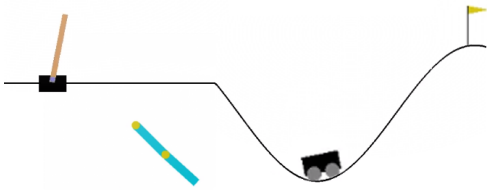
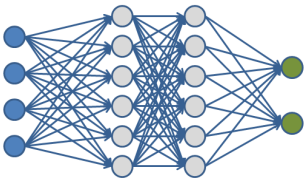
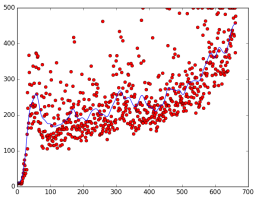
BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2018
http://excel.fit.vutbr.cz

Hraní her pomocí neuronových sítí

Petr Buchal

Abstrakt
Tato práce se zabývá zkoumáním algoritmu zpětnovazebního učení DQN, jeho vylepšeními a jejich vlivem na efektivitu učení. Využitím prostředí, která jako svůj popis poskytují místo obrazu vektor čísel, bylo dosaženo snížení doby trénování z několika dní na několik minut. Tato změna umožnila efektivně testovat různé varianty algoritmu DQN a zkoumat jejich účinnost. Za použití několika vylepšení bylo na všech prostředích dosaženo řádově v desítkách až stovkách procent lepších výsledků. Provedené experimenty mohou sloužit k lepšímu pochopení algoritmu DQN a jeho vylepšení.
Klíčová slova: neuronové sítě — zpětnovazební učení — DQN

Přiložené materiály: Stáhnutelný kód — Demonstrační video

*xbucha02@stud.fit.vutbr.cz, *Fakulta informačních technologií, Vysoké učení technické v Brně*

1. Úvod

Tato práce se zabývá algoritmem zpětnovazebního učení DQN [1] a jeho vylepšeními. Cílem DQN je vytvořit agenta, který se v libovolném prostředí bude schopen naučit inteligentnímu chovaní podle reakcí okolí ve formě odměn, jde tedy o formu umělé inteligence.

Historie má několik významných milníků, které jsou spjaty s uvedením agentů, kteří dokázali plnit konkrétní úlohu v konkrétním prostředí. Pravděpodobně nejznámějším takovým milníkem je vítězství počítače Deep Blue nad Garri Kasparovem v šachové partii v roce 1997 [2]. Tehdy se jednalo o agenta specializovaného na hraní šachů, tedy nic jiného než hrát šachy nedokázal. Byl nepochybně průlomem v umělé inteligenci (AI), ale k obecné umělé inteligenci (AGI) [3] měl daleko.

Obecnou umělou inteligencí rozumíme agenta, který dokáže úspěšně vyřešit jakýkoli problém, který dokáže úspěšně vyřešit člověk. Významný krok k vytvoření AGI udělala firma DeepMind v roce 2013, když vytvořila právě algoritmus DQN, jenž dokázal úspěšně hrát různé Atari hry, aniž by se specializoval na každou hru zvlášť.

Algoritmus DQN má dva problémy, které spolu úzce souvisí. Prvním problémem je relativně pomalé učení, které u Atari her trvá běžně několik dní, druhým problémem je poté obtížné nastavení hyperparametrů, kterých je nemalé množství a které naprosto kriticky ovlivňují úspěšnost a dobu učení.

Pokud jedno trénování agenta trvá několik dní, je problém algoritmus rozsáhleji zkoumat. Místo snímků obrazovky ale může neuronová síť na vstup dostávat vektor čísel přímo popisující stav prostředí. Tato úprava umožní odstranění konvolučních vrstev z neuronové sítě a z několika miliónů parametrů bude mít síť najednou několik stovek, s čímž se i doba trénování zkrátí z několika dní na několik jednotek až desítek minut. Toto zkrácení doby učení poté dovoluje provést daleko větší množství testů a umožňuje lepší pochopení chování algoritmu v různých prostředích.

