VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

POSILOVANÉ UČENÍ PRO POHYB ROBOTA

REINFORCEMENT LEARNING FOR MOBILE ROBOTS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **DAVID HÁS**
AUTHOR

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

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Hás David**
Program: Informační technologie
Název: **Posilované učení pro pohyb robota**
Reinforcement Learning for Mobile Robots
Kategorie: Zpracování obrazu

Zadání:

1. Prostudujte základy neuronových sítí a posilovaného učení.
2. Vytvořte si přehled o současných metodách využívají neuronové sítě a posilované učení k naučení efektivních pohybů robota.
3. Připravte si prostředí ve fyzikálním simulátoru pro experimenty.
4. Vyberte konkrétní metodu a navrhněte experimenty.
5. Implementujte navrženou metodu a proveďte experimenty.
6. Porovnejte dosažené výsledky a diskutujte možnosti budoucího vývoje.
7. Vytvořte stručné video prezentující vaši práci, její cíle a výsledky.

Literatura:

- Lee et al.: Learning a family of motor skills from a single motion clip. ACM Trans. Graph., 2021.
- Peng et al.: DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills. ACM Trans. Graph., 2018.
- T. Lillicrap et al.: Continuous control with deep reinforcement learning. CoRR, 2016.
- J. Schulman et al.: High-Dimensional Continuous Control Using Generalized Advantage Estimation. ICLR, 2016.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Hradiš Michal, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 1. listopadu 2021

Abstrakt

Tato práce se zabývá využitím posilovaného učení pro pohyb robota v simulovaném fyzikálním prostředí. Pro posilované učení se jedná o náročné úlohy, kde agenti čelí několika výzvám. Jednou z nich je spojitý prostor akcí, jelikož agent obvykle ovlivňuje prostředí aplikací síly na jednotlivé klouby. Druhým problémem je, že části robota se často vzájemně ovlivňují složitým způsobem a navíc jsou ovlivněny gravitací, setrvačností a dalšími fyzikálními efekty. Z těchto a dalších důvodů nejsou pro tyto úlohy jednoduché algoritmy posilovaného učení vhodné. Jedním z relativně nových řešení je algoritmus Soft Actor-Critic (SAC), který se objevil současně s podobně performním TD3, a oba překonávají starší DDPG. Agent SAC je odměňován za více náhodné chování, jeho cílem je tedy kromě maximalizace odměny také maximalizace entropie. Tato práce ukazuje použití tohoto algoritmu při učení agenta na úloze robotického pohybu. Je popsána implementace s použitím frameworku PyTorch a úspěšnost algoritmu je vyhodnocena na úlohách z prostředí PyBullet a OpenAI Gym. Algoritmus je na závěr použit na vlastní upravené prostředí s robotem Atlas.

Abstract

This paper is concerned with reinforcement learning for robotic movement in simulated physical environment. These are difficult problems for reinforcement learning, where agents need to face several challenges. One of them is continuous action space, as agent usually interacts with the environment by applying force on joints of the robot. Another problem is that parts of the robot often affect each other in complex ways and are also affected by gravity, inertia and other physical effects. For these and more reasons simple reinforcement learning algorithms are not suitable for these tasks. One of recent solutions is the Soft Actor-Critic algorithm (SAC), which emerged at the same time as similarly performing TD3, and both outperforming the older DDPG. SAC agents are rewarded for behaving more randomly, thus their goal is to maximize entropy along with maximizing the reward. This paper describes usage of this algorithm for teaching agents robotic movement. It describes implementation of the algorithms using the PyTorch machine learning framework and evaluates it on environments from OpenAI Gym platform using the PyBullet physics engine. Lastly, the algorithm is applied on custom environment with robot Atlas.

Klíčová slova

strojové učení, neronové sítě, posilované učení, hluboké učení, soft actor-critic, SAC, OpenAI Gym

Keywords

machine learning, neural networks, reinforcement learning, deep learning, soft actor-critic, SAC, OpenAI Gym

Citace

HÁS, David. *Posilované učení pro pohyb robota.* Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Hradiš, Ph.D.

Posilované učení pro pohyb robota

Prohlášení

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

.....
David Hás
11. května 2022

Poděkování

Rád bych poděkoval Ing. Michalu Hradišovi, Ph.D. za trpělivost a odborné vedení. Dále děkuji kolegům z Phonexie za odborné rady a flexibilitu a své přítelkyni za podporu a gramatickou kontrolu.

