Bakalářská práce

České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra kybernetiky

Porovnání standardních metod posilovaného učení

Michaela Cihlářová

Školitel: doc. Ing. Karel Zimmermann, Ph.D.
Květen 2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Cihlářová** Jméno: **Michaela** Osobní číslo: **483482**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra kybernetiky**

Studijní program: **Kybernetika a robotika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Porovnání standartních metod posilovaného učení

Název bakalářské práce anglicky:

Comparison of State-of-the-Art Baselines for Reinforcement Learning

Pokyny pro vypracování:

1. Nastudujte state-of-the-art metody posilovaného učení [1-3] a náhodného prohledávání [4] a ozkoušejte existující stable-baseline implementace na několika vybraných prostředích Open AI gym: https://gym.openai.com/
2. Navrhněte vlastní svět v simulátoru PyBullet https://pybullet.org, který bude v budoucnu použitelný jako domácí úloha na učení konvolučních neuronových sítí (tj. obsahuje kamerové či lidarové obrázky a je dostatečně jednoduchý aby šel naučit na běžném počítači).
3. Vyberte vhodnou metodu ze stable baselines a zkuste ji reimplementujte v prostředí PyTorch https://pytorch.org/
4. Porovnejte vaší implementaci s existujícími metodami.

Seznam doporučené literatury:

[1] DDPG https://arxiv.org/pdf/1509.02971.pdf
[2] A2C https://arxiv.org/abs/1602.01783
[3] https://arxiv.org/abs/1502.05477
[4] CMA-ES https://arxiv.org/abs/1604.07269

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Karel Zimmermann, Ph.D., vidění pro roboty a autonomní systémy FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **08.01.2021** Termín odevzdání bakalářské práce: **21.05.2021**

Platnost zadání bakalářské práce: **30.09.2022**

doc. Ing. Karel Zimmermann, Ph.D. podpis vedoucí(ho) práce	prof. Ing. Tomáš Svoboda, Ph.D. podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Studentka bere na vědomí, že je povinna vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.	
Datum převzetí zadání	Podpis studentky

Poděkování

Mé poděkování patří doc. Ing. Karlu Zimmermannovi, Ph.D. za odborné vedení, trpělivost a ochotu, kterou mi v průběhu zpracování bakalářské práce věnoval.

Prohlášení

Prohlašuj, že jsem předloženou bakalářskou práci vypracovala samostatně a s použitím uvedené literatury a pramenů.

V Praze, 21. května 2021

Abstrakt

V této bakalářské práci se zaměřujeme na porovnání vybraných metod posilovaného učení a algoritmů z oblasti evolučních metod pro spojitý akční prostor. První část práce je věnována teoretickému základu zadaných algoritmů. V druhé části popíšeme tvorbu vlastního prostředí a vývoj funkce odměny. Dále porovnáme metody na jak již existujících prostředích, tak na námi vytvořeném, a nakonec se detailněji podíváme na implementaci Advantage Actor-Critic algoritmu.

Klíčová slova: posilované učení, strojové učení, simulace a modelovani, evolucni metody

Školitel: doc. Ing. Karel Zimmermann, Ph.D.
Resslova 307/9,
120 00 Praha 2

Abstract

This bachelor thesis focuses on comparing the selected reinforcement learning methods and the evolution methods algorithms for continuous action space. The first part of this thesis is dedicated to the theoretical background of the assigned algorithms. In the second part, we describe the creation of the environment and the evolution of the reward function. Next, we compare the methods in the existing environment and the environment made by us. Finally, we take a closer look at the implementation of the Advantage Actor-Critic algorithm.

Keywords: reinforcement learning, machine learning, simulation and modeling, evolution methods

Title translation: Comparison of State-of-the-Art baselines for Reinforcement Learning

Obsah

Obrázky

Tabulky

Kapitola 1

Úvod

Umělá inteligence je vědní obor, který se zabývá vývojem algoritmů a strojů vykazujících znaky inteligentního chování. Nejedná se však o přesnou definici, protože vysvětlení tohoto pojmu může být několik a náhled na ně není vždy stejný. Tvorba umělé inteligence je velice komplexní záležitost, která si vyžaduje velké znalosti z mnoha různých vědních oborů. Zejména informatice, matematice a logice.

Historie umělé inteligence sahá až do poloviny minulého století. Hlouběji se umělou inteligencí začala zabývat kybernetika. Je to nauka o principech řízení a přenosu informací ve strojích. Výpočetní technika však v minulém století neměla dostatečné podklady pro rozvoj tohoto směru. V současnosti však vývoj postoupil dopředu a zdá se, že budoucnost umělé inteligence je možné dále rozvíjet. V průběhu let vzniklo několik přístupů řešících umělou inteligenci jako celek. Mezi nejznámější se řadí expertní systémy a neuronové sítě. Tento obor má v dnešní době široké uplatnění v mnoha oblastech techniky, např. při vývoji humanoidních robotů. Budoucnost umělé inteligence je v této chvíli v podstatě neodhadnutelná.

V této bakalářské práci se zaměříme na podoblast umělé inteligence zvané posilované, neboli zpětnovazební, učení a uvedeme také krátký náhled do evolučních metod. Posilované učení je učení pomocí zpětné vazby a získané zkušenosti by se měly promítnout do budoucího chování. Jedná se o jedno ze základních východisek umělé inteligence. Mimo algoritmy uvedené v této práci existují mnohé další, např. Monte Carlo [1], Asynchronous Advantage Actor-Critic (A3C) [2], Deep Q Network (DQN) [3] atd., které jsou aplikovatelné jak na diskrétní, tak na spojitý prostor. Výhodou těchto metod je schopnost učení bez učitele, tedy nevyžadují testovací data ani dohled. Cenou toho, že pro fungování algoritmu nemusíme mít uložená testovací data, je ovšem větší výpočetní náročnost.

V kapitole 2 se budeme věnovat teoretickému základu tří vybraných algoritmů posilovaného učení Trust Region Policy Optimization [4], Advantage Actor-Critic [5] a Deep Deterministic Policy Gradient [6]. Pro porovnání těchto metod s odlišným odvětvím umělé inteligence využijeme algoritmus z oblasti evolučních metod Covariance Matrix Adaptation Evolution Strategy [7], který popíšeme v kapitole 3.

V rámci této práce se soustředíme na vytvoření vlastního 3D prostředí

Headcrab-v0 inspirovaného počítačovou hrou Halflife a určeného k porovnání výše uvedených metod. V kapitole 4 popíšeme postup, jakým jsme vytvářeli modely a navrhovali funkci odměny.

Nakonec v kapitole 5 porovnáme vybrané algoritmy jak na již existujících prostředích, tak na námi vytvořeném prostředí Headcrab-v0. A zaměříme se také na vlastní implementaci Advantage Actor-Critic metody.

Motivací k této práci byla myšlenka vytvořit domácí úlohu pro studenty předmětu Vidění robotů, která by pro ně, díky spojení s populární počítačovou hrou, byla zajímavá a zároveň dostatečně jednoduchá pro zpracování i na běžném počítači.

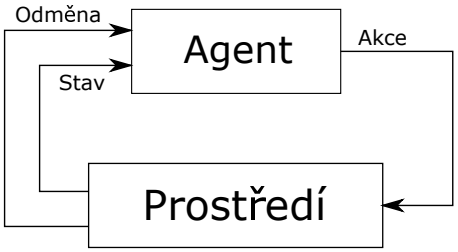
Kapitola 2

Metody posilovaného učení

Posilované učení, nebo také zpětnovazební učení (*Reinforcement learning*, RL) se řadí mezi jednu z metod strojového učení (*Machine learning*, ML), tedy podoblast umělé inteligence (*Artificial intelligence*, AI). Hlavní myšlenka je existence agenta, který na základě zpětné vazby hledá nejlepší řešení pro daný problém. Jedná se o případ bez učitele, agent nemá dohled, který by hodnotil správnost akce, učí se přímo ze svého prostředí pomocí odměn. Výhodou metody je, že nejsou potřeba testovací data a agent je schopen pracovat v naprosto neznámém prostředí. Ovšem tento způsob je často časové i výpočetně náročný.

2.1 Markovův rozhodovací proces

Markovovy rozhodovací procesy poskytují matematický rámec pro modelování problémů posilovaného učení. Zjednodušeně můžeme tento rámec definovat následovně. Agent se snaží generovat akce, díky kterým získá největší odměnu. Jeho vstup v čase t tvoří, často neúplná, informace o stavu prostředí $s_t \in \mathbb{R}^m$. Počáteční stav značíme s_0. Z těchto dat určí přípustnou akci $a_t \in \mathbb{R}^n$, čímž se dostane do stavu s_{t+1}. Výsledky akcí nejsou obecně deterministické, jelikož reakce může záviset na parametrech, ke kterým nemá agent přístup. Také samo prostředí může mít stochastický charakter. Za stav s_t získá agent odměnu[1] $r(s_t) \in \mathbb{R}$, která mu pomůže v nalezení optimálních hodnot a_t. Odměna může být i záporná, v tom případě ji lze nazvat penalizací, a agent získá informaci, že stav s je nežádoucí.