Algoritmus 1 DQN [1]

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

2. Algoritmus a jeho vylepšení

DQN vychází z algoritmu Q-učení [4]. Při použití Q-učení se agent učí pohybovat v prostředí na základě předpovědí tzv. Q-funkce. Ta předpovídá očekávaný užitek z provedení konkrétní akce v konkrétním stavu tzv. Q-hodnotu. Akci s největší Q-hodnotou poté agent provede. Problémem Q-učení je, že pro každý stav a v něm možnou akci musí mít uložený záznam. Jejich počet činí jen pro piškvorky o velikoti 3x3 skoro 27 000 záznamů [5]. Když se poté vezme v úvahu ještě to, že by agent měl při učení všech 27 000 záznamů aktualizovat, stane se z Q-učení nepříliš použitelná metoda pro složitější prostředí. DQN problém s ukládáním záznamů řeší tak, že je neukládá. Místo toho veškeré Q-hodnoty aproximuje pomocí neuronové sítě. Ta se učí z paměti se vzpomínkami, ve které jsou uloženy stavy, kterými agent prošel. Celý pseudokód DQN se nachází v algoritmu 1.

Algoritmus DQN byl navržen v roce 2013 [1], od té doby bylo představeno několik vylepšení, která až v řádu stovek procent vylepšují jeho efektivitu. Těmito rozšířeními se zabývají následující odstavce.

2.1 Cílová síť

V každém kroku DQN se mění váhy neuronové sítě, což způsobuje, že hodnoty odhadů kvality akcí uložených v paměti jsou nestabilní. Tato nestabilita může při učení neuronové sítě způsobit, že odhady kvality akcí začnou divergovat a naučená síť se destabilizuje. Ve snaze zmírnit toto riziko se do algoritmu přidává druhá tzv. cílová neuronová síť [6], která se používá pro odhad kvality akcí během trénování. Váhy cílové sítě jsou stabilnější díky způsobu jejich aktualizace, což má kladný vliv na stabilitu hodnot odhadů kvality akcí. Používají se dva způsoby aktualizace vah v cílové síti. Buď to se jednou za delší dobu zkopírují váhy z primární sítě do cílové úplně anebo se v každém kroku vezme zlomek vah z primární sítě se zlomkem vah z cílové sítě a jejich součet utvoří nové váhy pro cílovou síť.

2.2 DDQN

Vanilla verze DQN občas nadhodnocuje kvalitu určitých akcí. Problém nastává v okamžiku, kdy nejsou nadhodnoceny všechny akce stejnou měrou, síť poté nemůže konvergovat ke skutečným kvalitám akcí [6].

Tento problém se snaží eliminovat algoritmus DDQN (dvojité DQN) [7]. Princip tohoto vylepšení je jednoduchý, při trénování vybírá akce primární neuronová síť, ale odhad kvality akce dodává cílová síť.

2.3 Duální neuronová síť

Výstup originální architektury neuronové sítě algoritmu DQN říká, jak dobré je provést konkrétní akci v konkrétním stavu. Tuto skutečnost můžeme vyjádřit funkcí $Q(s,a)$. Cílem duální architektury neuronové sítě je zpřesnění výstupních hodnot této funkce, skrze její rozložení na dvě jiné funkce [8]. Na funkci hodnoty stavu $V(s)$, která určuje, jak dobré je být v daném stavu a na funkci výhody akce $A(s,a)$, která sděluje, jak kvalitní je daná akce oproti ostatním akcím v daném stavu. V rovnici 1 je zapsán matematický vztah zmíněných funkcí [6].

$$Q(s,a) = V(s) + A(s,a) \tag{1}$$

Architektura duální neuronové sítě tedy po zpracování vstupních dat dál počítá odděleně funkci hodnoty stavu $V(s)$ a funkci výhody akce $A(s,a)$. Jejich hodnoty jsou poté zkombinovány na poslední vrstvě neuronové sítě, což vede ke zpřesnění odhadů kvality akcí.