Obsah

Kapitola 1

Úvod

Posilované učení je jeden z přístupů, jakým se umělá inteligence může sama naučit něco nového. Výhoda tohoto způsobu spočívá v tom, že nepotřebuje žádný vstup od uživatele, agent se totiž učí sám pouze na základě interakce s prostředím. Ve spojení se skutečnými roboty se toto učení používá spíš okrajově. Roboti se častěji manuálně programují, jsou-li známy jejich přesné parametry. Avšak v případě, kdy tomu tak není, může pomoci právě posilované učení.

Tato práce se věnuje metodám posilovaného učení a jejich aplikaci na pohyblivé roboty. Jedná se o náročný úkol, kde agent čelí několika výzvám. Za prvé, prostor akcí je v těchto úlohách spojitý, takže agent vybírá z nekonečně mnoha možností. Za druhé, agent má učení ztížené fyzikálními mechanismy, které na něj působí.

V současnosti se odborná veřejnost začíná zajímat o aplikaci metod posilovaného učení na reálné roboty. Umožnily to nové algoritmy, které se ukázaly být velice spolehlivé a robustní. Jedním z nich, metodou Soft Actor-Critic, se zabývá i tato práce.

Existují prostředí pro posilované učení se simulovanými roboty, které se dají pro trénování použít. Tato práce využívá framework OpenAI Gym s fyzikálním enginem PyBullet. Bohužel ty nejzajímavější úlohy, jako je třeba úloha naučit chodit simulovaný model skutečného robota, buď nejsou implementovány vůbec, nebo jsou pouze rozpracované a projekty zapomenuté. Pro větší zajímavost úlohy jsem se rozhodl opravit jedno takové nefunkční prostředí s modelem robota Atlas.

Práce obsahuje úvod do posilovaného učení a metod Actor-Critic. Dále probírá hluboké neuronové sítě a v podrobnosti se zabývá vybranou metodou Soft Actor-Critic. SAC je popsána nejdřív z teoretického hlediska a posléze popsána implementace metody a technické detaily. Nakonec je ukázáno použití metody na úlohách robotického pohybu včetně modelu reálného robota Atlase.

Kapitola 2

Posilované učení

Posilované učení je oblast strojového učení, jejímž cílem je naučit agenta chovat se optimálně v daném prostředí. To znamená, že agent se musí naučit vybírat takové akce, které povedou k maximalizaci získaných odměn. Subjekt neví, které akce jsou nejvýhodnější, a musí je objevit tím, že je bude náhodně zkoušet. Akce navíc nemusejí vést k odměně okamžitě, ale až po zvolení příští akce nebo později. Tato dvě specifika – metoda pokus-omyl a zpožděná odměna – jsou pro posilované učení charakteristická [14].

V české literatuře se setkáme také s názvem *zpětnovazební*. Kaelbling et al. [10] chápou pojem spíše jako třídu problémů, než soubor postupů. Naopak Sutton a Barto [14] píší, že se jedná jak o třídu problémů, tak o souhrn metod, které je řeší a zároveň o vědeckou oblast, která je studuje.

Strojové učení se často dělí na *učení s učitelem* a *učení bez učitele* podle toho, zda se algoritmus učí na označených nebo neoznačených datech. Posilované učení se nedá zařadit do ani jedné z těchto kategorií [10, 13]. Nejedná se o učení s učitelem, které vyžaduje soubor anotovaných dat a jeho cílem je soubor aproximovat tak, aby bylo možné správně zařadit nová data. Zároveň se nejedná ani o učení bez učitele, které si klade za cíl najít skrytou strukturu v kolekci neanotovaných dat [8, 14]. Je zřejmé, že z pohledu klasifikace se jedná o samostatnou oblast strojového učení.

Oblast posilovaného učení proniká do mnoha odlišných inženýrských a vědeckých disciplín jako je statistika, optimalizace a další matematická odvětví. Silně spolupracuje také například s psychologií a neurovědou [14].

Jednou z oblastí, pro které je posilované učení atraktivní, je i robotika. Pohyblivý robot není běžně ovládán umělou inteligencí, ale častěji softwarem vytvořeným na základě precizní znalosti struktury a fyzických parametrů robota. To mohou být například úhly mezi jednotlivými klouby, vzdálenosti mezi nimi, váhy, tření a podobně. Posilované učení pak může pomoci v případě, kdy tyto parametry nejsou známy. Jeho výhodou totiž je, že se učí na datech, a může dosáhnout požadovaného cíle metodou pokus-omyl[5].

