se vzpomínkami
        Vyber náhodnou sadu (s, a, r, s') z paměti vzpomínek
        $t_i = \begin{cases} r_i & \textit{pro koncove stavy } s_i \\ r_i + \gamma \cdot argmax(Q(s_i')) & \text{pro ostatni stavy} \end{cases}$
        proveď krok trénování se stavy s_i jako daty a t_i jako jejich štítky
    until Konec epizody

Později se objevila i další vylepšení, jako například duální architektura neuronové sítě, prioritní paměť se vzpomínkami a cílová stabilizační síť pro trénováni, při použití všech dostupných vylepšení se může rychlost učení sítě DQN až zněkolikanásobit[16].

V mé práci používám jen vylepšení cílové stabilizační sítě. Ta se používá proto, že když počítáme cílové hodnoty pro učení (v algoritmu t_i), využíváme $argmax(Q(s'))$, takže pokud máme síť která ještě není naučená, můžeme získávat nepřesné informace o tom, které akce jsou dobré a které jsou špatné. Navíc se nám mění funkce kterou chceme aproximovat v průběhu výpočtu. Proto využíváme druhou stabilizační síť která má uzamčené váhy [6]. Tím hlavní síť dostane čas aby získala informace o stavech které navštěvuje a učení je potom stabilnější. Váhy této sítě jsou potom periodicky nastavovány na hodnoty vah hlavní sítě.

4.3 Architektura modelu neuronové sítě pro DQN

Neuronová síť pro DQN je modelovaná podobně jako běžné sítě pro rozpoznávání obrazu. Ty se skládají z několika konvolučních vrstev pro extrakci rysů z obrazu a po nich následuje několik plně propojených vrstev pro plnění specifického úkolu. Oproti sítím pro rozpoznávání obrazu síť pro DQN neobsahuje max pooling vrstvy ani trénovací dropout vrstvy. Je to z toho důvodu, že max pooling vrstvy by zkreslovaly informace o pozicích objektů v obraze, které mohou být pro úkoly jako například hraní her velmi důležité.

Neuronová síť kterou používám pro trénování je založená na modelu, který použila společnost deep mind pro hraní her konzole atari 2600. 4.3 [17]. Originálně jako vstup pro tuto síť sloužily obrázky o velikosti 84x84, překonvertované do odstínů šedi a následně škálovaných na hodnoty mezi 0 a 1.

Já pro experimenty využívám o něco menší obrázky o velikosti 64x64 pixelů, je to z toho důvodu, že čím menší je velikost obrázků tím rychleji probíhají výpočty učení. Jediné na co je třeba dát si pozor je, aby na obrázcích bylo vidět vše podstatné.

konvoluční vrstvy	počet rysů	velikost okna	krok
1.	32	8x8	4
2.	64	4x4	2
3.	64	3x3	1

Plně propojené	počet neuronů
4.	1024
5.	512

Tabulka popisuje neuronovou síť použitou pro experimenty.

Počet výstupů této sítě je potom stejný, jako počet akcí, které je možné v daném prostředí zvolit. Jako ztrátovou funkci pro výpočet gradientu při učení používám průměrnou čtvercovou chybu (MSE).

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

Kde pro t_i je i-tý prvek cílových hodnot, y_i je i-tý prvek předpovězených hodnot a n je velikost vybrané sady z paměti se vzpomínkami. Z rovnice je vidět, že čím větší je rozdíl nové a původní hodnoty, tím více daný vzorek ovlivní změnu vah sítě. Často takové chování není žádoucí. V takových případech se používá průměrná absolutní chyba (MAE). Průměrná absolutní chyba se počítá jako

$$MAE = \frac{1}{n}\sum_{n=1}^{n} \mid t_i - y_i \mid .$$

.

Někdy se také používá kombinace těchto dvou funkcí, kde například pro rozdíl $t_i - y_i$ v intervalu $< -1, 1 >$ používáme průměrnou čtvercovou chybu a pro hodnoty mimo tento interval použijeme průměrnou absolutní chybu. Huberova chyba potom umožňuje změnit bod, kde se kvadratický průběh mění na lineární a je popsána rovnicí [2]

$$L_\delta = \frac{1}{n}\sum_{n=1}^{n}\begin{cases}\frac{1}{2}(t_i - y_i)^2 & \text{pro} \mid t_i - y_i \mid < \delta \\ \delta \mid t_i - y_i \mid - \frac{1}{2}\delta & \text{jinak}\end{cases}$$

Kde δ je označuje bod, ve kterém se průběh funkce mění z kvadratického na lineární.

Kapitola 5

Prostředí starých 2D her

Prostředí starých 2D konsolových her jsou ideální pro testování nových algoritmů strojového učení. Je to z toho důvodu že jsou většinou poměrně jednoduchá. Díky nim máme širokou škálu prostředí pro testování často teoretických "proof of concept"algoritmů pro strojové učení. Když potom tyto algoritmy fungují na jednoduchých hrách můžeme je potom testovat i na větší problémy třeba i z reálného světa.

5.1 OpenAi Gym

V této práci využívám toolkitu od společnosti OpenAi který se jmenuje Gym. Tento toolkit je výborný pro vývoj a porovnávání výkonu algoritmů pro posilované učení. Sám osobě obsahuje mnoho prostředí od CartPole (vyvažování převráceného kyvadla), přes hry jako je pong až po mnoho her pro konzoli atari 2600.

Cílem tohoto toolkitu je zpřístupnit více vývojářům koncept posilovaného učení. Nutnost implementace vlastního rozhraní s emulátory jednotlivých her bývala totiž značná bariéra vstupu pro kohokoliv, kdo chtěl dělat cokoliv více než úplné základy posilovaného učení. Dalším cílem tohoto projektu je do jisté míry standardizovat rozhraní jednotlivých prostředí, každé řešení problému posilovaného učení může mít drobné rozdíly v definicích prostředí. Například počty povolených akcí mohou drasticky ovlivňovat výkon jednotlivých algoritmů posilovaného učení.

Gym Retro Gym Retro je potom nadstavba toolkitu Gym. V základu Gym Retro obsahuje více než tisíc starých her, k tomu využívá vícero simulátorů podporujících Libereto API. Bohužel i pro prostředí pro které obsahuje integraci si je třeba nahrát vlastní ROM paměti.