2.4 Paměť s prioritními vzpomínkami

V původní verzi algoritmu DQN mají všechny vzpomínky při výběru trénovacího vzorku z paměti stejnou priotitu. Během takového výběru nevyužíváme skutečnosti, že z některých vzpomínek se může agent učit více než z jiných [9].

Vypočítat informaci o tom, ze kterých vzpomínek se agent může naučit nejvíce, můžeme rozdílem odhadů kvality nejhodnotnějších akcí podle sítě primární a cílové. Čím větší tento rozdíl je, tím více se z daného stavu může síť naučit [10].

$$p = (|r| + \tau)^{\alpha} \tag{2}$$

Rovnice 2 je kompletní vzorec pro výpočet priority stavu p, r je rozdíl odhadů neuronových sítí, τ je malá pozitivní konstanta, která zajišťuje, že žádný stav nebude mít prioritu 0 a α je konstanta v rozsahu od 0 do 1, která určuje, jakou měrou se bude vybírání vzpomínek pro trénování řídit jejich prioritou. V případě, že se α rovná 0, je priorita všech vzpomínek stejná.

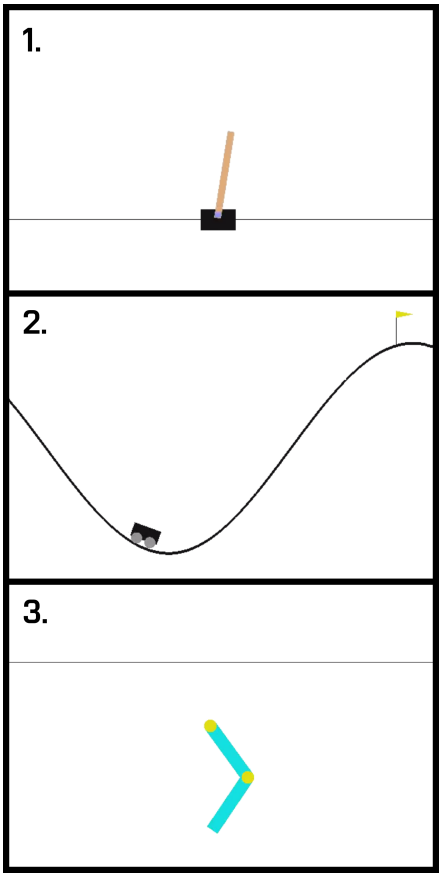
Priorita se ukládá do paměti společně se vzpomínkami. Časová složitost operací s pamětí s prioritními vzpomínkami je $O(log\ n)$, kde n je počet prvků.

3. Prostředí

K testování algoritmu a jeho vylepšení jsou použita prostředí z toolkitu Open AI Gym. Každé prostředí má daný limit počtu kroků, po jejichž provedení nastane koncový stav a prostředí musí být zrestartováno. Prostředí na vstupu očekává číslo akce, která má být provedena a na výstup dává vektor čísel popisující stav prostředí, viz tabulka 1.

3.1 CartPole-v0 a CartPole-v1

V prostředích CartPole-v0 a CartPole-v1 má agent podobu vozíku s tyčí, kterou má za úkol balancovat, pokud se tyč nakloní o více než 15 stupňů, hra končí. Tato prostředí jsou díky rozdílům v odměnách relativně snadno řešitelná. To je dáno tím, že od začátku dostávají kladnou odměnu a nemusí hledat jiný stav. Poté v okamžiku ztracení rovnováhy dostanou zápornou odměnou ihned informaci o stavu, který k tomu vedl. Tato prostředí jsou vhodná k testování efektivity učení agenta.