2.1 Principy posilovaného učení

Kromě agenta a prostředí můžeme identifikovat čtyři hlavní prvky v posilovaném učení. Jsou to *strategie*, *odměna*, *hodnotová funkce*[1] a volitelně *model* prostředí [14].

[1] *Hodnotová funkce* se v české literatuře nazývá také *užitková funkce* nebo *funkce hodnoty*.

Strategie (anglicky *policy*) definuje, jak se agent v dané situaci chová. Dá se na ni pohlížet jako na zobrazení přiřazující stavu, v němž se prostředí právě nachází, akci, kterou agent zvolí [14].

V každém časovém kroku dostává agent od prostředí jediné číslo, které se nazývá *odměna*[2]. Ta definuje cíl řešeného problému [13]. Odpovídá tomu, jak dobře si agent v daném prostředí daří, a může být i negativní. Cílem je maximalizovat tuto odměnu v dlouhodobém měřítku, které může být i nekonečné – příkladem může být simulace chůze po dvou, kde je agent odměňován podle uražené vzdálenosti nebo času stráveného ve vzpřímené poloze. Na základě získané odměny agent upravuje svou strategii. Vedla-li vybraná akce k nízké odměně, může strategii změnit tak, aby příště byla vybrána akce jiná. Někdy se hovoří o *signálu odměny*, čímž je myšlena průběžná odměna.

Zatímco odměna indikuje, jak dobrá nebo špatná je situace v daný okamžik, *hodnotová funkce* říká, co je dobré v dlouhodobém měřítku. *Hodnota* stavu se dá chápat jako celková odměna, kterou může agent v budoucnu nasbírat. Ačkoliv cílem agenta je získat co největší *celkovou* odměnu, stačí mu zajímat se o *hodnotu* stavu [14]. Okamžitá odměna v daném stavu může být nízká, ale jeho hodnota přesto vysoká, neboť může vést do stavů s vysokou odměnou. Naopak jiný stav může agentovi dát vysokou odměnu, ale protože vede do stavů se špatnou odměnou, jeho hodnota je nízká.

Konečně *model* je něco, co napodobuje prostředí a snaží se předvídat, k jakému výsledku povede určitá akce zvolená v určitém stavu. Agent jej může využít k *plánování*, jako protiklad jednoduchému *procházení* prostředí. Může zvážit budoucí scénáře, aniž by je vyzkoušel. Metody posilovaného učení, které využívají model, se nazývají *model-based*, a metody, které jej nevyužívají, se nazývají *model-free* [14]. Jednou z výhod posilovaného učení je právě možnost jeho aplikace v případě, kdy přesný model prostředí není znám, a tímto způsobem je možné jej aproximovat.

Metoda použitá v této práci patří do skupiny *model-free*. Metody ze skupiny *model-based* zde nejsou podrobněji probrány, přehledově je však zmiňuje například Kaelbling et al. v sekci 5 [10], prakticky je pak využívá Kaiser et al. [11].

Někdy není jasné, kde se nachází hranice mezi agentem a prostředím. Nemusí být intuitivní a často není stejná jako například fyzická hranice lidského těla. Sutton a Barto [14] uvádějí obecné pravidlo, že vše, co agent nemůže libovolně měnit, je považováno za součást prostředí. Například motor a mechanické klouby robota jsou součástí prostředí, jelikož podléhají fyzikálním vlivům a agent je ovládá pouze skrze akce – nemůže je měnit libovolně. V tomto ohledu je také důležité zmínit, že odměna je vždy vypočítávána v prostředí a mimo agenta, neboť definuje problém, kterému agent čelí.

Obecně se předpokládá, že jak prostředí, tak strategie agenta, se mohou chovat nedeterministicky. Zvolení stejné akce ve stejném stavu může vést k různým výsledkům. Předpokládáme však, že prostředí je stacionární a že pravděpodobnosti jednotlivých přechodů se nemění [10]. Stejně tak strategie může mít ke každé akci v daném stavu přiřazenu pravděpodobnost jejího výběru [14].

[2] *Odměna*, v angličtině *reward*, se ve starší anglické literatuře nazývá *reinforcement*, například v práci od Kaelbling et al. [10] Odtud zřejmě pochází i anglický název oboru *reinforcement learning*.