Obrázek 2.1: Princip posilovaného učení

[1] anglicky reward

Cílem je najít strategii $\pi(a_t|s_t)$, která maximalizuje kumulativní odměnu, tedy

$$\pi^* = \arg\max_{\pi} J_\pi \tag{2.1}$$

$$J_\pi = \mathbb{E}_{\tau\sim\pi}\left[\sum_{r_t\sim\tau} \gamma^t r(s_t)\right] \tag{2.2}$$

kde $0 \leq \gamma < 1$ je srážkový faktor. Pokud se hodnota γ blíží k nule, je pro agenta důležitější okamžitá odměna, naopak pokud se blíží k 1, soustředí se na budoucí zisk. Faktor nemůže být roven jedné, jelikož by neumožnil algoritmu konvergovat [8].

Metody posilovaného učení můžeme rozdělit na dvě základní kategorie. Algoritmy spadající do první kategorie, tzv. model-based, nejdříve získávají všechna data o prostředí, ve kterém se agent nachází a na základě získaného modelu vytvoří strategii. Jsou to například metody dynamického programování. Pro jejich fungování je však potřeba mít plně pozorovatelné prostředí, což v našem případě není možné. Proto se v této práci soustředíme na druhou kategorii, tvz. model-free metody.

2.2 Model-free metody

Tyto algoritmy nepotřebují pro nalezení optimální akce znát všechny informace o prostředí. Jsou tedy méně náročné na paměť, jelikož neukládají všechny dosažitelné stavy a akce. Existují dva způsoby, jak k učení model-free algoritmů přistupovat. V prvním případě se síť učí ohodnotit dosažený stav nebo kombinaci provedené akce a následujícího stavu. Nazveme ji value-based. V druhém se učí přímo strategii, kterou má agent vykonat - policy-based.

2.2.1 Value-based metody

Jak již bylo řečeno, value-based metody spočívají v existenci ohodnocující funkce, která určí, jak dobrý je daný stav. Cílem je nalézt strategii, která maximalizuje očekávanou odměnu.

V-hodnota

Přístup, při kterém zkoumáme pouze dosažený stav a nezajímá nás, jakou akcí se tam agent dostal, se nazývá V-funkce, V-hodnota. Tuto funkci vyjádříme následovně

$$V^\pi(s) = \mathbb{E}\left[\sum_{k=0}^{T} \gamma^k r_{t+k} | s_t = s, \pi\right] \tag{2.3}$$

kde γ je srážkový faktor a π je strategie. Optimální V-funkce je definována jako

$$V^*(s) = \max_{\pi} V^{\pi}(s) \tag{2.4}$$

Q-hodnota

Dalším přístupem k value-based metodám není pouze ohodnocení daného stavu, ale dvojice akce-stav. Tuto funkci nazveme Q-funkce nebo také Q-hodnota. Její výpočet je podobný jako pro V-funkci

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{k=0}^{T} \gamma^k r_{t+k} | s_t = s, a_t = a, \pi\right]. \tag{2.5}$$

Optimální hodnota je obdobně definována jako

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a). \tag{2.6}$$

Vzhledem k tomu, že bereme v úvahu i akce, kterými jsme do daného stavu došli, můžeme z Q-hodnoty přímo získat strategii pro agenta.

$$\pi^* = \arg\max_{a} Q^*(s,a) \tag{2.7}$$

2.2.2 Policy-based metody

Tento typ metod dokáže přímo získat optimální strategii bez nutnosti vytvoření V-funkce. Strategie, značená $\pi(a|s)$, je nejčastěji reprezentovaná jako pravděpodobnostní rozdělení nad akcemi, které agent může v daném stavu vykonat. Jejich výhodou je schopnost naučit se stochastickou strategii, která je v mnoha případech lepší, než deterministická [5]. Avšak mají tendenci konvergovat do lokálního optima namísto globálního [5]. Abychom mohli určit, která akce je pro momentální stav nejlepší parametrizujeme strategii parametrem θ na základě kterého aktualizujeme síť.

V následujících sekcích se budeme nejdříve věnovat optimalizaci policy-based metod pomocí gradientu, tzv. policy grandient, a poté se zaměříme na rozšíření informací o value-based algoritmech.

2.3 Policy gradient

I přes to, že se policy gradient metody řadí mezi policy-based, využíváme zde ohodnocující funkci $J(\theta)$ jako optimalizační funkci. Nejedná se však o V-funkci ani Q-funkci, jelikož hodnota $J(\theta)$ je určena z parametrizované strategie $\pi(\theta)$ a ne přímo parametrizována θ. Tedy

$$J(\theta) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t); \pi_\theta\right] \tag{2.8}$$

Parametry policy gradient metod jsou aktualizovány pomocí gradientní metody

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta_t) \tag{2.9}$$

kde α je velikost kroku. Derivace funkce $J(\theta)$ je definována v [9], zde uvedeme pouze výsledný vzorec.

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) r(\tau_\theta)\right] \tag{2.10}$$

kde τ_θ je trajektorie vygenerována strategií π_θ.

Všechny algoritmy popsané v této kapitole využívají k aktualizaci strategie tuto metodu.

2.3.1 Trust Region Policy Optimization (TRPO)

Postup ve směru gradientního růstu může být velmi nebezpečný. Máme-li nastavený pevný krok α je možné, že bude pro danou iteraci příliš velký a agent spadne do oblastí, ze které už není schopen nalézt optimum funkce. Tímto problémem se zabývá algoritmus Trust Region Policy Optimization (TRPO). Pomocí Kullbackovy–Leiblerovy divergence [10], míry používané v matematické statistice k určení, jak se od sebe odlišuji dvě distribuční funkce, určíme maximální velikost kroku pro danou iteraci, tak abychom dodrželi míru odlišnosti nové a staré strategie δ. Aktualizaci parametru θ provedeme pomocí rovnice

$$\begin{aligned} \theta_{k+1} = &\arg\max_\theta \mathcal{L}(\theta_k, \theta) \\ &\text{Za podmínky } \overline{D}_{KL}(\theta, \theta_k) \leq \delta \end{aligned} \tag{2.11}$$

kde $\mathcal{L}(\theta_k, \theta)$ je kompenzační zvýhodnění a $\overline{D}_{KL}(\theta, \theta_k)$ je průměrná Kullbackova–Leiblerova divergence.

$$\mathcal{L}(\theta_k, \theta) = \mathbb{E}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A_{\pi_{\theta_k}}(s, a)\right] \tag{2.12}$$

$$\overline{D}_{KL}(\theta, \theta_k) = \mathbb{E}\left[D_{KL}(\pi_\theta(\cdot|s)||\pi_{\theta_k}(\cdot|s))\right] \tag{2.13}$$

kde $A_\pi(s, a)$ nazveme zvýhodnění a je definované jako

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s) \tag{2.14}$$

Výše uvedené rovnice představují teoretický základ TRPO. Je však velmi obtížně s nimi pracovat a proto se je snažíme aproximovat. Postup aproximace lze nalézt v [4]. V tomto dokumentu uvedeme pouze výsledné vzorce.

V případě, že označíme parametry strategie θ, parametry ohodnocující funkce ϕ, iteraci učení k a $\mathcal{D}_k$ množinu trajektorii získanou strategií $\pi(\theta_k)$, získáme rovnice

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{x_k^\top H_k x_k}} x_k \tag{2.15}$$

$$x_k \approx H_k^{-1} g_k \tag{2.16}$$

$$g_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} A_t \tag{2.17}$$

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} (V_\phi(s_t) - r_t)^2 \tag{2.18}$$

kde g je gradient strategie a H je Hessián průměrné Kullbackovy–Leiblerovy divergence.