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

3.2 MountainCar-v0

Prostředí MountainCar-v0 se skládá z údolí a z vozíku, který je na jeho dně. Úkolem agenta je vyjet na pravý vrchol údolí. Problém spočívá v tom, že nemá dostatečnou hybnost, aby na vrchol vyjel pouze pomocí pohybu doprava. Musí se nejprve rozjet k levému vrcholu a až poté k pravému, pomocí tohoto manévru získá dostatečnou hybnost pro dosažení vrcholu pravého kopce. Prostředí MountainCar-v0 je přesným opakem prostředí CartPole-v0 a CartPole-v1. Agent náhodně prozkoumává svahy kopce, ale nedostává jinou odměnu než zápornou a tedy žádnou informaci o prostředí. Až po dosažení cíle na vrcholu pravého kopce dostane první informaci o prostředí, kterou může využít pro učení. Než ovšem cíle náhodnými akcemi dosáhne, chvíli to trvá a učení probíhá delší dobu. V některých případech dokonce vrchol vůbec objevit nemusí a učení končí neúspěšně. Toto prostředí je vhodné k testování prohledávání stavového prostoru.

3.3 Acrobot-v1

V prostředí Acrobot-v1 má agent podobu ramena se dvěma klouby, při čemž jeden je pevně ukotvený

Tabulka 1. Vlastnosti prostředí [11]

Název prostředí	Vlastnosti agenta	Možné akce	Maximální počet kroků	Řešení prostředí
CartPole-v0	Pozice vozíku Rychlost vozíku Úhel tyče Rychlost tyče na špičce	Pohyb doleva Pohyb doprava	200	Dosažení průměrného skóre alespoň 195 ze 100 epizod
CartPole-v1	Pozice vozíku Rychlost vozíku Úhel tyče Rychlost tyče na špičce	Pohyb doleva Pohyb doprava	500	Dosažení průměrného skóre alespoň 475 ze 100 epizod
MountainCar-v0	Pozice vozíku Rychlost vozíku	Pohyb doleva Žádný pohyb Pohyb doprava	200	Dosažení průměrného skóre alespoň -110 ze 100 epizod
Acrobot-v1	sic() pevného kloubu cos() pevného kloubu sic() volného kloubu cos() volného kloubu	Pohyb doleva Pohyb doprava	500	Nemá stanovené řešení, bere se průměrné skóre ze 100 epizod, po 100 epizodách trénování

a druhý je volný. Na začátku je rameno ve svislé poloze směrem dolů. Agentovým úkolem je pohybovat ramenem tak, aby dosáhlo určité výšky, ta je znázorněna čárou nad ramenem. Za splnění úlohy dostane agent odměnu 0, jinak dostává zápornou odměnu. Toto prostředí je vhodné na testování stability učení agenta. Často totiž dochází k jevu, že se na tomto prostředí během 100 epizod zvládne neuronová síť natrénovat a destabilizovat.

4. Experimenty

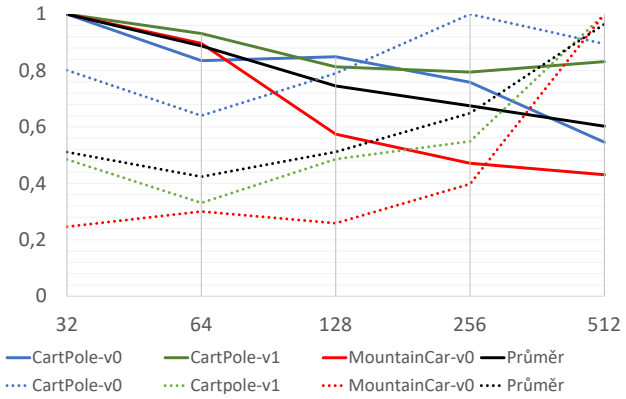
Algoritmus a jeho vylepšení jsem implementoval v jazyce Python 3.5. K implementaci neuronových sítí byla využita knihovna Keras. Odkaz na repozitář s kódem se nachází v úvodu.