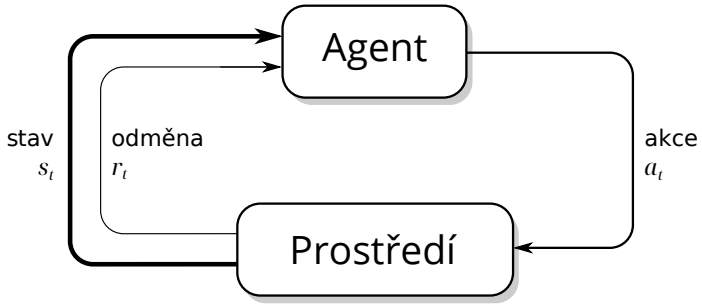
2.2 Posilované učení jako Markovův rozhodovací proces

Problém posilovaného učení je formalizován pomocí *Markovova rozhodovacího procesu*, neboli *MDP*[3]. Takto lze definovat problémy, v nichž je výsledek částečně náhodný a částečně pod kontrolou aktéra.

Takto Markovův rozhodovací proces definují Otterlo a Wiering[13]:

Definice 1 *Markovův rozhodovací proces je čtveřice* (S, A, T, R)*, kde* S *je konečná množina stavů,* A *je konečná množina akcí,* T *je přechodová funkce definovaná jako* $T : S \times A \times S \to \langle 0, 1 \rangle$ *a* R *je funkce odměn definovaná jako* $R : S \times A \times S \to \mathbb{R}$*.*

V případě posilovaného učení definujeme interakci mezi učícím se agentem a jeho prostředím. Tato interakce je znázorněna na diagramu 2.1.



Obrázek 2.1: **Posilované učení.** Interakce mezi agentem a prostředím. V čase t agent dostává odměnu r_t a popis stavu s_t a na základě toho vybírá akci a_t. Odměna je zpravidla skalární veličina, zatímco stav může být popsán vektorem.

Na základě diagramu lze popsat, jak probíhá interakce mezi agentem a prostředím. Děje se tak při každém ze sekvence diskrétních časových kroků $t = 0, 1, 2, ...$ V každém časovém kroku t agent dostává reprezentaci stavu prostředí $s_t \in S$ a na základě toho vybírá akci a_t. V dalším časovém kroku dostává odměnu $r_{t+1} \in R$ a reprezentaci stavu s_{t+1}.

Obecně může být přechod stochastický, stejně tak získaná odměna. Předpokládejme čas t, stavy $s_t, s_{t+1} \in S$, odměnu $r_{t+1} \in R$ a akci $a_t \in A(s_t)$. Pro každou kombinaci stavu s_{t+1} a odměny r_{t+1} existuje pravděpodobnost, že dojde k přechodu ze stavu s_t skrze akci a_t do stavu s_{t+1} se ziskem odměny r_{t+1}:

$$p(s_{t+1}, r_{t+1}|s_t, a_t) = Pr(s_{t+1}, r_{t+1}|s_t, a_t) \tag{2.1}$$

přičemž platí, že součet pravděpodobností všech přechodů v daném stavu a akci je roven jedné [14].

Přechody přitom splňují *Markovovu vlastnost*. To znamená, že distribuce pravděpodobností přechodů závisí pouze na aktuálním stavu, ne na předchozích. Aktuální stav tedy plně vypovídá o tom, kde se právě systém nachází. V důsledku to znamená, že systém se v čase nemění a v daném stavu bude rozdělení pravděpodobností vždy stejné.

Algoritmy posilovaného učení se snaží maximalizovat celkovou odměnu

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

[3] *MDP* – z anglického *Markov decision process*

Je žádoucí upřednostňovat odměny, které přijdou dříve, před odměnami, které se nacházejí dále v budoucnosti. Proto se zavádí diskontní faktor $\gamma \in \langle 0; 1)$, jehož hodnota je typicky číslo blízké 1. Diskontní faktor umožňuje rovnici použít i na nekončící úlohy. Celková budoucí diskontovaná odměna (anglicky *expected discounted return*) je pak definována vztahem[14]:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.3}$$