2.4 Temporální diference TD(0)

Existuje několik metod temporální diference, např. TD(1), TD(λ), TD(0). Tématem této práce jsou ale pouze algoritmy založené na TD(0), ostatním se nebudeme věnovat. Kombinuje se zde myšlenka metody Monte Carlo [1] a již zmíněného dynamického programování [11]. Toto propojení nám umožňuje učení z přímé zkušenosti, bez nutnosti vytvoření modelu prostředí, což by metody dynamického programování vyžadovaly, a zároveň je možná aktualizace odhadů bez nutnosti čekání na konečný výsledek. Tedy V-funkci upravíme po každé vykonané akci následujícím způsobem

$$V(s_t) \leftarrow V(s_t) + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t)). \tag{2.19}$$

Stejně přistupujeme k úpravě Q-funkce

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \tag{2.20}$$

kde α je velikost kroku a γ srážkový faktor. Definujeme si nyní ještě TD-cíl a TD-chybu[2], které využijeme v níže popsaných algoritmech

$$\tau_t = r_t + \gamma V(s_{t+1}) \tag{2.21}$$

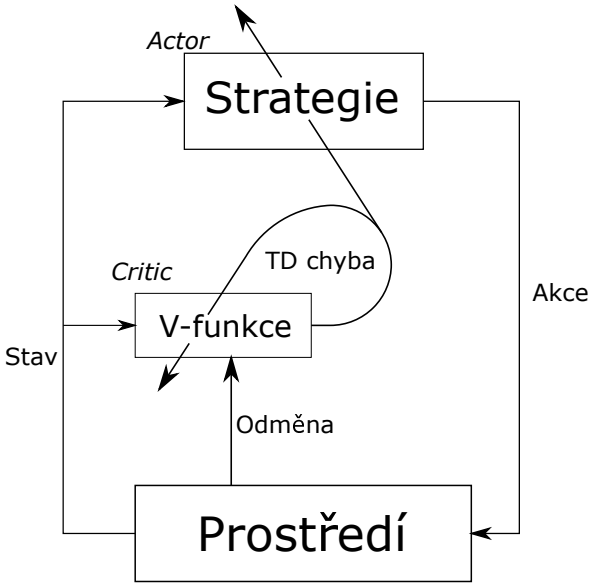
$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t). \tag{2.22}$$

2.5 Actor-Critic (AC)

Actor-Critic algoritmy vznikly jako propojení value-based a policy-based metod. Jelikož každý přístup má určité nevýhody, jejich spojením se je snažíme utlumit. Princip AC spočívá v existenci jak sítě, která generuje akce (Actor), tak sítě, která ohodnocuje daný stav (Critic). Tedy Actor dostane jako vstup informace o prostředí a výstupem jsou co nejlepší akce agenta.

[2] anglicky TD-target a TD-error

Zároveň také kontroluje jejich odměnu a na základě toho optimalizuje svou strategii (policy-based). Současně si Critic vytváří V-funkci, která hodnotí stav, ve kterém se prostředí právě nachází (value-based). Obě tyto sítě učíme zvlášť pomocí gradientního vzestupu. Jejich parametry se aktualizují po každém kroku, jedná se tedy o učení pomocí TD.



Obrázek 2.2: Princip Actor-Critic metod

2.5.1 Advantage Actor-Critic algoritmus (A2C)

Jeden z nepopulárnějších přístupů k AC metodám je Advantage Actor-Critic, neboli A2C [**?**]. Vychází z rozložení Q-hodnoty

$$Q(s_t, a_t) = V(s_t) + A(s_t, a_t) \tag{2.23}$$

kde funkci $A(s_t, a_t)$ nazveme zvýhodnění[3]. Tedy

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t) \tag{2.24}$$

Abychom nemuseli učit zvlášť dvě sítě, jednu pro V-hodnotu a jednu pro Q-hodnotu, můžeme si pomoci tím, že Q-hodnotu aproximujeme TD-cílem.

$$A(s_t, a_t) = r_t + \gamma V(s_{t+1}) - V(s_t). \tag{2.25}$$

Všimněme si, že se vlastně jedná o TD-chybu. Díky tomu stačí ohodnocovat pouze stavy a tím se value-based síť výrazně zjednoduší. Critic se tedy učí zvýhodňující funkci, což mu umožní zjistit nejen jak dobrá byla akce, ale také jak se může zlepšit. Hlavní výhodou tohoto přístupu je větší stabilita modelu [5]. Pro učení Critica využijeme princip MSE (*Mean squared error*), což je rozdíl mezi odhadovanou hodnotou a skutečnou hodnotou, to celé umocněné

[3] anglicky advantage

na druhou. Jelikož ale naše aproximované zvýhodnění už udává tento rozdíl, stačí zadat

$$L_{\theta^V} = A(s_t, a_t)^2 \tag{2.26}$$

Actor, v závislosti na ohodnocení kritika, generuje pravděpodobnostní rozdělení akcí. Díky tomu můžeme jeho účelovou funkci definovat jako

$$L_{\theta^\pi} = \log(P(a_t|s_t)) \cdot A(s_t, a_t) \tag{2.27}$$

2.5.2 Deep Deterministic Policy Gradient (DDPG)

Pokud bychom chtěli přistupovat k metodě AC z pohledu Q-učení, můžeme k tomu využít Deep Deterministic Policy Gradient (DDPG) algoritmus. Jedná se o propojení Deterministic Policy Gradient (DPG) [12] a Deep Q-network (DQN) [3] přizpůsobené pro spojitý akční prostor. Jeho myšlenka je existence ne dvou, ale čtyř neuronových sítí.

- θ^Q: Q síť
- θ^π: deterministická síť pro strategii
- $\overline{\theta^Q}$: cílová Q síť
- $\overline{\theta^\pi}$: cílová síť pro strategii

Princip prvních dvou sítí je velmi podobný jako u A2C s tím rozdílem, že Actor přímo generuje akce pro prostředí a ne jejich pravděpodobnostní rozdělení. Zbylé cílové sítě pomalu následují předchozí naučené sítě. Tím je učení výsledné strategie tolik nediverguje, je mnohem stabilnější.

Další odchylkou od A2C algoritmu je podmínka existence paměti. A2C je možné pomocí ní vylepšit, ale není to podmínkou pro správné fungování. U DDPG je. To nám umožní vracet se k již prozkoumaným stavům a zlepšovat ohodnocení a akce.

Na začátku iterace tedy nejdříve zaplníme paměť, poté z ní náhodně vybereme prvky $[s_t, a_t, r_t, s_{t+1}]$, podle kterých zaktualizujeme Q-síť a deterministickou síť pro strategii.

$$y = r_t + \gamma Q_{\overline{\theta^Q}}(s_{t+1}, \pi_{\overline{\theta^\pi}}(s_{t+1})) \tag{2.28}$$

$$L_{\theta^Q} = \arg\min_{\theta^Q} \sum_{s,a,y} \|Q_{\theta^Q(s_t,a_t)} - y\| \tag{2.29}$$

$$L_{\theta^\pi} = \arg\max_{\theta^Q} \sum_{s} Q_{\theta^Q}(s_t, \pi_{\theta^\pi}(s_t)) \tag{2.30}$$

Nyní můžeme upravit cílové sítě.

$$\overline{\theta^Q} = \alpha\theta^Q + (1-\alpha)\overline{\theta^Q} \tag{2.31}$$

$$\overline{\theta^\pi} = \alpha\theta^\pi + (1-\alpha)\overline{\theta^\pi} \tag{2.32}$$

kde $\alpha \ll 1$ je velikost kroku.

Kapitola 3

Evoluční metody

Evoluční strategie jsou založeny, jak je podle názvu zřejmé, na principu evoluce. Soustřeďují se hlavně na přirozený výběr. Tyto algoritmy probíhají ve smyčce a každá iterace se nazývá generace. V generaci se vybere několik jedinců, kteří mají lepší výsledky, větší šanci na přežití než ostatní a jejich vlastnosti předáme další generaci. Tím se postupem času dostaneme do optima. Pokud se na to podíváme z matematického hlediska, máme funkci $f(x)$, kterou chceme optimalizovat. Abychom nalezli vyhovující hodnoty x, vytvoříme pravděpodobnostní rozdělení $p_\theta(x)$ s parametry θ. Našim cílem je nalézt co nejlepší konfiguraci θ[7].

3.1 Covariance Matrix Adaptation Evolution Strategy (CMA-ES)

Nejjednodušší přístup k evolučním metodám je parametrizace normálního rozdělení.

$$\theta = (\mu, \sigma) \tag{3.1}$$

$$p_\theta(x) \sim \mathcal{N}(\mu, \sigma^2 I). \tag{3.2}$$

Pokud však musíme pracovat ve složitějším prostředí, toto základní rozdělení nemusí být dostačující. Proto využijeme mnohorozměrné normální rozdělení:

$$p_\theta(x) \sim \mathcal{N}(\mu, \sigma^2 C) \tag{3.3}$$

$$\theta = (\mu, \sigma, C) \tag{3.4}$$

kde C je kovarianční matice díky které bereme do úvahu i závislosti mezi jednotlivými proměnnými. Dále uvedeme postup pomocí kterého upravíme jednotlivé parametry CMA-ES. Je důležité nejdříve připomenout vlastnosti kovarianční matice:

- je čtvercová a symetrická
- je pozitivně semidefinitní

- všechny její vlastní čísla jsou reálná
- existuje její inverze

Při každé iteraci budeme aktualizovat pět parametrů. První tři jsou definované θ. Zbylé dva, p_σ a p_c, tzv. cesty vývoje, jsou pomocné proměnné, které určují změnu kroku σ a kovarianční matice C. Máme-li prohledat prostor dimenze n v iteračním kroku t, pak $\mu^t, p_\sigma, p_c \in \mathbb{R}^n$, $C^t \in \mathbb{R}^{n\times n}$ a $\sigma^t > 0$