Pokud není explicitně zmíněno jinak, je každá hodnota průměrem z 25 totožných testů. Pro všechna prostředí kromě Acrobot-v1 reprezentují hodnoty počet epizod, za který byl agent schopen prostředí vyřešit. V případě Acrobot-v1 se jedná o průměrné skóre ze 100 epizod po 100 epizodách trénování.

První dva experimenty se věnují nastavení hyperparametrů vanilla verze DQN, aby bylo v dalších experimentech zabývajících se jednotlivými vylepšeními algoritmu dosaženo co nejlepších výsledků.

4.1 Velikost trénovacího vzorku

Důležitou roli při učení agenta hraje velikost trénovacího vzorku. Jedná se o počet vzpomínek, ze kterých se bude agent v každém kroku učit. Obecně platí, že čím je trénovací vzorek větší, tím více se agent učí. Nastavení velké hodnoty trénovacího vzorku ovšem nemusí být ideální krok za efektivnějším učením agenta. Pravděpodobně totiž povede k trénování, které bude trvat méně epizod, ale jehož časová náročnost bude větší než při použití menší velikosti trénovacího vzorku. Následující experiment hledá hranici, od které už není výhodné velikost vzorku zvěšovat. Autoři originální práce použili při trénování agenta v prostředích Atari her velikost trénovacího vzorku 32 [1]. Následující testy byly provedeny na vanilla verzi DQN.



Obrázek 2. V grafu se nachází porovnání hodnot efektivity učení DQN s různými velikostmi trénovacího vzorku pro 3 prostředí a jejich průměr. Hodnoty v grafu jsou normalizované kvůli porovnání časové a výsledkové efektivity. Plné čáry znázorňují průměrný počet epizod, za který bylo prostředí vyřešeno a tečkované čáry průměrnou dobu trénování. Obecně platí, že čím je hodnota nižší, tím bylo řešení rychlejší/úspěšnější.

Na obrázku 2 vidíme výsledky experimentu. Pro jejich analýzu je důležité stanovit, zdali se bere jako nejúspěšnější výsledek experimentu, ten s nejkratší dobou trénování nebo ten s nejmenším počtem epizod. Průměr doby trvání učení je nejkratší pro velikost vzorku 64, tato varianta je časově efektivnější oproti

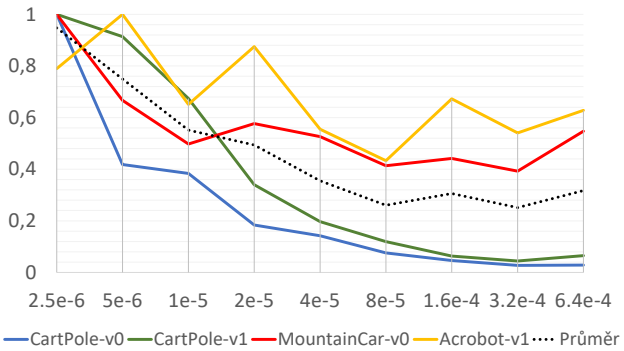
původní velikosti trénovacího vzorku přibližně o 17,5 % a v porovnání počtu epizod o přibližně 11 %. Počet epizod, který je potřeba k úspěšnému řešení prostředí, se s narůstající velikostí trénovacího vzorku snižuje, takže nejlépe vychází velikost vzorku 512, jeho časová náročnost je ovšem největší ze všech.

Jako kvalitní velikosti vzorku bych označil hodnoty 64, 128 a 256. Hodnota 128 má přibližně stejnou časovou náročnost jako původní verze trénovacího vzorku, ale do počtu epizod je efetivnější přibližně o 25,5 %. Hodnota 256 je poté časově náročnější o 27 %, ale efektivnější v počtu epizod o 32,5 %. V prostředích, které agentovy poskytují informace ve formě obrazu místo vektroru čísel, je ovšem kladen důraz na časovou efektivitu, protože i její malé zhoršení může znamenat o několik hodin delší dobu trénování. V takovém případě by byla nejlepší variantou velikost trénovacího vzorku 64. Pro další experimenty bude použita hodnota 256, protože mírně zvýšená doba učení u těchto prostředí nepředstavuje větší problém.