Některé stavy vedou k lepší odměně a jsou tak pro agenta řícího se strategií π výhodnější, než jiné. Tato myšlenka je formalizovaná pomocí **funkce hodnoty stavu**, která je rovna součtu všech budoucích odměn za předpokladu chování podle strategie π:

$$V_\pi(s_t) = \mathbb{E}[R_t|s_t] \tag{2.4}$$

Podobně se dá definovat **funkce hodnoty akce**, která ohodnocuje kvalitu akce v daném stavu:

$$Q_\pi(s_t, a_t) = \mathbb{E}[R_t|s_t, a_t] \tag{2.5}$$

Cíl posilovaného učení se dá zapsat jako hledání optimální strategie π:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, V_\pi(s), \tag{2.6}$$

tedy takové strategie π, která povede k nejvyšší odměně. V reálných úlohách bohužel přechodová funkce není známa, jinak by se jednalo o triviální problém. Častěji se tedy formuluje problém jako:

$$\pi^* = \underset{a}{\operatorname{argmax}}\, Q(s, a), \tag{2.7}$$

neboli hledání optimální akce pro každý stav. Obvykle není problém zjistit, jaký stav a odměna následují po zvolení dané akce v daném stavu. Pak lze iterativně získat Q hodnoty dalších stavů a akcí[14]. Výsledkem je Bellmanova rovnice, která je podstatou Q-učení:

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a') \tag{2.8}$$

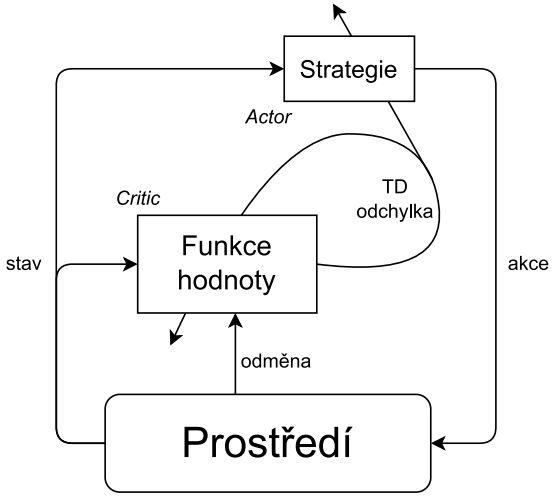
2.3 Metody Actor-Critic

Metody Actor-Critic jsou metody založené na tzv. *temporal difference* (*TD*). Jejich podstata je v oddělení strategie od funkce hodnoty, takže se trénují jako dva nezávislé aproximátory. Model, který se učí strategii, se nazývá *actor*, protože se používá k výběru akcí. Model, který se učí odhadovat hodnotovou funkci se nazývá *critic*. Jeho úkolem je hodnotit akce actora. Učení je vždy *on-policy*, což znamená, že critic vždy hodnotí akci, kterou actor právě vybral. Tento skalární signál je jeho jediným výstupem a má na starosti učení obou modelů[14]. Viz obrázek 2.2.

Po každém kroku critic vyhodnotí nový stav a rozhodne, zda je lepší nebo horší než předchozí. Výsledek je *temporal difference error*:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{2.9}$$

Tato odchylka pak může být použita pro vyhodnocení právě vybrané akce, neboli akce a_t zvolené ve stavu s_t. Pokud je odchylka kladná, znamená to, že strategie by měla posílit pravděpodobnost výběru akce a_t. Pokud je odchylka záporná, strategie by měla pravděpodobnost akce a_t snížit[14].



Obrázek 2.2: **Actor-Critic.** Odchylka vypočítaná modelem Critic je použita k aktualizaci obou sítí.

2.4 Neuronové sítě v hlubokém posilovaném učení

Umělá neuronová síť je matematický model, který je podobný biologickým neuronovým sítím. Základní výpočetní jednotka neuronové sítě je neuron. Neuron přijme vstup x a zpracuje jej tak, že jej vynásobí svým váhovým vektorem $\mathbf{w}$ a přičte k němu bias b:

$$z = \sum_i w_i x_i + b. \tag{2.10}$$

Na výstup je pak aplikována aktivační funkce f, která musí být diferencovatelná, a vypočítá tak aktivaci neuronu:

$$y = f(z) \tag{2.11}$$

Aktivační funkce slouží k zanesení nelineárnosti do neuronové sítě a tím umožňuje vyřešit nelineární problémy. Mezi nejběžnější aktivační funkce patří ReLU (Rectified Linear Unit), tanh nebo sigmoid. U posledních dvou je problém s takzvaným „mizejícím gradientem“, proto ustoupily do pozadí před ReLU, která tento problém nemá. Její další výhoda je nízká výpočetní náročnost, má však nevýhodu spočívající v tendenci „umírat“, tzn. začít produkovat nulový signál bez možnosti se někdy z tohoto stavu dostat.