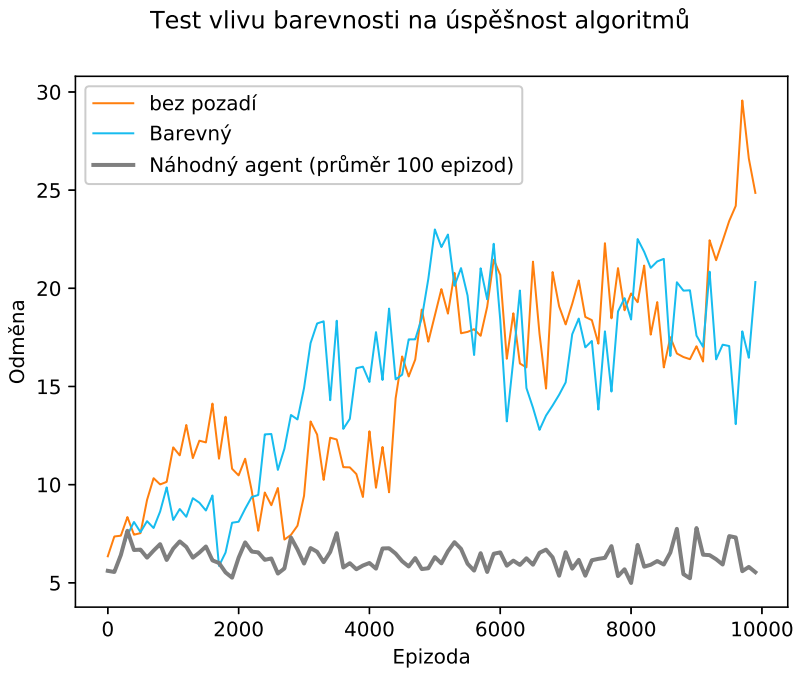
5.2 Úpravy vybraných prostředí

Pro implementaci algoritmů pro přenos zkušeností jsem si vybral kategorii her známou jako "Sestřel je všechny"z anglického "shoot 'em up". Jde o poměrně jednoduchou kategorii her, kde hráč ovládá stíhačku a snaží se sestřelit co nejvíce nepřátelských stíhaček, přičemž se vyhýbá jejich střelám. Do této kategorie se řadí například hry jako "Air raid", "Airstriker", "Demon attack"nebo jedna z nejznámějších her pro konsoli atari 2600 - Space invaders.

Takto jednoduchou kategorii jsem zvolil z toho důvodu, že v oblasti přenosu zkušeností v posilovaném učení dle mého vědomí ještě nebyl zveřejněn žádný významný výzkum, tudíž není moc od čeho se odrazit a vše musím testovat od začátku.

Akce prostředí Všechna vybraná prostředí z toolkitu Gym mají v základu sadu akcí které dávají v daném prostředí smysl. Oproti tomu prostředí z Gym Retro používá multibinární akce, to znamená že jako akce slouží vektor binárních hodnot, který vyjadřuje, jaká tlačítka konzole jsou v daném okamžiku aktivní. Vzhledem k tomu, že s tímto přístupem počet možných akcí roste exponenciálně s počtem tlačítek, jsem se rozhodl tento prostor akcí zmenšit. A to konkrétně na stejnou velikost jako u základních atari prostředí z Gymu, to znamená že jsou povolené jen následující akce: Letět vpravo, Letět vlevo, střílet, letět vpravo a střílet, letět vlevo a střílet, nedělat nic. Pro některé experimenty zakazuji pro jednoduchost 4. a 5. možnost.

Úpravy vstupů do modelu Jako vstupy pro trénování modelů používám přímo 2D obrázky z her. Velikost těchto vstupů potom zmenšuji na rozměry 64x64 pixelů. Následně, kvůli tomu, že používám přenos zkušeností, také převádím tyto obrázky do odstínů šedi, abych zvýšil podobnost jednotlivých domén učení. Abych zjistil, jestli konverze do odstínů šedi má výrazný význam na úspěšnost učení jsem použil experiment v prostředí hry 1942 pro konzoli Nintendo Entertainment System (NES).



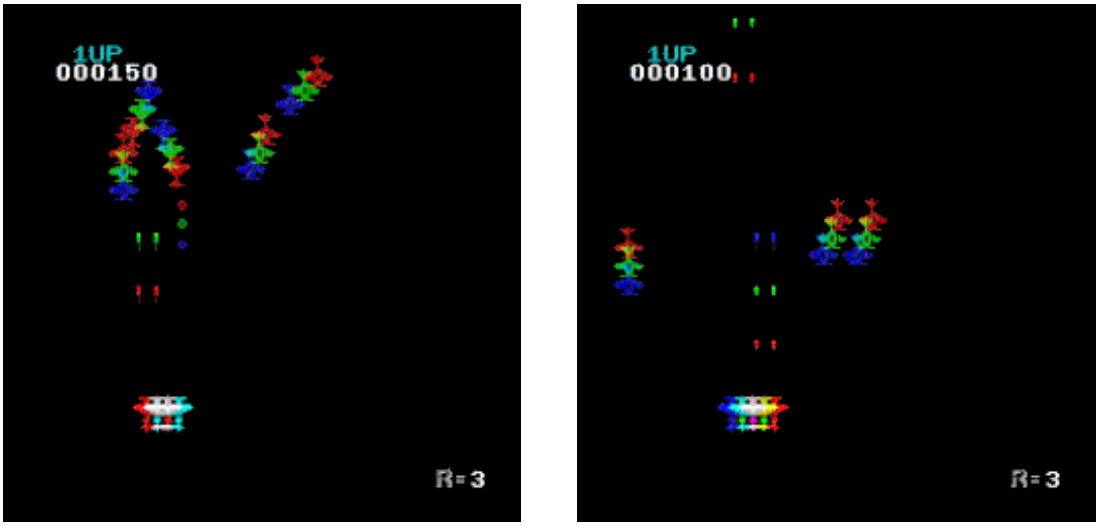
Obrázek 5.1: Cílem tohoto experimentu bylo zjistit, Jaký vliv má používání černobýlých oproti barevným obrázkům jako vstu pro neuronovou síť.

Z experimentu plyne, rychlost ani úspěšnost učení nezávisí na tom, jaké vstupy používáme. Jedinou metriku kterou zde můžu použít je rozdíl v době trénování co se GPU času týče. Pro černobílé obrázky trénování trvalo 6 hodin a barevné trvalo 18, což je očekávaný nárůst, protože velikost každého obrázku se zvýšila na trojnásobek. Vzhledem k tomu že

trénování na barevných obrázcích trvá několikanásobně delší dobu, než na černobílých, a nevykazuje žádné významné benefity, zbytek testů budu provádět s černobílými obrázky, které také používám v experimentech pro sdílení zkušeností.