4.2 Vliv náhodných akcí na učení

Hyperparametr ε značí pravděpodobnost provedení náhodné akce místo té, kterou by provedl agent. Hodnota ε je na začátku trénování 1, tedy všechny prováděné akce jsou náhodné. Postupně se hodnota ε snižuje a agent začíná provádět akce podle předpovědí neuronové sítě. Otázkou zůstává, jak rychle by měl agent ε snižovat. V případě, že bude ε snižovat příliš pomalu, trénování bude trvat dlouho. Na druhou stranu, když ho bude snižovat moc rychle, agent nikdy nemusí nalézt cíl (prostředí MountainCar-v0), protože s nízkým ε se bude strategie řídit předpověďmi nenaučené neuronové sítě, která bude provádět stále stejné kroky, které nepovedou k cíli. Tento experiment zkoumá, jaká je ideální velikost konstanty, o kterou se v každém kroku trénování bude ε snižovat. Následující testy byly provedeny na vanilla verzi DQN.

V grafu na obrázku 3 lze pozorovat klesající trend počtu epizod pro vyřešení prostředí s rostoucí hodnotou konstanty pro snižování ε. U Acrobot-v1 je tento trend vidět až po proložení výsledků jednotlivých testů křivkou. To je dáno nestabilitou učení agenta. Z experimentu vyplývá, že obecně je pro tato prostředí nejlepší hodnota snižování ε 3.2e-4. U nejbližší větší zkoumané hodnoty byl ve všech prostředí zaznamenán nárůst počtu epizod potřebných pro vyřešení. Zároveň je tato hodnota nejefektivnější pro všechna prostředí kromě Acrobot-v1.

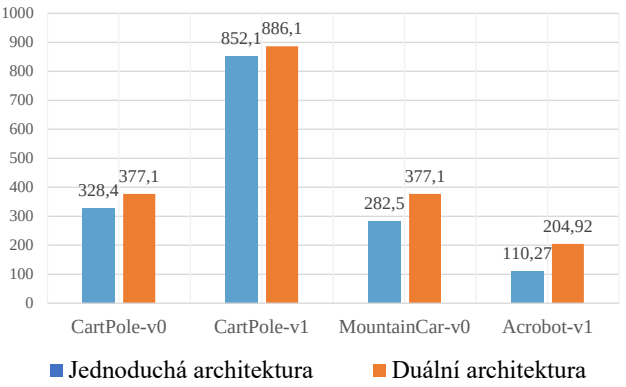


Obrázek 3. V grafu se nachází porovnání hodnot efektivity učení DQN s různými hodnotami konstanty, o kterou se snižuje ε pro všechna prostředí a jejich průměr. Hodnoty v grafu jsou normalizované kvůli vzájemnému porovnání hodnot z různých prostředí. Obecně platí, že čím je hodnota nižší, tím bylo řešení rychlejší/úspěšnější.

4.3 Vylepšení algoritmu DQN

Dále byly provedeny testy pro všechny kombinace výše zmíněných vylepšení algoritmu ve všech prostředích. Hodnoty v grafech jsou průměrné počty epizod, kterých bylo třeba pro vyřešení prostředí. Výjimkou je prostředí Acrobot-v1, tam se jedná o absolutní hodnotu průměrného skóre ze 100 epizod po 100 epizodách trénování. Obecně platí, že čím je hodnota nižší, tím bylo řešení rychlejší/úspěšnější. V následujících řádcích rozeberu zajímavé poznatky z těchto experimentů.