Neurony se řadí do vrstev, které mohou být mezi sebou různě propojeny. Jedním z častých způsobů propojení je *úplné*[4], kde je každý neuron jedné vrstvy spojen s každým neuronem druhé vrstvy.

Spojení vrstev dohromady se nazývá neuronová síť. Ta má vždy daný počet vstupů a daný počet výstupů. Ostatní vrstvy se nazývají *skryté*. Síť se nazývá *hluboká*, pokud má alespoň jednu skrytou vrstvu.

Neuronové sítě jsou universální aproximátory, a proto jsou ideální k aproximaci strategií a hodnotových funkcí v posilovaném učení. Konvoluční neuronové sítě mohou být zase využity pro zpracování obrazu. Skládají se z trojrozměrných filtrů, které jsou posouvány po vstupním obrazu a hledají v něm naučené vzory.

Neuronové sítě pracují s tzv. ztrátovou funkcí (*loss function*). Těch existuje mnoho druhů, mezi ty nejznámnější patří *mean squared error (MSE)* a *cross entropy*. Klíčovým

[4] Anglicky *fully connected layer*.

pojmem je zde zpětná propagace (*backpropagation*). Jedná se o metodu, která má na starosti učení neuronových sítí. Nejprve je proveden *forward pass*, čili data přivedena na vstup sítě, a spočítán výsledek. Určí se velikost chyby a provede se *backward pass*. Pozpátku se prochází výpočetní strom a vypočítává se gradient, který představuje vliv jednotlivých vah na chybu. Nakonec je gradient použit pro aktualizaci vah za účelem snížení chyby.

Kapitola 3

Soft Actor-Critic

Soft Actor-Critic (SAC)[9] je relativně nový algoritmus, který se pyšní efektivitou a stabilitou a přibližuje použití posilovaného učení na nesimulovaná skutečná prostředí[15]. Jeho hlavní myšlenkou je maximalizace entropie spolu s odměnou, to znamená, že jeho cílem je chovat se tak náhodně, jak je to možné a stále se snažit získat co nejvyšší odměnu. Metoda je navržena pro prostředí se spojitým prostorem akcí, je proto ideální k použití při trénování pohybu robota.

Předchozí nejúspěšnější algoritmy jako *Trust Region Policy Optimization* (*TRPO*), *Proximal Policy Optimization* (*PPO*) nebo *Asynchronous Advantage Actor-Critic* (*A3C*) patří mezi on-policy algoritmy. To znamená, že po každé aktualizaci strategie potřebují nové vzorky, a nejsou tedy příliš efektivní. Oproti tomu algoritmy typu off-policy jako *Deep Deterministic Policy Gradient* (*DDPG*) nebo *Twin Delayed Deep Deterministic Policy Gradient* (*TD3*) se učí velmi efektivně použitím starých vzorků uložených v přehrávací paměti. Jejich problém však spočívá v křehkosti a vysoké náchylnosti na správné hyperparametry[15].

Soft Actor-Critic řeší problémy obou skupin. Jedná se také o off-policy metodu, takže může využít přehrávací paměti a učit se ze vzorků efektivně. Zároveň je velmi stabilní díky snaze maximalizovat entropii, neboli náhodnost, ve své strategii. To úzce souvisí s kompromisem mezi průzkumem a užitkem (*exploration-exploition trade-off*). Zvýšení entropie vede k více prozkoumávání, díky čemuž agent předchází tomu, že se zasekne v lokálním optimu[3].

Algoritmus předpokládá spojitý prostor akcí. Místo aby ale vybíral přímo akci, vrací parametry normálního rozdělení, ze kterého je pak akce náhodně vybrána. Maximalizace entropie v tomto případě pak jednoduše znamená, že se agent snaží vybrat co nejširší rozdělení. Pokud by více akcí vedlo ke stejné vysoké odměně, agent by jim přiřadil stejnou pravděpodobnost.