Na začátku každé generace získáme set prvků x. Z nich vybereme λ prvků, které mají nejlepší výsledky a ty využijeme k aktualizaci parametrů θ. Nejdříve se zaměříme na průměrnou hodnotu μ.

$$\mu^{t+1} = \mu^t + \alpha_\mu \frac{1}{\lambda}\sum_{i=1}^{\lambda}(x_i^{t+1} - \mu^t) \tag{3.5}$$

kde $\alpha_\mu \leq 1$ je faktor učení, kterým určujeme rychlost změny μ. Další parametr je délka kroku σ. Abychom správně určili, o kolik krok zvětšit, využijeme pomocný parametr p_σ.

$$p_\sigma^{t+1} = (1-\alpha_\sigma)p_\sigma^t + \sqrt{\alpha_\sigma(2-\alpha_\sigma)\lambda}C^{t^{-\frac{1}{2}}}\frac{\mu^{t+1}-\mu^t}{\sigma^t}. \tag{3.6}$$

$$\sigma^{t+1} = \sigma^t \cdot \exp\left(\frac{\alpha_\sigma}{d_\sigma}\left(\frac{\|p_\sigma^{t+1}\|}{\mathbb{E}\|\mathcal{N}(0,I)\|} - 1\right)\right) \tag{3.7}$$

kde d_σ je parametr útlumu a α_σ je faktor učení. Očekávaná velikost náhodné proměnné z normálního rozdělení je definovaná jako

$$\mathbb{E}\|\mathcal{N}(0,I)\| = \sqrt{2}\frac{\Gamma\left(\frac{n+1}{2}\right)}{\Gamma\left(\frac{n}{2}\right)} \tag{3.8}$$

pro zjednodušení ji ale aproximujeme

$$\mathbb{E}\|\mathcal{N}(0,I)\| \approx \sqrt{n}(1 - \frac{1}{4n} + \frac{1}{21n^2}) \tag{3.9}$$

Při úpravě kovarianční matice využijeme p_c stejně tak jako u kroku σ.

$$p_c = (1-\alpha_{cp}) + \sqrt{\alpha_{cp}(2-\alpha_{cp})\lambda}\frac{\mu^{t+1}-\mu^t}{\sigma^t} \tag{3.10}$$

$$\begin{aligned} C^{t+1} = (1-\alpha_{c\lambda}-\alpha_{c1})C^t + \alpha_{c1}p_c^{t+1}p_c^{t+1^\top} + \\ +\alpha_{c\lambda}\frac{1}{\lambda}\sum_{i=1}^{\lambda}\left(\frac{x_i^{t+1}-\mu^t}{\sigma^t}\right)\left(\frac{x_i^{t+1}-\mu^t}{\sigma^t}\right)^\top \end{aligned} \tag{3.11}$$

kde $\alpha_{c\lambda}, \alpha_{c1}$ a α_{cp} jsou faktory učení. Obecně se používá $\alpha_{c\lambda} \approx \min(1, \frac{\lambda}{n^2})$.

Po aktualizaci kovarianční matice celý postup opakujeme dokud nedojdeme do optimálního řešení. V případě, že všechny vzorky x_i nemají stejnou váhu je možné výše uvedené vzorce jednoduše upravit [7]. (V této práci se tímto problémem nezabýváme).

Kapitola 4

Prostředí

Vzhledem k tomu, že je tato práce směřována jako domácí úloha pro studenty předmětu vidění robotů, rozhodli jsme ve ve vývoji prostředí inspirovat populární počítačovou hrou Halflife. Přesněji skokem parazita, nazývaným Headcrab, na člověka G-Man. V této kapitole se zaměříme na knihovnu Pybullet[13], pomocí které provádíme simulaci, vytvoření modelů a nakonec uvedeme implementaci prostředí v jazyce Python.

4.1 Pybullet

Pybullet je nadstavba fyzikálního enginu Bullet pro programovací jazyk Python. Poskytuje simulaci dopředné dynamiky a výpočet Pomocí ní lze načíst soubory v mnoha formátech, např. URDF (Unified Robot Description Format), SDF (Standard Database Format) a MJCF (Multi-Joint dynamics with Contact). Tuto knihovnu jsme zvolili díky její kompatibilitě s OpenAI Gym[14] a TensorFlow[15].

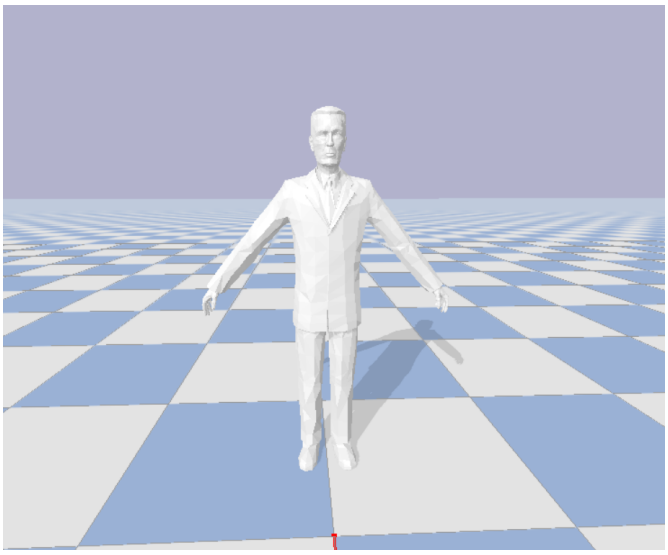
4.2 Tvorba modelů

Na popis modelů jsme zvolili formát URDF. K tomu využijeme 3D modelovací program Blender[16] a rozšiřující knihovnu Phobos[17]. Tato knihovna umožňuje export právě formátu URDF, dále SDF a SMURF (Supplementable, Mostly Universal Robot Format). I přes to, že existují novější verze programu Blender, k vytvoření modelů jsme využili verzi 2.79b, protože knihovna Phobos byla
v době sepsání této práce dostupná pouze pro tuto verzi.

G-Man

Začneme popisem modelu G-Man. Tento model je velmi jednoduchý, jelikož ho nemusíme ovládat je statický. Z internetu získáme model ve formátu OBJ [18], který importujeme do programu Blendr. Abychom mohli jednoduše detekovat, zda se Headcrab dotkl hlavy, rozdělíme celý model G-Man na dva objekty. Hlavu a zbytek těla. Pybullet nám umožní detekovat kolizi s jen

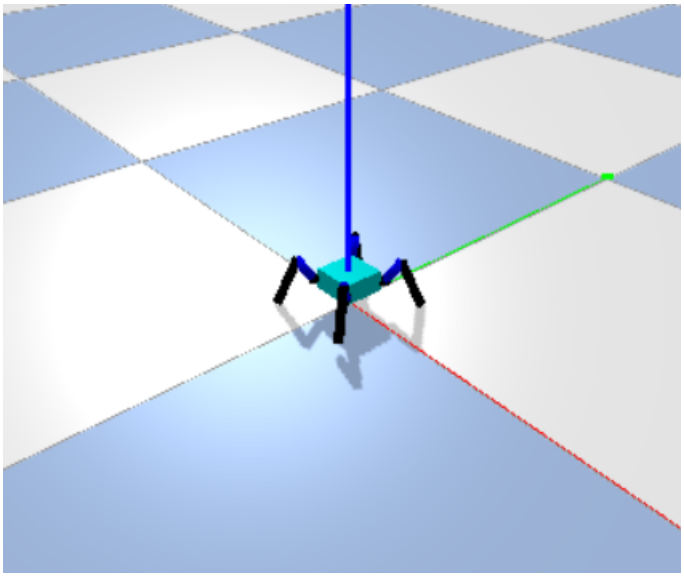
potřebnou částí. Poté nastavíme nulovou hmotnost celého G-man, což značí, že se jedná o tuhé těleso a nemůžeme změnit jeho polohu působením jakékoli velké síly.



Obrázek 4.1: Model postavy G-Man v prostředí Pybullet

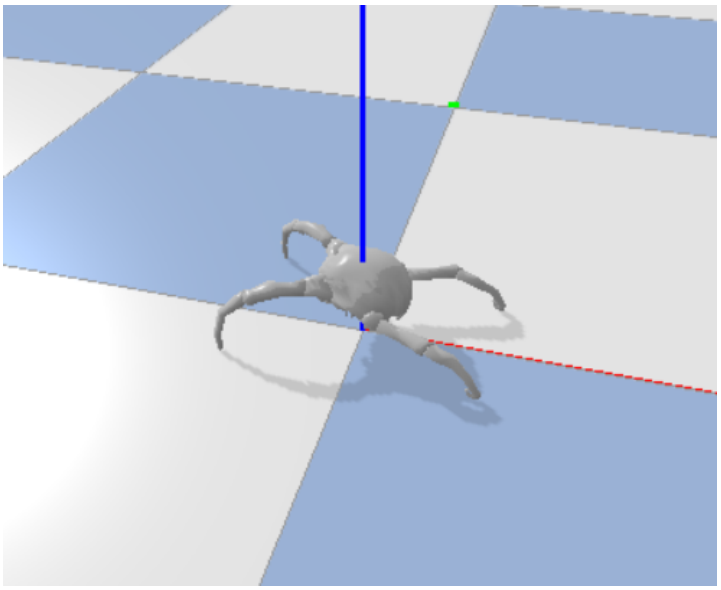
Headcrab

Model Headcrab je náročnější na vytvoření. Jako základ opět stáhneme volně dostupný model ve formátu OBJ [19]. To však využijeme pouze jako vizuální část. Fyzikální model se snažíme zjednodušit, aby simulace nebyla příliš náročná na výpočet. K tomu nám poslouží jednoduchá reprezentace čtyřnohého robota.