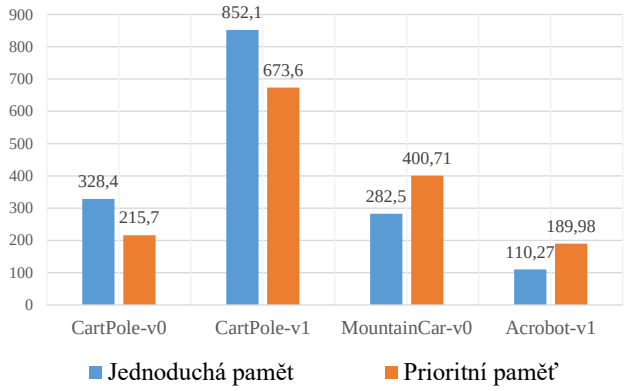
Prvním zajímavým poznatkem je bezesporu to, že ačkoliv se sama o sobě duální architektura na všech prostředích ukázala jako méně účinná než vanilla verze neuronové sítě (obrázek 4), tak v kombinaci s jinými vylepšeními obecně podporovala lepší výsledky (obrázek 7).



Obrázek 4. Porovnání efektivity učení DQN s jednoduchou a duální architekturou neuronové sítě

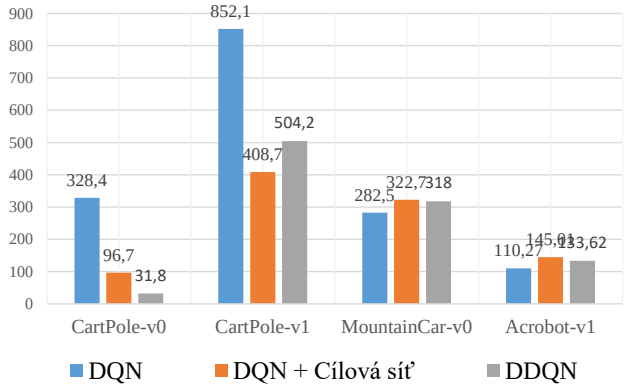
Podobný efekt bylo možné pozorovat u experimentů s využitím prioritní paměti (obrázek 5). Ta byla sice úspěšnější v prostředí CartPole-v0 a CartPole-v1, ale propadla u prostředí MountainCar-v0 a Acrobot-

v1, tedy u prostředí, kde je třeba více prohledávat stavový prostor. V kombinaci s jinými vylepšeními ovšem prioritní pamět rovněž obecně podporovala lepší výsledky (obrázek 7), stejně jako duální architektura neuronové sítě. Výsledky experimentů mohly být ovlivněny nevhodným nastavením konstanty α při určování priority vzpomínek, tuto konstantu jsem totiž rozsáhleji netestoval.



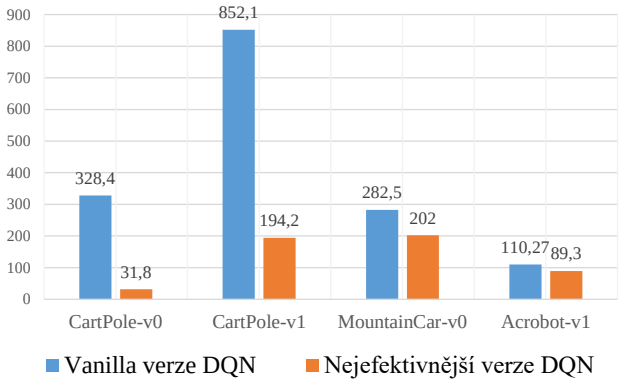
Obrázek 5. Porovnání efektivity učení DQN s využitím jednoduché a prioritní paměti

Pokud se podíváme pouze na vylepšení, týkající se přímo algoritmu (obrázek 6), zjistíme, že vylepšené verze DQN byly úspěšnější v prostředích CartPole-v0 a CartPole-v1, tedy v prostředích, kde není zapotřebí více prohledávat stavový prostor. V těchto prostředích byly úspěšnější v řádu stovek procent, v ostatních prostředích byly všechny verze relativně vyrovnané. To mohlo být způsobeno skutečností, že doba počátečního náhodného prohledávání stavového prostoru u složitějších prostředí zůstává relativně stejná se změnou algoritmu.