3.1 Definice cíle

Standardní posilované učení maximalizuje očekávanou sumu odměn[3]:

$$\sum_t \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t,a_t)] \tag{3.1}$$

Zaveďme nyní entropii $\mathcal{H}$ jako funkci distribuce pravděpodobnosti P a proměnné x takto:

$$\mathcal{H}(P) = \mathop{\mathbb{E}}_{x\sim P}[-\log P(x)] \tag{3.2}$$

Pak můžeme použít novou definici cíle, který se snaží maximalizovat součet odměny a entropie:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t,a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))] \tag{3.3}$$

kde α je regularizační koeficient entropie (v původní práci *temperature parameter*). Ten udává důležitost entropie oproti odměně a tím ovládá stochasticitu naučené strategie. Dle autorů algoritmu se jedná o nejdůležitější hyperparametr a z jejich zkušenosti často jediný, který je nutno nastavovat[9]. Pomocí diskontního koeficientu lze cíl rozšířit i na nekonečné problémy.

3.2 Model algoritmu SAC

Metoda Soft Actor-Critic, kterou Haarnoja et al. popisují, se od ostatních algoritmů liší také v tom, že používá odděleně modely pro aproximování funkce hodnoty stavu (V) a funkce hodnoty akce (Q). Principiálně v tom není rozdíl, nicméně dle autorů to zvyšuje stabilitu algoritmu[9].

Soft Actor-Critic tedy používá tři sítě[1]: funkci hodnoty stavu V s parametrem ψ, funkci hodnoty akce Q s parametrem θ a funkci strategie π s parametrem ϕ.

Následující popis parametrů a jejich ztrátových funkcí je pro stručnost ochuzen o některé derivační kroky, které nejsou k pochopení algoritmu podstatné. Dychtivý čtenář je odkázán na originální práci autorů algoritmu SAC[9], kde jsou jednotlivé kroky odvození popsány podrobně. Implementace však vychází s různými optimalizacemi z následujících rovnic.

Funkce hodnoty stavu V

Funkce hodnoty stavu V je trénována, aby minimalizovala následující odchylku:

$$J_V(\psi) = \mathop{\mathbb{E}}_{s_t\sim\mathcal{D}}\left[\frac{1}{2}(V_\psi(s_t) - \mathop{\mathbb{E}}_{a_t\sim\pi_\phi}[Q_\theta(s_t,a_t) - \log\pi_\phi(a_t|s_t)])^2\right] \tag{3.4}$$

kde $\mathcal{D}$ je rozdělení dřívěji navštívených stavů a akcí, neboli přehrávací paměť[2]. Jedná se o složitý zápis popisující jednoduše to, že skrze všechny vzorky vybrané z přehrávací paměti je nutné vypočítat rozdíl mezi předpovědí funkce V a očekávanou předpovědí funkce Q, tento rozdíl umocnit na druhou a přičíst entropii strategie π.

Parametry ψ funkce V pak aktualizujeme pomocí gradientu rovnice 3.4:

$$\hat{\nabla}_\psi J_V(\psi) = \nabla_\psi V_\psi(s_t)(V_\psi(s_t) - Q_\theta(s_t,a_t) + \log\pi_\phi(a_t|s_t)) \tag{3.5}$$

Funkce hodnoty akce Q

Funkce hodnoty akce Q je trénována minimalizovat následující odchylku:

$$J_Q(\theta) = \mathop{\mathbb{E}}_{(s_t,a_t)\sim\mathcal{D}}\left[\frac{1}{2}\Big(Q_\theta(s_t,a_t) - \hat{Q}(s_t,a_t)]\Big)^2\right], \tag{3.6}$$

přičemž

$$\hat{Q}(s_t,a_t) = r(s_t,a_t) + \gamma \mathop{\mathbb{E}}_{s_{t+1}\sim p}[V_{\bar{\psi}}(s_{t+1})]. \tag{3.7}$$

[1] SAC ve skutečnosti používá sítí pět, což je v této práci probráno později.

[2] *Přehrávací paměť* – z anglického *replay buffer*.

Minimalizace proběhne takto: pro všechny vybrané páry $(stav, akce)$ z přehrávací paměti je minimalizován rozdíl predikce funkce Q a okamžité odměny r, plus diskontovaná očekávaná hodnota následujícího stavu.