Obrázek 4.2: Fyzikální předloha pro model Headcraba

Aby pozice vizuální části Headcraba odpovídala fyzikální, rozdělíme model na tělo a končetiny. Každá noha má tři části, které budeme ovládat. Všech dvanáct kloubů definujeme, pomocí knihovny Phobos, jako rotační a jejich meze nastavíme na intervalu $[-\pi, \pi]$. Aby mohl Pybullet počítat síly a momenty hybnosti, je třeba určit hmotnost a tenzor setrvačnosti pro každou nohu a torzo Headcraba. Všechny části nohou jsme nastavili na stejnou hmotnost, tedy $0.1\,kg$ a pro torzo jsme zvolili $1.5\,kg$. Nakonec jsme pro všechny části zvolili tenzor setrvačnosti jako jednotkovou matici.



Obrázek 4.3: Model Headcraba v prostředí Pybullet

4.3 Implementace

Při tvorbě prostředí jsme se drželi formátu OpenAI Gym. Tím ho generalizujeme a můžeme poté velmi jednoduše pracovat se stable-baselines[20] implementacemi. Z toho vyplývá i existence několika funkcí, které uvedeme v následujících částech kódu. Na začátku každého učení je potřeba načíst prostředí a k tomu nám poslouží *gym.make*():

```
import gym
import headcrab_env

env = gym.make('headcrab-v0')
```

Dále pracujeme s funkcí *reset*, která vrátí prostředí do počátečního stavu, a funkcí *step*, která pro dané akce vyhodnotí nový stav, vypočte odměnu, zjistí zda jsme v konečném stavu a uvede komentář k momentálnímu stavu. Uvádíme zde cyklus pro jednu epizodu:

```
state = env.reset()
done = False

while not done:
    # get action from current state
    action = function(state)

    # step environment
    next_state, reward, done, info = env.step(action)
```

4.3.1 Akční a stavový prostor

Abychom mohli správně pracovat s prostředím, je důležité vědět, zda se jedná o diskrétní či spojitý akční prostor. V našem případě pracujeme se spojitým akčním prostorem. Akce, které přijímá prostředí, jsou reprezentovány dvanácti-dimenzionálním vektorem představujícím natočení rotačních kloubů Headcraba, které se následně škálují na přednastavený interval.

Jedním z výstupů prostředí je stavový vektor. I přes to, že agent neumí určit, co přesně jeho hodnoty přestavují, musí získat informace, které mu pomohou dosáhnout cíle. Jak jsme již uvedli v popisu akčního prostoru, dokážeme ovlivnit natočení kloubů. K tomu abychom však mohli určit jak nohu ohnout, je potřeba znát momentální stav. Dále agentovi předáme pozici těla Headcraba v prostoru a indikaci, zda se jeho nohy dotýkají země a zda trefil hlavu G-Mana. Výsledkem je dvaceti dimenzionální vektor.

Z popisu akčního a stavového prostoru jsme získali výsledné dimenze, tedy

$$a_t \in \mathbb{R}^{12} \tag{4.1}$$

$$s_t \in \mathbb{R}^{20}. \tag{4.2}$$

4.3.2 Funkce odměny

Dalším výstupem je odměna, kterou agent využívá k ohodnocení akcí a stavů. Při návrhu výše odměny se zaměříme na to, čeho chceme při učení dosáhnout. U skoku na hlavu G-Mana je nejdůležitější informace, zda se Headcrab hlavy dotkl. To je však pouze jednorázová akce a nemůže být samostatně použita jako odměna, jelikož by Headcrab nevěděl, kterým směrem se má pohybovat. Přidáme tedy i vzdálenost těla robota od hlavy. Tím je agent schopen určit cílovou pozici v prostoru. Označíme-li kolizi s hlavou jako *collision_rew* a vzdálenost *dist*, pak

$$collision\_rew = \begin{cases} 1 & \text{pokud došlo ke kolizi} \\ 0 & \text{jinak} \end{cases} \tag{4.3}$$

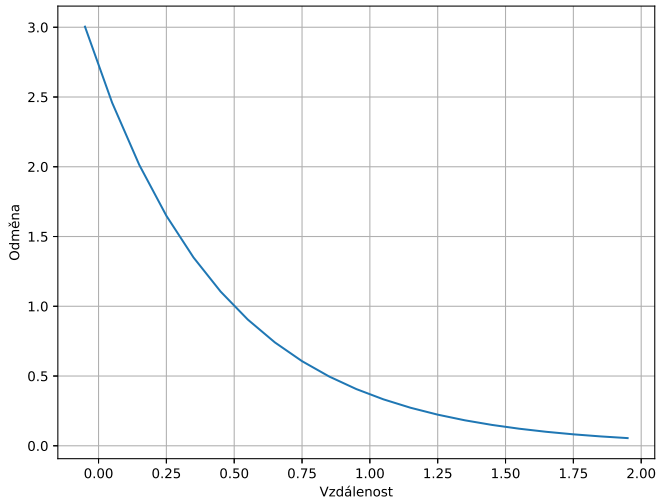
$$dist = \|(x, y, z) - (x_h, y_h, z_h)\| \tag{4.4}$$

kde (x, y, z) jsou okamžité souřadnice torza Headcraba a (x_h, y_h, z_h) jsou neměnné souřadnice hlavy G-mana. Ve výsledné odměně ale nemůžeme použít *dist_rew* ve tvaru 4.4, jelikož cílem je maximalizovat odměnu a euklidovská

vzdálenost je nulová při nulové vzdálenosti. Navíc chceme dát větší důraz na případy, kdy je Headcrab těsně u hlavy. K tomu využijeme převrácenou exponenciální funkci a odměnu za vzdálenost definujeme jako

$$dist\_rew = \frac{1}{\exp\left(\alpha(dist - \beta)\right)} \tag{4.5}$$

kde α udává rychlost stoupání a β posun. V naší implementaci jsme zvolili $\alpha = 2$ a $\beta = 0.5$. Výslednou funkci $dist\_rew$ vyobrazíme do grafu.



Obrázek 4.4: Závislost odměny na vzdálenosti

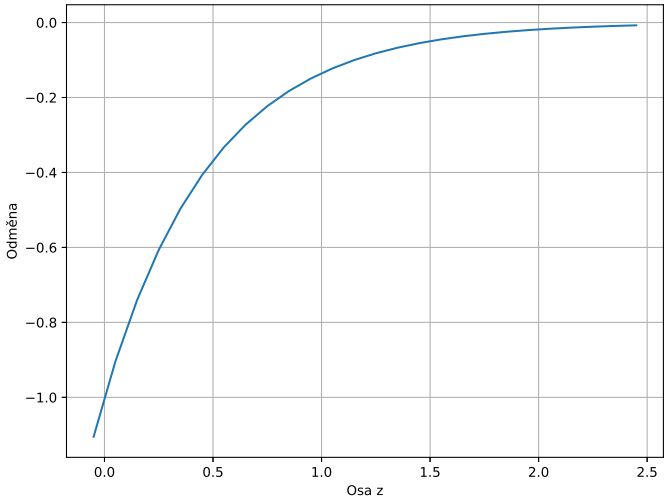
Nejdříve jsem vyzkoušeli pouze kombinaci $dist\_rew$ a $collision\_rew$, tedy

$$r = dist\_rew + collision\_rew \tag{4.6}$$

Headcrab měl však problém dostat se do odpovídající výšky z a proto jsme do odměny přidali trest, pokud zůstává příliš dlouho u země. Opět jsme využili převrácenou exponenciální funkci, ovšem její opačnou hodnotu, tedy pokud označíme z_rew penalizaci na ose z, dostaneme

$$z\_rew = -\frac{1}{\exp\left(\alpha(z - \beta)\right)} \tag{4.7}$$

v tomto případě jsme určili $\alpha = 2$ a $\beta = 0$



Obrázek 4.5: Závislost odměny na ose z

Výsledná funkce odměny má tvar

$$r = a \cdot dist\_rew + b \cdot z\_rew + c \cdot collision\_rew \tag{4.8}$$

kde $a, b, c \in \mathbb{R}$ škálují jednotlivé odměny, aby například trest za osu z nepřevažoval odměnu za vzdálenost. Pro naše účely jsme zvolili $a = 1.5$, $b = 0.3$ a $c = 2$.