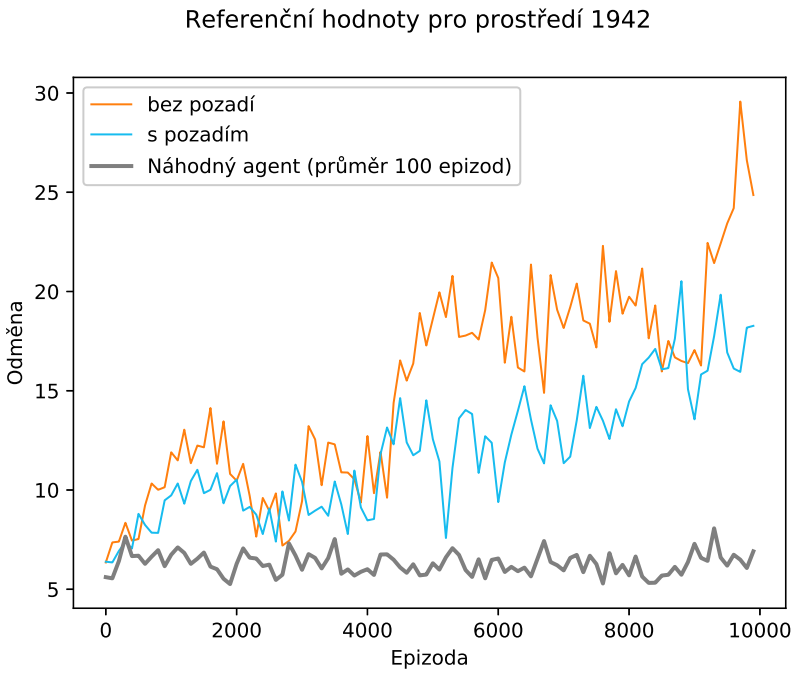
Jeden snímek herní obrazovky většinou nestačí pro přesné popsání toho, co se ve hře děje. Neobsahuje totiž žádné informace o pohybu jednotlivých objektů a to často negativně ovlivňuje výkon algoritmů. Abych řešil tento problém, používám jako vstup neuronové sítě místo posledního snímku poslední 3 snímky. Díky tomu algoritmus dostává více informací o prostředí v každém kroku. Samotné kroky učení trvají velmi dlouhou dobu oproti emulaci prostředí. Navíc není třeba aby agent volil novou akci v každém novém snímku. Pokud chtěl v minulém snímku letět doprava, pravděpodobně bude chtít letět doprava i ve snímku následujícím. Paměť se vzpomínkami se takto velmi rychle zaplní značně podobnými snímky a zpomaluje se tak učení. Z toho důvodu jsem se rozhodl použít ještě jednu úpravu prostředí a tou je stochastické přeskočení snímků. To funguje tak, že agent určí akci kterou chce provést jen každých n snímků, a ta se potom provádí ve všech přeskočených snímcích. Tato akce je navíc s určitou pravděpodobností ignorována v prvním z přeskočených snímků a provádí se místo ní předchozí akce. Tato metoda byla použita v OpenAi baselines [4], kde pro trénování používají hodnotu $n = 4$. Pro každé testované prostředí jsem zkoušel jinou hodnotu a zvolil takovou, aby granularita akcí byla dostatečná pro manévrování.

Přeskakování snímků spolu s používáním 3 posledních snímků znamená, že jako vstup pro mojí neuronovou síť používám data agregovaná z posledních někdy až 18 snímků, Příklad takového vstupu potom můžeme vidět znázorněný na obrázku 5.2.

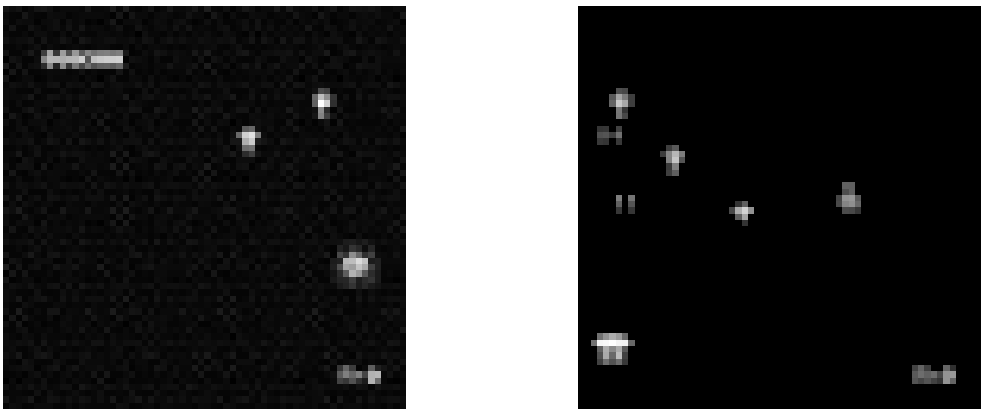


Obrázek 5.2: Tyto obrázky znázorňují výsledný vstup do neuronové sítě. Jednotlivé černobílé vstupní snímky jsou dle času pro znázornění obarveny na červenou zelenou a modrou barvu, kde modrý je aktuální snímek a červený je nejstarší.

Také jsem zkoušel vliv mírně složitějších než čistě černých pozadí v prostředích, pro tento test jsem také použil hru 1942. Data z experimentu říkají, že síť učená na prostředí s vyříznutým pozadím dosahuje konzistentně vyšší odměny, než síť učená bez úprav vstupního obrazu a to někdy o více než 25% 5.3. A to i přesto, že pozadí představuje pro člověka jen mírný šum. Porovnání je vidět na obrázku 5.4.



Obrázek 5.3: Cílem tohoto experimentu bylo zjistit, jestli a popřípadě jaký mají mírně složitější pozadí vliv na rychlost trénování algoritmu DQN.



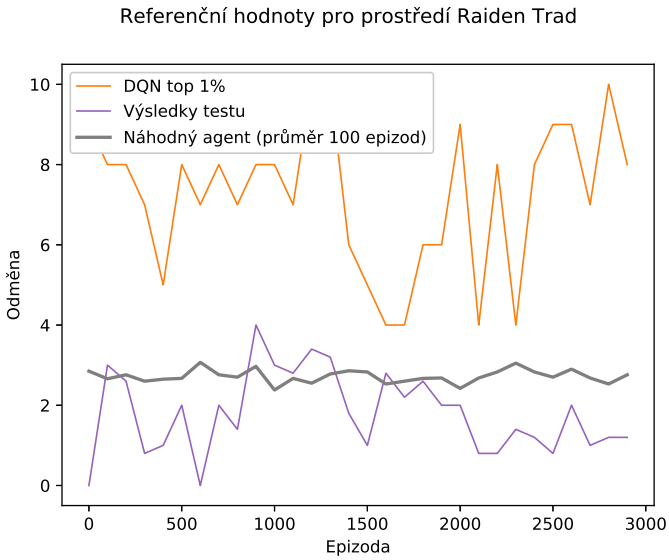
Obrázek 5.4: Porovnání mezi prostředím 1942 s pozadím a bez pozadí. Je vidět, že co je pro člověka jen nevýrazný šum, může mít pro neuronovou síť viditelně negativní dopad

Limitace životů Ve většině vybraných her má hráč na začátku několik životů. Rozhodl jsem se tento počet snížit na 1 abych zajistil, že agentovi bude vadit umírat. Intuice za tímto rozhodnutím je, že když agent může ztratit až 2 životy před koncem epizody, je pro něj složitější identifikovat situace, které vedly k tomu, že epizoda skončila. Váže se to k jednomu z fundamentálních problémů posilovaného učení - Problému zpožděné odměny (en. Credit assignment problem).