Obrázek 6. Porovnání efektivity učení DQN, DQN, která využívá cílovou síť a DDQN

Na obrázku 7 se nachází graf s porovnáním výsledků vanilla verze DQN s variantami, které se během experimentování ukázaly pro každé prostředí jako nejefektivnější. V případě CartPole-v0 se jednalo o DDQN bez prioritní paměti se základní architekturou neuronové sítě, tato varianta byla efektivnější cca o 90 %. V prostředí CartPole-v1 byla nejúspěšnější variantou algoritmu verze DDQN s prioritní pamětí a duální architekturou neuronové sítě, tato varianta byla efektivnější cca o 52 %. V prostředí MountainCar-v0 byla nejúspěšnější variantou algoritmu verze DQN s prioritní pamětí a duální architekturou neuronové sítě, tato varianta byla efektivnější cca o 28 %. V prostředí Acrobot-v1 byla nejúspěšnější variantou algoritmu verze DDQN bez prioritní paměti s duální architekturou neuronové sítě, tato varianta byla efektivnější cca o 19 %.



Obrázek 7. Porovnání efektivity učení vanilla verze DQN, s nejúspěšnější vylepšenou variantou pro každé prostředí

5. Závěr

Cílem práce bylo experimentovat s algoritmem DQN a jeho vylepšeními. Provedením mnoha testů jsem zkoumal závislost efektivity učení na dvou hyperparametrech - rozkladu ε a velikosti vzorku. Experimentálně jsem jako nejvhodnější velikost vzorku vybral hodnotu 256 a jako nejvhodnější konstantu rozkladu ε hodnotu 3,2e-4.

Dále jsem zkoumal jednotlivá vylepšení DQN a jejich vliv na efektivitu učení. Pro každé prostředí vyšla nejefektivnější mírně odlišná varianta vylepšeného algoritmu DQN. Procentuální nárůst efektivity dosáhl až 90 %. Důležitým faktem je, že vylepšení samy o sobě nedosahovaly znatelně lepších výsledků. Těch začaly dosahovat až při použití více vylepšení najednou.

Problematice algoritmu DQN se plánuji věnovat i nadále, zejména experimentováním s prostředími Atari her. Na těch chci ověřit správnost výsledků, které mi vyšly během experimentování s jednoduchými prostředími.

Poděkování

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

Dále bych chtěl bych poděkovat Ing. Michalu Hradišovi, Ph.D. za pomoc s tímto článkem.

Literatura

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, December 2013. https://arxiv.org/pdf/1312.5602.pdf.

[2] Wikipedia contributors. Artificial intelligence, April 2018. https://en.wikipedia.org/wiki/Artificial_intelligence.

[3] Wikipedia contributors. Artificial general intelligence, March 2018. https://en.wikipedia.org/wiki/Artificial_general_intelligence.

[4] Wikipedia contributors. Q-learning, April 2018. https://en.wikipedia.org/wiki/Q-learning.

[5] Henry George Bottomley. How many tic-tac-toe (noughts and crosses) games are possible?, 2002. http://www.se16.info/hgb/tictactoe.htm.

[6] Arthur Juliani. Simple reinforcement learning with tensorflow part 4: Deep q-networks and beyond. blogpost (english), September 2016. https://goo.gl/LWvMWe.

[7] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, December 2015. https://arxiv.org/pdf/1509.06461.pdf.

[8] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning, April 2016. https://arxiv.org/pdf/1511.06581.pdf.

[9] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, February 2016. https://arxiv.org/pdf/1511.05952.pdf.

[10] Jaromír Janisch. Let's make a dqn: Double learning and prioritized experience replay. blogpost (english), November 2016. https://goo.gl/5WPedM.

[11] Github contributors. Table of environments, June 2016. https://github.com/openai/gym/wiki/Table-of-environments.