Kapitola 5

Experimenty

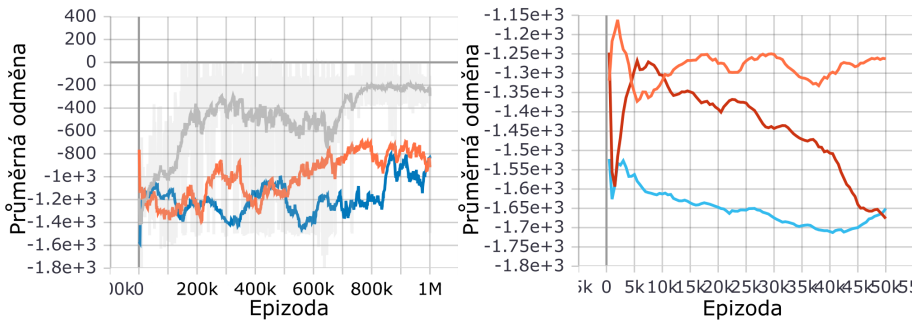
V kapitole 2 a 3 jsme teoreticky popsali algoritmy TRPO, DDPG, A2C a CMA-ES. V této části se zaměříme na porovnání těchto metod z knihoven stable-baselines a cmaes, nejdříve na několika prostředích poskytovaných OpenAI Gym knihovnou, a poté na prostředí Headcrab popsaném v kapitole 4. Nakonec se podíváme na implementaci vhodného algoritmu.

5.1 Porovnání metod v OpenAI Gym prostředí

Pro porovnání již hotových algoritmů z knihovny stable-baselines jsme vybrali dvě OpenAI Gym prostředí. Obě pracují se spojitým akčním prostorem a proto se na ně zaměříme. K vykreslení grafu využijeme knihovnu TensorBoard[21], která nám umožní vykreslit více průběhů učení do jednoho grafu.

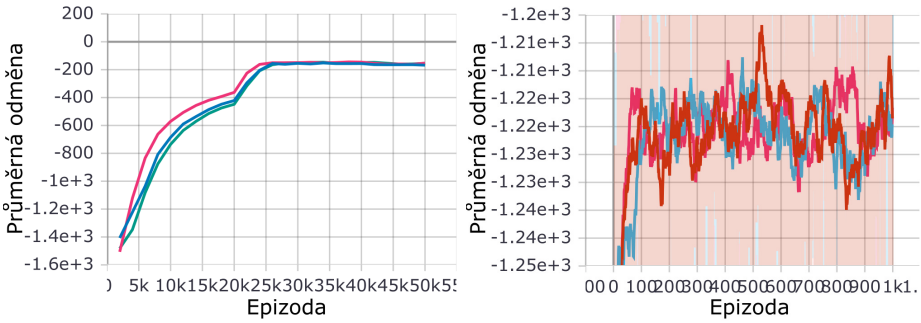
5.1.1 Pendulum-v0

Toto prostředí simuluje invertované kyvadlo[1]. Úkol agenta je roztočit kyvadlo tak, aby zůstalo ve vzpřímené poloze. Všechny algoritmy jsme spustili třikrát, abychom viděli i závislost na počátečních podmínkách, a průměrnou odměnu z každého průběhu jsme vnesli do obrázku 5.1 .



(a) : Průměrná odměna učení TRPO **(b) :** Průměrná odměna učení A2C

[1] `https://gym.openai.com/envs/Pendulum-v0/`



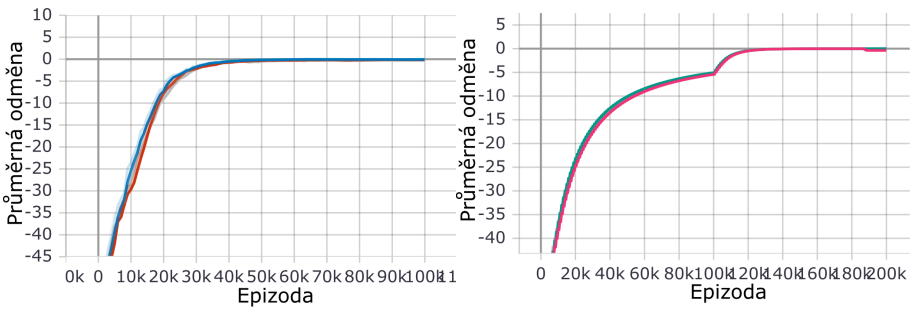
(c) : Průměrná odměna učení DDPG **(d) :** Průměrná odměna učení CMA-ES

Obrázek 5.1: Učení vybraných algoritmů v prostředí Pendulum-v0

V tomto prostředí byl úspěšný pouze algoritmus DDPG, a to ve všech třech případech. Pomocí této metody jsme dokázali vyhoupnout inverzní kyvadlo nahoru a udržet ho v vzpřímené poloze. TRPO algoritmus zvládl kyvadlo pouze roztočit a to jen při jednom učení (šedá linie). A2C ani CMA-ES algoritmus nedokázal najít pro prostředí Pendulum-v0 optimální řešení.

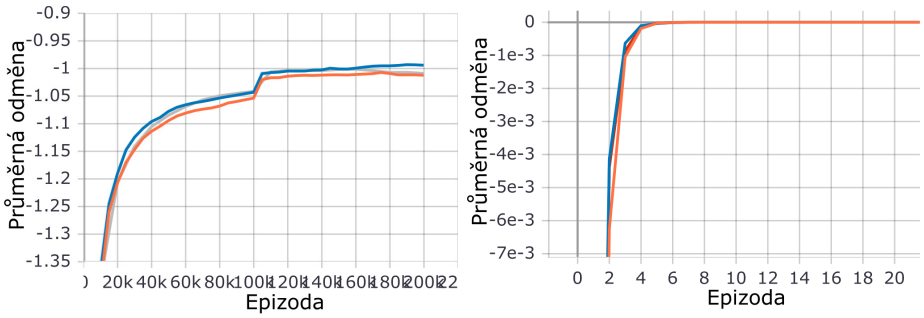
5.1.2 MountainCarContinuous-v0

Jako další příklad uvedeme prostředí, které simuluje auto v údolí, které má za úkol vyjet na kopec[2]. Jeho motor však není dostatečně silný a proto musí jezdit tam a zpátky, aby nabral dostatečnou rychlost. Opět jsme každý algoritmus spustili třikrát a průběhy jsme uvedli v obrázku 5.2



(a) : Průměrná odměna učení TRPO **(b) :** Průměrná odměna učení A2C

[2] `https://gym.openai.com/envs/MountainCarContinuous-v0/`



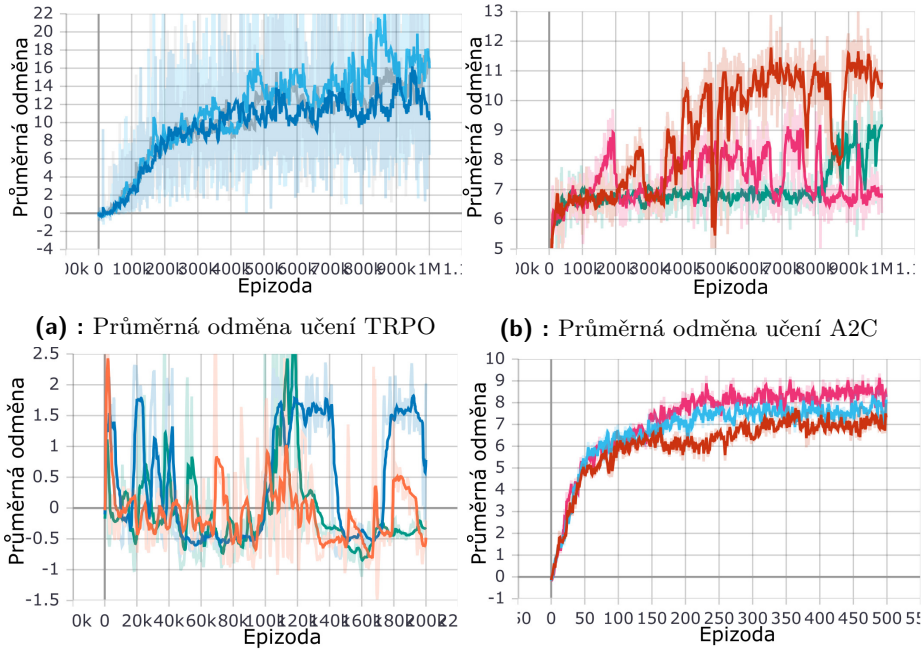
(c) : Průměrná odměna učení DDPG **(d) :** Průměrná odměna učení CMA-ES

Obrázek 5.2: Učení vybraných algoritmů v prostředí MountainCarContinuous-v0

V tomto prostředí nejlépe pracoval algoritmus TRPO, díky kterému auto vyjelo na vrchol kopce. I přes to, že na první pohled dosáhly algoritmy A2C a CMA-ES stejné průměrné odměny jako TRPO, výsledný model ani jednoho z nich nebyl schopen dostat auto na kopec. Pokud jsme u A2C a CME-ES zvýšili dobu učení, průměrná odměna obou algoritmů po nějaké době opět klesla. Metoda DDPG nenalezla optimální řešení.