Nakonec kvůli tomu, jak se počítá ztrátová funkce je dobrý nápad nějakým způsobem normalizovat odměny. Běžně totiž tato prostředí dávají odměnu 100 za zničení jedné nepřátelské stíhačky. Takto velké odměny by způsobovaly velká čísla v odhadech Q hodnot modelu, a jak již bylo zmíněno, velká čísla a neuronové sítě moc nejdou dohromady. Proto aby nevznikaly aproximační chyby používám odměnu která je oříznutá. To znamená, že pozitivní odměny počítám jako 1, negativní odměny počítám jako -1 a jiné odměny počítám jako 0. Tím se dále zvyšuje stabilita učení. [17] Tato taktika také umožňuje používat stejný faktor učení v několika různých prostředích což je klíčové zejména u experimentů na přenos zkušeností.

Další úpravy jsou specifické pro prostředí a budu o nich mluvit spolu s detaily o daných prostředích.

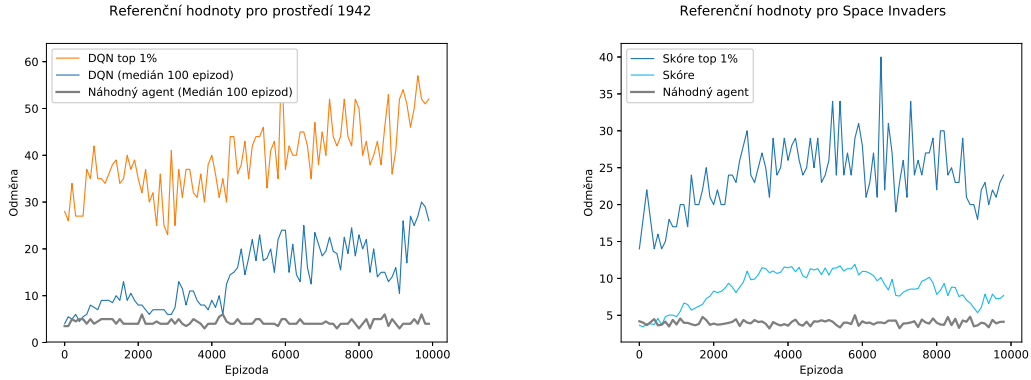
Limitace při výběru prostředí Konkrétně jsem si vybral několik her pro konzoli Atari a vybraná další prostředí. Původně jsem chtěl využít rozmanitější škálu prostředí ale experimenty na nich selhávaly. Problémy s novějšími konzolemi jako například Genesis od SEGA je, že hry mají velmi často pozadí které je příliš komplikované (například stíhačka prolétá nad městem a podomně) a často se v takových případech stává, že není vůbec vidět, co se děje. U mnoha takových prostředí se může stát, že agent se nikdy nic nenaučí a konzistentně dosahuje zásadně horších výsledků než náhodný agent. Příkladem takového prostředí je například hra Raiden Trad.



Obrázek 5.5: Výsledky experimentu pro prostředí Raiden Tard. I po 3000 epizodách trénování je agent ovládaný sítí DQN konzistentně horší než agent, který provádí zcela náhodné akce. Na druhém obrázku je ukázka prostředí pro ilustraci

5.3 Referenční hodnoty pro jednotlivá prostředí

Space Invaders První prostředí které Splňovalo má kritéria jsou Space Invaders. Zde si můžeme všimnout že střely a objekty jsou často vykreslovány pouze každý druhý snímek. To je chyba která se objevuje u více prostředí od konzole Atari2600. Tuto nepříjemnou záležitost, při které by se mohlo stát, že agent vůbec neuvidí na něj letící střelu jsem se rozhodl řešit tak, že jednoduše sčítám hodnoty posledních 2 snímků. Dalším zvoleným prostředím je hra 1942. Je to prostředí na kterém jsem dělal již výše zmíněné experimenty.



Obrázek 5.6: Experiment pro získání referenčních hodnot prostředí 1942 a Space invaders.

Z experimentů vidíme, že algoritmus DQN konzistentně překonává náhodného agenta. Z chování agenta plyne, že je mechanicky velmi zdatný a umí se vyhýbat střelám a mířit dopředu. Pro vyřešení prostředí Space invaders je ovšem potřeba jistá strategie a dlouhodobé plánování, které se síť nenaučila.

Jako další prostředí používám hru Air Striker. Jedná se do další jednoduchou hru, pozadí je černé a vše čemu je třeba se vyhýbat je poměrně dobře vidět. Pro toto prostředí nebylo třeba použít žádné další úpravy a agent DQN velmi rychle dokáže překonat náhodného agenta.

Následující 2 prostředí jsou podobná všem předchozím, hlavním rozdílem je, že hráč má k dispozici další akce. Jedná se o hru Assault,

Obrázek 5.7: Experiment pro získání referenčních hodnot prostředí Phoenix a Assault

kde pro úspěšné projití hrou je potřeba, aby agent byl schopný střílet do stran a o hru Phoenix kde má agent k dispozici štít který udělá jeho stíhačku nezničitelnou, tyto akce při trénování ale nepovoluji, abych později s těmito modely mohl provádět pokusy na přenos zkušeností.

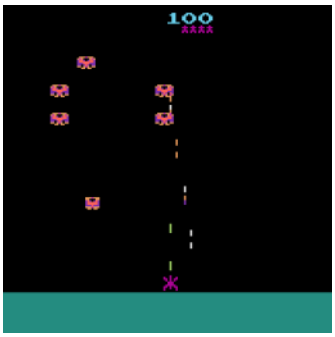
U hry phoenix je omezení akcí vidět velmi dobře, hra je totiž bez štítu velmi těžká a aby agent dokázal pokročit, musel by být faktor prozkoumávání vypnut nebo snížen. Když se podíváme na průměrné hodnoty odměny vycházející z testů (Greedy politika, každých 100 kroků 10 epizod) tak jsou konzistentně vyšší, než průměry při trénování. To značí, že ho při učení často zabíjejí náhodné akce, kterými se agent řídí kvůli prozkoumávání.

Dále používám prostředí hry demon attack. Tato hra je z vizuálního hlediska velmi podobná hře Assault. S tím, že není potřeba střílet do stran a nepřátelé se pohybují podstatně rychleji a náhodněji.