5.2 Porovnání v prostředí Headcrab-v0

Nejdříve porovnáme již hotové algoritmy než se pustíme do vlastní implementace. I zde jsme pro každou metodu spustili učení třikrát a na obrázku 5.3 vidíme výsledky jednotlivých učení.



(a) : Průměrná odměna učení TRPO **(b) :** Průměrná odměna učení A2C

(c) : Průměrná odměna učení DDPG **(d) :** Průměrná odměna učení CMA-ES

Obrázek 5.3: Učení vybraných algoritmů v prostředí Headcrab-v0

Z uvedených výsledků vidíme, že jediný algoritmus, který nedokázal provést skok Headcraba na G-Mana je DDPG. V jeho případě se odměna po celou dobu učení pohybovala kolem nuly. U CMA-ES jsme dosáhli toho, že Headcrab v několika případech vyskočil a dotkl se hlavy, ale trajektorie nebyla dokonalá a při většině pokusů hlavu minul. Algoritmus A2C pracoval v tomto prostředí velmi dobře, ovšem učení bylo nestabilní a ne vždy jsme došli k žádanému výsledku. Nejlépe z vybraných algoritmů fungovala metoda TRPO. Učení bylo stabilní, pokaždé našlo vyhovující řešení a Headcrab se dokonce několikrát přichytil nohama k hlavě. Ve všech případech se Headcrab při letu točil, ale funkci odměny to neovlivňuje.

5.3 Implementace A2C

Pro vlastní implementaci jsme vybrali algoritmus A2C (bez paměti a s pamětí). V předchozí sekci jsme určili, že je s ním možné nalézt takovou strategii, při které Headcrab skočí na G-Mana. Myšlenkou je přizpůsobit algoritmus zadanému prostředí tak, aby se ve výsledcích přiblížil metodě TRPO. Princip A2C je popsán v sekci 2.5.1, přejdeme tedy rovnou k vlastní implementaci pomocí knihovny Pytorch[22]. Začali jsme trojvrstvou lineární sítí pro obě sítě (Actor, Critic) s vnitřní dimenzí 64. Nejdříve jsme jako aktivační funkci použili hyperbolický tangens, poté jsme ji nahradili funkcí Mish, která je definována v [23]. Výstupem neuronové sítě Actor je normální rozdělení $\mathcal{N}(\mu, \sigma^2 I)$, kde $\mu, \sigma \in \mathbb{R}^{12}$ a výstupem sítě Critic je výsledek V-funkce. Srážkový faktor γ jsme nastavili jako

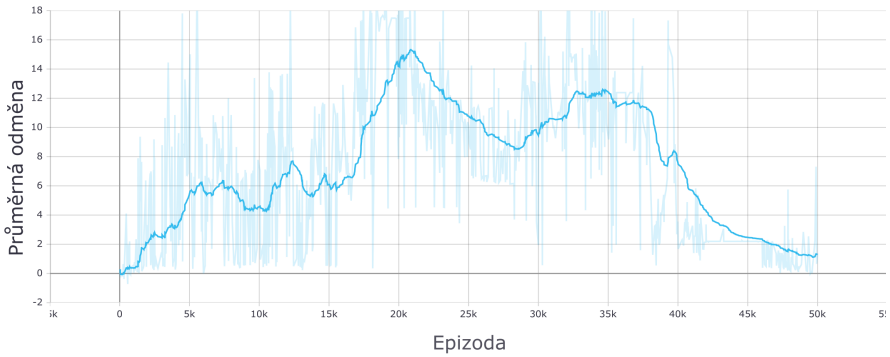
$$\gamma = 0.9 \tag{5.1}$$

K aktualizaci parametrů neuronových sítí jsme využili optimalizační algoritmus Adam [24] a faktory učení jsme nastavili následovně

$$\alpha_A = 0.0004 \tag{5.2}$$
$$\alpha_C = 0.004 \tag{5.3}$$

5.3.1 Bez paměti

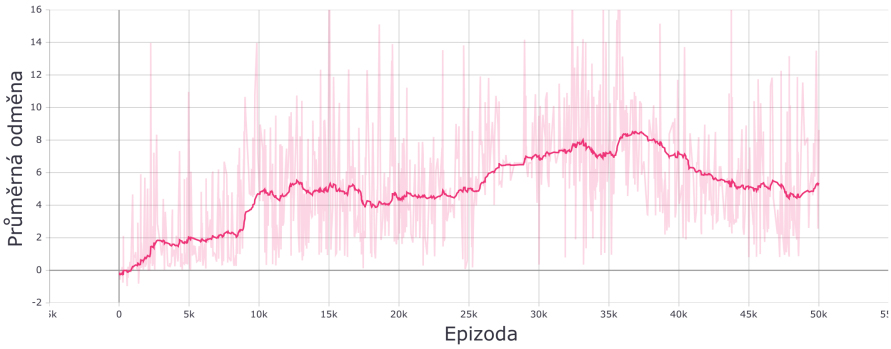
Vytvořili jsme základní učící algoritmus bez paměti, tedy Actor a Critic se aktualizují po každém kroku. I s touto jednoduchou implementací se nám podařilo dosáhnout cíle a Headcrab opakovaně skočil na hlavu G-Mana. Obrázek 5.4 znázorňuje průběh učení agenta, který dokázal skok provést a v některých případech se také na hlavě chvíli udržet.



Obrázek 5.4: Průběh učení u A2C algoritmu

Musíme si všimnout, že učení není příliš stabilní a agent se často při prozkoumávaní nových stavů odučí dobrou strategii. Pokud je odučení dobré strategie dočasné a agent se po nějaké době vrátí k původnímu řešení, naše učení to pouze zpomalí. Pokud dojde k tomu dojde na konci učení, jak je vidět na obrázku 5.4, negativně to ovlivní výsledek. Jestliže však ukládáme momentální nejlepší strategii během učení, výsledek není tímto ovlivněn.

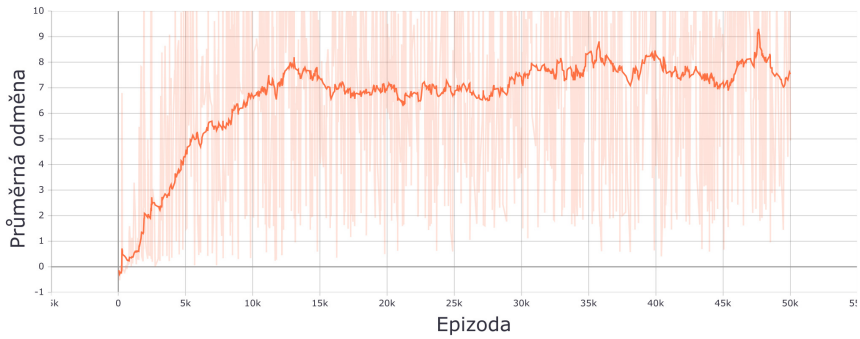
Velmi důležité jsou počáteční podmínky, které jsou nastavovány Pytorch knihovnou. Může nastat případ, že agent z daným nastavením nenalezne vhodnou strategii. Takový průběh znázorníme na obrázku 5.5



Obrázek 5.5: Průběh učení u A2C algoritmu

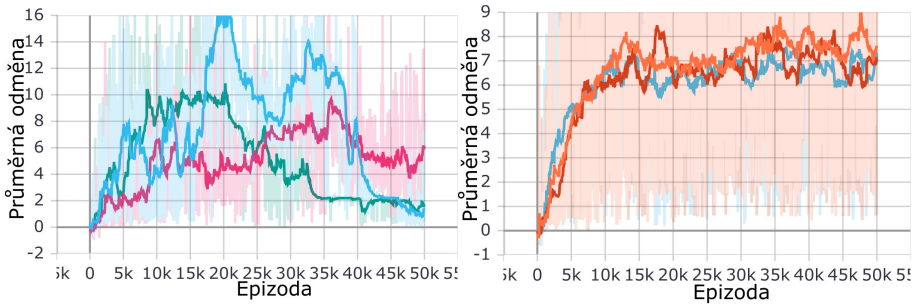
5.3.2 S pamětí

V rámci pokusu o stabilizaci učení jsme do programu přidali paměť. V této verzi se Actor a Critic až po pěti krocích simulace. Na základě jejich výsledků upravíme obě sítě. To způsobí, že okamžitá odměna sice velmi osciluje, ale průměrná odměna se stabilizuje a agent se neodnaučí již nalezenou strategii. Nemusíme ji tedy průběžně ukládat.