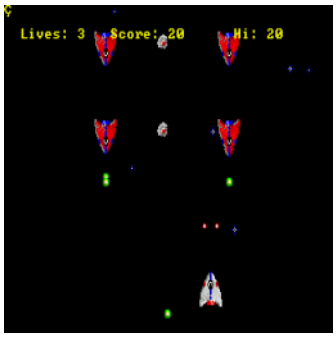
(a) Space Invaders

(b) Demon Attack

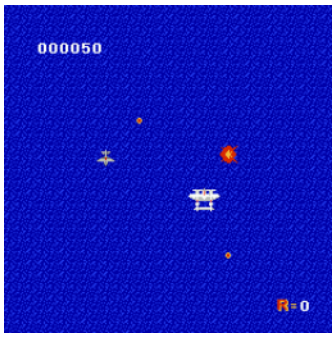
(c) Assault



(d) Phoenix



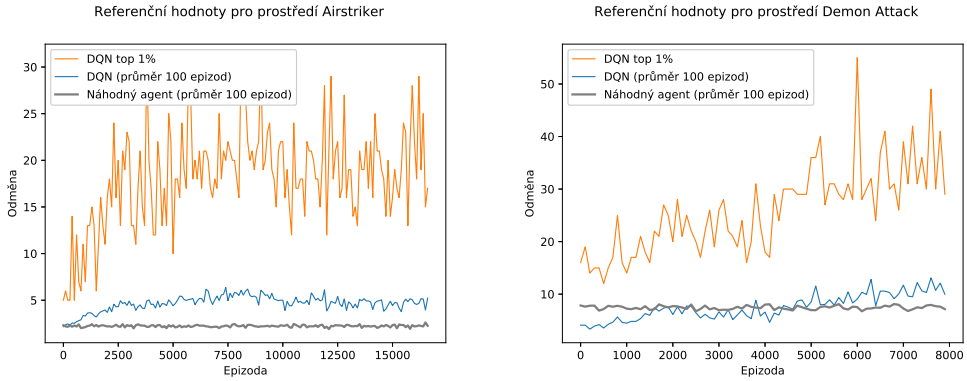
(e) Air Striker



(f) 1942

Obrázek 5.8: Obrázky ze všech použitých prostředí

Jednotlivá prostředí jsem se snažil vybírat tak, aby byla na jednu stranu podobná, ale na druhou stranu aby každé vyžadovalo jiné dovednosti pro úspěšné řešení. Například ve hře space invaders je potřeba značná úroveň strategického plánování aby bylo možné zničit všechny stíhačky. Hra demon attack potom testuje jak dobře se agent dokáže naučit prokličkovat mezi střelami a zničit přitom nepřítele.



Obrázek 5.9: Experiment na získání referenčních hodnot pro prostředí Airstriker a Demon Attack

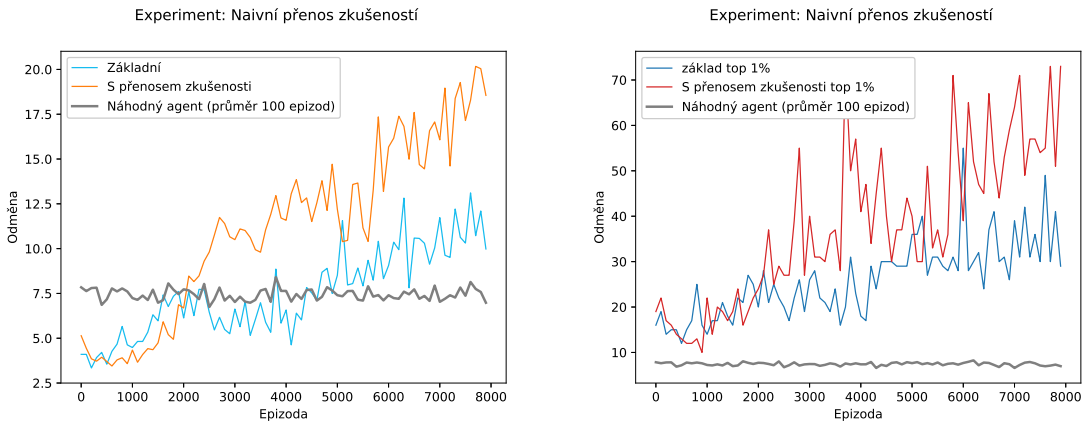
Kapitola 6

Experimenty sdílení zkušeností pro DQN

V této kapitole se zaměřuji na metody přenesení zkušeností používané na rozpoznávání obrazu a podobné úkoly. Tyto metody následuje aplikuji na model učící se pomocí zpětnovazebního učení.

6.1 Naivní přístup k přenosu zkušeností

Tomuto přístupu se také říká vývoj modelu [8]. A funguje tak, že využívá již existující model, který přetrénuje v nové doméně a doufá že se nic nepokazí. Pro tento experiment používám nejdříve agenta trénovaného pro prostředí assault a pokouším se jej přetrénovat na prostředí demon attack. Tato prostředí jsou si značně podobná, takže by teoreticky neměl model mít problém se orientovat v novém prostředí.

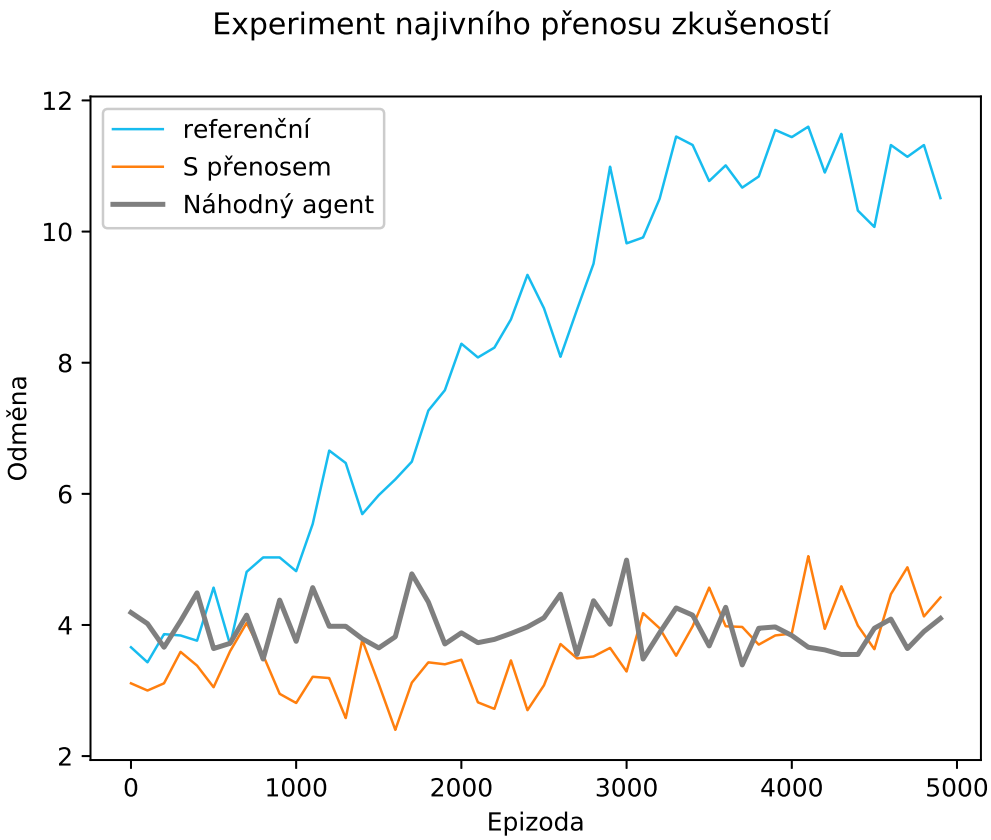


Obrázek 6.1: Experiment využívající naivní přístup k přenosu zkušeností podobných prostředí

Výsledky tohoto experimentu značí, že i velmi naivní přístup k přenosu zkušeností může mít dobré výsledky. Z grafů můžeme vyčíst, že síť nejprve plně využívala svých znalostí z předchozího prostředí. Tím dokázala bez jakéhokoliv přetrénování překonat náhodného agenta. Postupně se ale kvůli neočekávané změně v dosahovaných odměnách začala zhoršovat. Kolem epizody 2000 můžeme vidět, že agent stále nezapomněl svoje předchozí zkuše-

nosti a náhle překonal náhodného agenta a referenční algoritmus. Po 3000 epizodách tento přístup dokonce dosahuje lepších odměn, než referenční algoritmus po 8000 epizodách. Z dat můžeme vyčíst, že naivní způsob k přenosu zkušeností zvedl průměrnou odměnu za 8000 epizod o 51%. Když beru v potaz pouze odměny posledních 2000 epizod, můžeme pozorovat odměny které jsou o 128% vyšší než u náhodného agenta, a o 62% vyšší než u referenčního modelu. To znamená o 218% větší nárůst skóre oproti náhodnému agentovi než základní model. Maximální dosažená skóre v průběhu 8000 epizod jsou potom 24 pro náhodného agenta, 51 pro referenční model a 72 pro model s přenesením zkušenosti, kde 1 bod = zničení jedné stíhačky.

Jako další potom zkouším model trénovaný na 1942 přetrénovat na prostředí Space Invaders, tato prostředí jsou značně rozdílná a experiment by tento fakt také odráží.

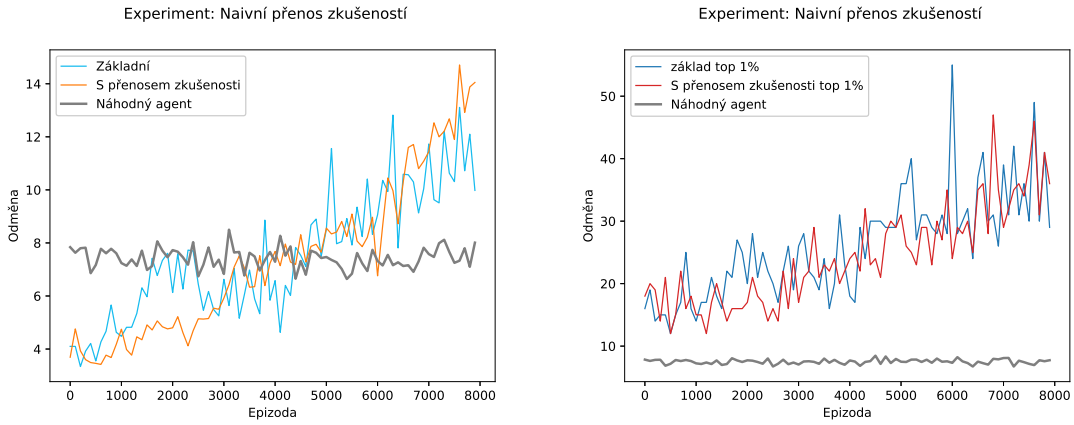


Obrázek 6.2: Výsledky experimentu naivního přenosu z prostředí Phoenix do prostředí Space Invaders.

Jde o vzorový příklad negativního přenosu zkušenosti. Algoritmus až po přibližně 3500 epizodách začíná dosahovat průměrného skore srovnatelného s náhodným agentem. Tímto experimentem jsem demonstroval ostré kontrasty výsledků které můžeme očekávat při naivním přístupu k přenosu zkušeností v posilovaném učení.

6.2 Využití metody přenesení rysů domén

V tomto příkladu využívám metodu přenesení rysů domén. Vzhledem k tomu, že jako vstup modelu slouží obrazová data, měly by se konvoluční vrstvy na začátku modelu naučit rysy které jsou relevantní pro řešení daného úkolu. Teorie říká, že pokud budou rysy dvou prostředí podobné, a úkoly téměř totožné, měla by tato metoda fungovat dobře. Tuto metodu implementuji tak, že využívám konvolučních vrstev modelu natrénovaném na zdrojovém prostředí, a zamykám je, trénování potom probíhá jen na plně propojených vrstvách, což by měla být část sítě zodpovědná za přiřazování odměn jednotlivým rysovým mapám.



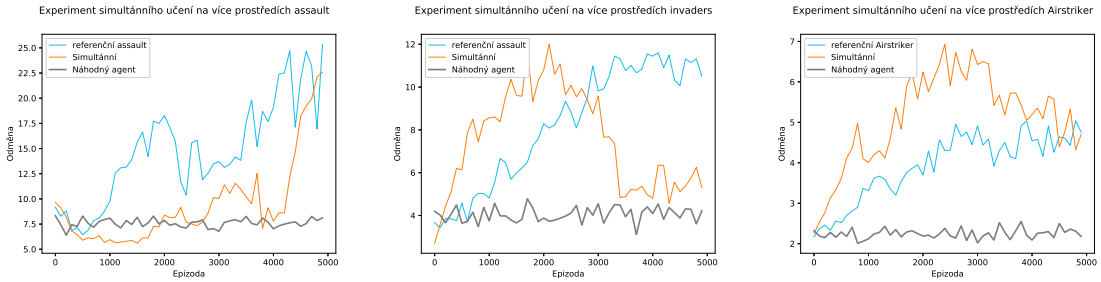
Obrázek 6.3: Výsledky experimentu na využití metody přenosu rysů domén.

Tato metoda předpokládá, že znalost rysů podobné domény pomůže algoritmu lépe řešit úkol, protože při učení se stačí naučit pouze váhy plně propojených vrstev. Výsledky experimentu ale značí že znalost rysů předchozí domény bez znalosti jejich hodnocení plně propojenými vrstvami není dostatečné. Ani při použití modelu z naivního přenosu tato metoda žádným výrazným způsobem nezlepšila schopnost algoritmu řešit úkol posilovaného učení, a průměrná odměna v průběhu 8000 epizod vzrostla o pouhých 0.41%. V posledních 2000 epizodách algoritmus s přenesením zkušeností získal o 4.5% větší odměny než referenční algoritmus. Tento nárůst je ale zanedbatelný v rámci chybovosti.

6.3 Simultánní učení ve více prostředích

V tomto experimentu prozkoumávám možnosti simultánního učení na více prostředích s použitím algoritmu DQN. Používám stejný model jako pro všechny ostatní experimenty doposud. Problémy tohoto přístupu sdílení zkušeností bývají většinou to, že je nevyvážený přístup k datům z jednotlivých domén. [13] Tento problém se projevuje tak, že v trénovací sadě dominují data z jedné domény nad všemi ostatními. Pokud bych tedy v tomto případě měl jedno prostředí ve kterém se agentovi dařilo významně lépe, model by se učil primárně z dat tohoto prostředí. Abych tomuto zabránil, používám několik různých pamětí se vzpomínkami. Konkrétně jednu pro každé prostředí, a při učení vybírám data z těchto pamětí rovnoměrně.