Obrázek 5.6: Průběh učení A2C s pamětí

Pro porovnání metody s pamětí a bez paměti jsme každý algoritmus spustili třikrát a výsledky vnesli do grafů.



(a) : Bez paměti **(b) :** S pamětí

Obrázek 5.7: Porovnání implementace A2C a pamětí a bez paměti

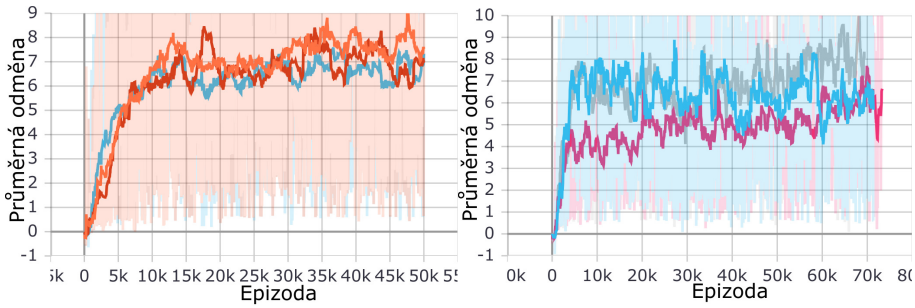
Vidíme, že metoda s pamětí je výrazně stabilnější než bez paměti a ve všech výsledcích učení se Headcrab při skoku dotkl hlavy G-mana. Můžeme si povšimnout, že v jednom případě bez paměti je na krátký čas průměrná odměna výrazně větší než v ostatních případech. S tímto jsme se při vlastní implementaci setkali pouze jednou. Označuje strategii, kdy se Headcrab kutálel podél hlavy dolů a získával tedy odměnu za kolizi s hlavou, ne pouze v jednom, ale v několika krocích. Tato situace se nám u algoritmů s pamětí nepodařila spolehlivě reprodukovat.

5.3.3 Mish funkce

Algoritmus jsme dále upravili změnou aktivační funkce z hyperbolického tangens na funkci Mish, která je definovaná jako

$$f(x) = x \tanh(softplus(x)) = x \tanh(\ln(1 + e^x)) \tag{5.4}$$

Touto úpravou chceme dosáhnout rychlejšího náběhu odměny a celkově vyšší průměrné odměny. Výsledky učení s upravenou aktivační funkcí a porovnání s původní implementací uvedeme v následujících grafech.



(a) : Aktivační funkce tanh

(b) : Aktivační funkce $Mish$

Obrázek 5.8: Porovnání A2C algoritmu s aktivační funkcí tanh a $Mish$

Z uvedených výsledků obrázku 5.8 vidíme, že nedošlo k výrazné změně při učení a tím i vyšší průměrné odměny.

Kapitola 6

Závěr

Cílem této práce bylo navrhnout prostředí simulující skok parazita Headcraba na hlavu postavy G-Mana a porovnat a implementovat vhodný algoritmus z oblasti posilovaného učení.

Úspěšně jsme vytvořili modely Headcraba a G-Mana pomocí programu Bledner, které jsme využili k tvorbě prostředí. Headcraba jsme motivovali ke skoku na hlavu G-mana funkcí odměny s exponenciálním základem, abychom dali důraz na stavy, ve kterých se Headcrab nachází v blízkosti hlavy.

Zadány byly čtyři algoritmy, Trust Region Policy Optimization (TRPO), Advantage Actor-Critic (A2C), Deep Deterministic Policy Gradient (DDPG) a Covariance Matrix Adaptation Evolution Strategy(CMA-ES). Tyto metody jsme nejdříve porovnali na dvou již existujících prostředích z knihovny OpenAI Gym se spojitým akčním prostorem. V prvním prostředí, Pendulum-v0, dosáhl nejlepších výsledku algoritmus DDPG a pro druhé prostředí, MountainCarContinuous-v0, se algoritmus TRPO prokázal jako nejvhodnější. Dané metody jsme dále otestovali na námi vytvořeném prostředí Headcrab-v0. V tomto případě opět nejlépe problém vyřešila metoda TRPO, díky které se Headcrab dokonce dokázal přichytit na hlavu G-Mana. Naopak nejhorší výsledky v tomto prostředí prokázala metoda DDPG.

Pro vlastní implementaci jsme zvolili algoritmus A2C, který měl sice, s porovnáním s TRPO, horší výsledky, ale nabídl nejlepší poměr mezi komplexitou implementace, výslednou funkčností a výpočetní náročnosti. Postupnými úpravami vybrané metody se nám podařilo dosáhnout lepší průměrné odměny a spolehlivějšího a stabilnějšího učení než u výsledků získaných implementací A2C z knihovny stable-baselines. I přes úpravu algoritmu A2C pro prostředí Headcrab-v0, metoda TRPO stále dosahovala lepších výsledků.

Do budoucna se zaměříme na vylepšení funkce odměny v prostředí Headcrab-v0 takovým způsobem, abychom omezili rotaci Headcraba při skoku jak v horizontální tak ve vertikální ose. Funkci odměny budeme i nadále upravovat tak, aby skok co nejlépe odpovídal předloze z počítačové hry Halflife.

Literatura

[1] M. Hénon, "The Monte Carlo Method," *International Astronomical Union Colloquium*, vol. 10, pp. 151–167, Nov. 1971. Publisher: Cambridge University Press.

[2] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," *arXiv:1602.01783 [cs]*, June 2016. arXiv: 1602.01783.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[4] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust Region Policy Optimization," *arXiv:1502.05477 [cs]*, Apr. 2017. arXiv: 1502.05477.

[5] R. S. Sutton and A. G. Barto, *Reinforcement Learning, second edition: An Introduction*. MIT Press, Nov. 2018. Google-Books-ID: uWV0DwAAQBAJ.

[6] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv:1509.02971 [cs, stat]*, July 2019. arXiv: 1509.02971.

[7] I. Loshchilov and F. Hutter, "CMA-ES for Hyperparameter Optimization of Deep Neural Networks," *arXiv:1604.07269 [cs]*, Apr. 2016. arXiv: 1604.07269.

[8] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey," *arXiv:cs/9605103*, Apr. 1996. arXiv: cs/9605103.

[9] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, May 1992.

[10] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *The Annals of Mathematical Statistics*, vol. 22, pp. 79–86, Mar. 1951. Publisher: Institute of Mathematical Statistics.

[11] R. Bellman, "Dynamic Programming," *Science*, vol. 153, pp. 34–37, July 1966. Publisher: American Association for the Advancement of Science Section: Articles.

[12] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms," in *International Conference on Machine Learning*, pp. 387–395, PMLR, Jan. 2014. ISSN: 1938-7228.

[13] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning." `http://pybullet.org`, 2016–2020.

[14] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[15] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[16] B. O. Community, *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018.

[17] K. von Szadkowski and S. Reichel, "Phobos: A tool for creating complex robot models," *Journal of Open Source Software*, vol. 5, no. 45, p. 1326, 2020.

[18] "Gman (From Half-life Alyx) - Download Free 3D model by Sandy_boi (@Sandy_boi) [e1d7293]." `https://sketchfab.com/models/e1d7293117e64f458e0d364869e16bae/embed?autostart=1`.

[19] Thingiverse.com, "Half-Life 2 Poison Headcrab: Low and High Resolution by BliNDF123."

[20] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines." `https://github.com/hill-a/stable-baselines`, 2018.

[21] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[22] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[23] D. Misra, "Mish: A Self Regularized Non-Monotonic Activation Function," *arXiv:1908.08681 [cs, stat]*, Aug. 2020. arXiv: 1908.08681.

[24] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv:1412.6980 [cs]*, Jan. 2017. arXiv: 1412.6980.

[25] M. Pecka, "Bezpečné autonomní posilované učení," May 2020. Accepted: 2020-09-28T19:35:32Z Publisher: České vysoké učení technické v Praze. Vypočetní a informační centrum.

[26] J. Mei, C. Xiao, C. Szepesvari, and D. Schuurmans, "On the Global Convergence Rates of Softmax Policy Gradient Methods," in *International Conference on Machine Learning*, pp. 6820–6829, PMLR, Nov. 2020. ISSN: 2640-3498.

[27] S. Gu, T. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine, "Q-Prop: Sample-Efficient Policy Gradient with An Off-Policy Critic," *arXiv:1611.02247 [cs]*, Feb. 2017. arXiv: 1611.02247.

[28] R. BELLMAN, "A Markovian Decision Process," *Journal of Mathematics and Mechanics*, vol. 6, no. 5, pp. 679–684, 1957. Publisher: Indiana University Mathematics Department.