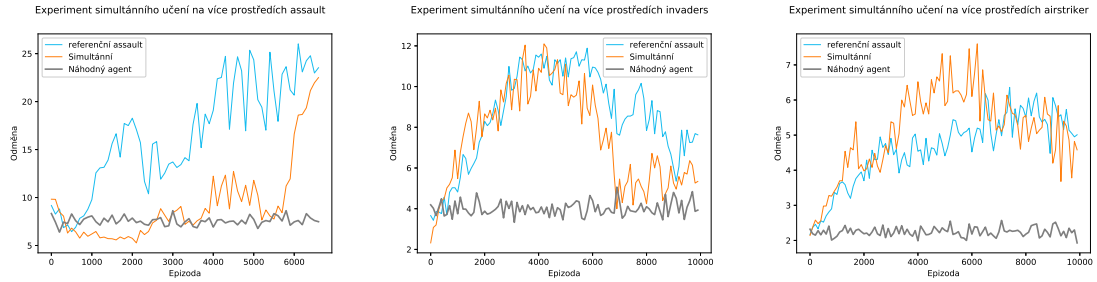
Konkrétně toho dosahuji tak, že po každém kroku v kterémkoliv z prostředí se podle modula 3 tohoto kroku učím z jedné z pamětí. Aby vzpomínky v jednotlivých pamětech nebyly staré, pravidelně každou epizodu střídám prostředí pro které generuji vzpomínky. Celkově experiment běžel přeš 15000 epizod, 5000 pro každé prostředí. Na obrázcích 6.4 je



Obrázek 6.4: Experiment na simultánní učení více prostředí.

znázorněno porovnání učení tohoto experimentu v epizodách jednotlivých prostředí oproti referenčním algoritmům pro toto prostředí za 5000 epizod.

Z experimentů vyplývá že pokud chceme více různých modelů pro různá prostředí, metoda simultánního učení může přinést velmi pozitivní výsledky. Výše uvedené obrázky ale neobsahují všechna data. Pokud nasbíraná data z tohoto experimentu přiložíme nad historie všech 10000 trénovacích epizod každého prostředí, jak můžeme vidět na obrázku 6.5 dostaneme kompletnější představu o úspěšnosti algoritmu. Zde je vidět že ve všech prostředích je v nějakém bodě algoritmus simultánního učení alespoň stejně dobrý, jako referenční algoritmus daného prostředí. V prostředí assault, odměn blízkých referenčním sice začal dosahovat až na konci trénování, nicméně dosahoval jich konzistentněji (méně často umřel v první vteřině prostředí) než u referenčního algoritmu. V prostředí airstriker byl kolem epizody 3000 až o 20% lepší než referenční algoritmus a v prostředí space invaders měl na začátku trénování lepší konvergenci (následně ale ztratil schopnost hrát na takto vysoké úrovni). Také je vidět, že relativně vysoké hodnoty odměn v prostředí assault oproti ostatním prostředím způsobily, že algoritmus začal ztrácet naučené taktiky na ostatní prostředí.



Obrázek 6.5: Experiment na simultánní učení více prostředí.

Možnosti budoucího vývoje Jako návaznost na tuto práci by bylo dobré otestovat různé metody normalizace pro metodu učení několika prostředí najednou. A to tak, aby prostředí svými relativně vysokými odměnami vůči ostatním prostředí nemělo negativní dopad na jejich učení, jako tomu bylo v případě prostředí Assault v mém experimentu. Dále by bylo vhodné prozkoumat možnosti využití již předtrénovaných konvolučních sítí na rozpoznávání obrazu a použít je pro extrakci rysů z prostředí.

Kapitola 7

Implementační detaily

Celý projekt jsem psal v programovacím jazyce python. Pro práci s neuronovými sítěmi jsem použil knihovnu Keras. Keras je vysokoúrovňové API psané v pythonu které zpřístupňuje uživateli sílu zvoleného backendu (v mém případě TensorFlow) daleko přívětivějším způsobem než kdyby s tímto backendem musel uživatel pracovat přímo. Umožňuje relativně snadné prototypování neuronových sítí. Zvolil jsem Keras pro jeho snadnost používání a Tensorflow pro možnost provádění výpočtů učení na grafické kartě, tato funkcionalita drasticky zrychluje matematické operace v neuronových sítích. Trénování které by na CPU mohlo trvat několik dní dokáže výkonná grafická karta zvládnout i za několik hodin.

Pro úpravy jednotlivých prostředí používám návrhový vzor adaptér, ten umožňuje změnit funkcionalitu třídy bez toho aby bylo třeba měnit její kód nebo práci s ní. Tento návrhový vzor používám na úpravy akcí, odměn a obecně všeho, co souvisí s prostředím. Všechny tyto adaptéry, a to i pro prostředí, která jsem se nakonec rozhodl nepoužít, se nacházejí v souboru "wrappers.py". V tomto souboru se nacházejí adaptéry pro škálování obrazu daného prostředí, různé ořezy nepodstatných informací a podobně. Dále tento soubor obsahuje stochastické přeskočení obrázku, ořezávání odměn, sčítání snímků pro atari hry, dále adaptér na převod akcí prostředí z multibinárního prostoru na diskrétní prostor a nakonec mapování akcí jednotlivých prostředí.

Definice neuronové sítě a učícího modelu se nachází v souboru DQN.py, ve kterém se nachází stejnojmenná třída. Tato třída dále zjednodušuje práci s modelem. Obsahuje inicializaci, ukládání a načítání modelu, učící model a paměť se vzpomínkami. Nacházejí se v něm i definice jednotlivých neuronových sítí použitých při experimentech.

Kód pro jednotlivé experimenty je potom ve složkách podle prostředí v jakém tyto experimenty byly prováděny. Kód provádění simulace experimentů je v podstatě 1:1 kopie algoritmu DQN popsaného v 2.

Každá složka obsahuje skript pro spuštění trénování, skript pro načtení modelu a následné renderování videa, jak agent hraje hru a předtrénovaný model pro dané prostředí. Pro zajímavost jsem ponechal soubory obsahující odměny z testů a odměny z jednotlivých epizod trénování.

Co se hyperparametrů algoritmu učení týče, jako rychlost učení používám hodnotu $5 \cdot 10^{-4}$, tuto hodnotu jsem převzal z kódu gym baselines [4]. Jako faktor prozkoumávání používám exponenciálně se zmenšující hodnotu. Nejprve je nastavena na 1 a po každé epizodě se vynásobí konstantou 0.999. Minimální hodnota faktoru prozkoumávání je nastavena na 0.1. Této minimální hodnoty dosahuje kolem epizody 2300. Některá prostředí používají větší faktor prozkoumávání. Volil jsem jej v závislosti na tom, jak je manévrování stíhačky citlivé na náhodné vstupy. Při učení provádím 4 kroky simulace mezi každou etapou učení

na náhodné sadě 32 vzorků z paměti se vzpomínkami. Každých 1000 kroků potom obnovuji cílovou stabilizační síť na nové hodnoty. Hodnotu stárnutí jsem převzal z článku "Playing Atari with Deep Reinforcement"[10] kde byla použita hodnota 0,99.

U některých prostředí na začátku každé epizody přeskakuji určitý počet snímků, to je z toho důvodu, že v těchto prostředích často dlouho trvá, než se začnou objevovat první nepřátelé, nedává tedy smysl se do té doby učit, a ukládat snímky do paměti si vzpomínkami. Toto vylepšení u některých prostředí drasticky zrychluje průchod jednotlivými epizodami a způsobuje, že se model učí daleko rychleji. Jednak dokáže lépe využít dostupná data z epizody a jednak není třeba provádět kroky učení při této době. Například učení modelu pro prostředí Airstriker po této změně procházelo přes epizody ze začátku až o 1000% rychleji a v učení dokázali překonat náhodného agenta za poloviční počet epizod. O data pro tento experiment jsem přišel kvůli technickým problémům a již mi nezbývá čas jej provést znovu.

Jediný algoritmus který je výrazně rozdílný od ostatních je algoritmus simultánního učení na více prostředích. V tomto algoritmu provádím v každé epizodě simulaci jiného prostředí, a po každém kroku simulace provádím učení z jiného prostředí. Tímto způsobem zajišťuji, aby data nebyla zaujatá vůči nějakému z prostředí. Paměť se vzpomínkami nejprve inicializuji náhodnými akcemi pro každé prostředí a následně spustím samotný algoritmus učení. Každou epizodu pomocí modula této epizody střídám mezi simulováním jednotlivých prostředí. Po každém kroku simulace potom opět podle modula kroku vyberu ze které vzpomínkové paměti se agent učí a provádím učení na 10 vzorcích. Cílovou stabilizační síť potom obnovuji každých 2000 kroků.

Kapitola 8

Závěr

V této práci jsem využil koncepty a mechanizmy přenosu zkušeností mezi neuronovými sítěmi v oblasti posilovaného učení. Rozhodl jsem pro řešení využít algoritmu DQN. Z experimentů vyplývá, že používání těchto metod pro tuto oblast strojového učení není intuitivní ale může přinést velmi pozitivní výsledky.

Použitím naivního přístupu k přenosu zkušeností pro model trénovaný algoritmem DQN jsem demonstroval více než 50% nárůst průměrných odměn po dobu 8000 trénovacích epizod oproti referenčnímu modelu a 128% nárůst v posledních 2000 epizodách.

Dále jsem demonstroval úskalí naivního přenosu zkušeností při experimentu 6.2 kde jsem se snažil vybrat rozdílná prostředí a ukázat tak negativní přenos. V tomto případě přetrénovaný model po většinu trénování hrál hůře než náhodný agent a trvalo mu přibližně 3500 epizod se mu vyrovnat.

Následně jsem předvedl neintuitivnost disciplíny přenosu zkušeností na experimentu přenesení rysů mezi doménami. Při použití pouze konvolučních vrstev modelu který při použití naivního přenosu dosáhl vynikajících výsledků, nedošlo k výraznému zlepšení učících vlastností algoritmu oproti referenčnímu modelu.

Nakonec jsem předvedl možnost učení modelu na více prostředích simultánně. Z tohoto experimentu plyne že pokud chceme natrénovat algoritmy pro více prostředí, je vhodné použít tento přístup, protože v určité fázi trénování byl tento algoritmus lepší než referenční algoritmus každého prostředí nebo dokonce překonal jeho nejlepší výsledky.

Tato práce mi pomohla naučit se pracovat s neuronovými sítěmi a implementovat vlastní algoritmy posilovaného učení. Práce poskytuje vhled do problematiky přenosu zkušeností v kombinaci se zpětnovazebním učením. Ukazuje benefity které můžeme získat používáním přenosu zkušeností a podobných metod, stejně jako rizika, která jsou s jejich používáním spojená.

Literatura

[1] Badreesh, S.: *Supervised Machine Learning: Classification*. 2018.
URL https://towardsdatascience.com/supervised-machine-learning-classification-5e685fe18a6d

[2] Contributors: *Huber loss univerzity*. 2015.
URL https://en.wikipedia.org/wiki/Huber_loss

[3] Contributors: *Reinforcement learning*. 2019.
URL https://en.wikipedia.org/wiki/Reinforcement_learning

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

[5] Dipanjan, S.: *Guide to Transfer Learning with Real-World Applications in Deep Learning*. 2018.
URL https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

[6] EECS, C. E.: *CS294-112 Záznamy z přednášek z Californské univerzity*. Září 2017.
URL https://www.youtube.com/playlist?list=PLkFD6_40KJIznC9CDbVTjAF2oyt8_VAe3

[7] freeCodeCamp.org, A.: *An introduction to Q-Learning: reinforcement learning* . 2018.
URL https://medium.freecodecamp.org/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc

[8] Jason, B.: *A Gentle Introduction to Transfer Learning for Deep Learning*. 2017.
URL https://machinelearningmastery.com/transfer-learning-for-deep-learning/

[9] Jonathan, H.: *RL introduction to Deep Reinforcement Learning*. 2018.
URL https://medium.com/@jonathan_hui/rl-introduction-to-deep-reinforcement-learning-35c25e04c199

[10] Mnih, V.; Kavukcuoglu, K.; Silver, D.: *Playing Atari with Deep Reinforcement*. Prosinec 2013.
URL http://arxiv.org/abs/1312.5602

[11] Pan, S. J.; Yang, Q.: *A Survey on Transfer Learning*. 2009.
URL https://www.cse.ust.hk/~qyang/Docs/2009/tkde_transfer_learning.pdf

[12] Pratt, L. Y.: *Machine Learning - Special Issue on Inductive Transfer*. Září 1997, [Retrieved 2017-08-10].
URL http://papers.nips.cc/paper/641-discriminability-based-transfer-between-neural-networks.pdf

[13] Rich, C.: *Multitask Learning*. 1998.
URL http://www.cs.cornell.edu/~caruana/mlj97.pdf

[14] Saenko, K.; Tzeng, E.; Hoffman, J.; aj.: *Deep Domain Confusion: Maximizing for Domain Invariance*. Prosinec 2014.
URL https://arxiv.org/abs/1412.3474

[15] Sebastian, R.: *Transfer learning - Machine Learning's next frontier*. 2017.
URL http://ruder.io/transfer-learning/

[16] Simonini, T.: *Improvements in Deep Q Learning*. 2018.
URL https://medium.freecodecamp.org/improvements-in-deep-q-learning-dueling-double-dqn-prioritized-experience-replay-and-fixed-58b130cc5682

[17] Volodymyr, M.; Koray, K.; David, S.; aj.: *Human-level control through deep reinforcement learning*. Únor 2015.
URL https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf

[18] Wang, T.; Huan, J.; Zhu, M.: *Instance-based Deep Transfer Learning*. 2018.
URL https://arxiv.org/abs/1809.02776v2

[19] Wikipedia: *Markov decision process*. 2019, [Date retrieved: 2 Feb. 2019 16:32 UTC].
URL https://en.wikipedia.org/w/index.php?title=Markov_decision_process&oldid=893936705

[20] Zirui, W.; Zihang, D.; Barnabas P, C., oczosand Jaime: *Characterizing and Avoiding Negative Transfer*. 2019.
URL https://arxiv.org/pdf/1811.09751.pdf