Funkce V je zde parametrizována $\bar{\psi}$, což značí *cílovou* funkci hodnoty stavu. K aktualizaci parametrů je pak použit následující gradient:

$$\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta(a_t, s_t)\big(Q_\theta(s_t, a_t) - r(s_t, a_t) - \gamma V_{\bar{\psi}}(s_{t+1})\big) \tag{3.8}$$

Funkce strategie π

Funkce strategie π je trénována minimalizací následující odchylky:

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}}\left[\mathrm{D}_{\mathrm{KL}}\left(\pi_\phi(\cdot|s_t) \,\middle\|\, \frac{\exp(Q_\theta(s_t, \cdot))}{Z_\theta(s_t)}\right)\right] \tag{3.9}$$

$\mathrm{D}_{\mathrm{KL}}(x \parallel y)$ zde značí Kullbackovu-Leiblerovu divergenci, zhruba řečeno míru odlišnosti dvou proměnných. Snažíme se tedy snížit odlišnost, neboli přiblížit hodnotu funkce strategie π k výrazu $\frac{\exp(Q_\theta(s_t,\cdot))}{Z_\theta(s_t)}$. Výraz $Z(x)$ značí takzvanou *partition function*[3], která nepřispívá do gradientu podle parametru ϕ a může být ignorována[9].

Aby mohl být tento cíl minimalizován, používají Haarnoja et al. „reparametrizační trik". Ten slouží k tomu, aby výběr z distribuce pravděpodobnosti (*sampling*) byl diferencovatelný a byla umožněna zpětná propagace. Spočívá v reparametrizaci funkce strategie takto:

$$a_t = f_\phi(\epsilon_t; s_t) \tag{3.10}$$

kde ϵ_t je vstupní šum pocházející z nějakého rozdělení pravděpodobnosti. Po odstranění funkce $Z_\theta(s_t)$, která nezávisí na parametru ϕ, je možno rovnici 3.9 přepsat takto:

$$J_\pi(\phi) = \mathop{\mathbb{E}}_{\substack{s_t \sim \mathcal{D} \\ \epsilon_t \sim \mathcal{N}}}[\log \pi_\phi(f_\phi(\epsilon_t; s_t)|s_t) - Q_\theta(s_t, f_\phi(\epsilon_t; s_t))], \tag{3.11}$$

s gradientem:

$$\hat{\nabla}_\phi J_\pi(\phi) = \nabla_\phi \log \pi_\phi(a_t|s_t) + (\nabla_{a_t} \log \pi_\phi(a_t|s_t) - \nabla_{a_t} Q(s_t, a_t))\nabla_\phi f_\phi(\epsilon_t; s_t) \tag{3.12}$$

3.3 Algoritmus SAC

Algoritmus Soft Actor-Critic je vylepšen ještě několika způsoby. Jedním z nich je použití dvou sítí pro odhad hodnoty stavu, V_ψ a $V_{\bar{\psi}}$. Váhy *cílové* sítě $\bar{\psi}$ jsou aktualizovány vahami *lokální* sítě ψ. Aktualizace přitom může být průběžná, kdy váhy cílové sítě tvoří průměr vah lokální sítě, nebo nárazová, kdy se v pravidelných intervalech váhy lokální sítě nakopírují na cílovou síť. Tato cílová síť je pak použita pro generování hodnot v rovnici 3.8. Bylo ukázáno, že tento přístup stabilizuje učení [12].

Druhé vylepšení spočívá v použití dvou sítí pro odhad hodnoty akce, Q_{θ_1} a Q_{θ_2}. Obě jsou nezávisle na sobě trénovány minimalizovat rovnici 3.6. Jako Q hodnota je pak vždy použito minimum z hodnot vrácených těmito dvěma sítěmi. Tento přístup řeší problém nadhodnocování Q hodnot[7].

Algoritmus dále využívá paměť, kam si ukládá zkušenosti přitom, co interaguje s prostředím. Zkušenost se skládá z původního stavu, zvolené akce, nového stavu, odměny a binární

[3] *Partition function* – autor práce kvůli nejasnému překladu ponechává anglický název.

hodnoty, která udává, zda akcí přešlo prostředí do konečného stavu. Paměť má omezenou velikost, takže nové zkušenosti nahrazují ty staré. V každém kroku učení je zvolen náhodný soubor zkušeností, což má za důsledek zvýšenou efektivitu učení, protože zkušenosti jsou použity mnohokrát. Dobrá vlastnost je i ta, že vzorky nejsou po sobě následující, nejsou korelované. Tento přístup lze ještě vylepšit tak, že se každé zkušenosti přiřadí priorita na základě toho, kolik by se z ní agent mohl naučit. Zajímavějším zkušenostem je přiřazena vyšší pravděpodobnost, že budou náhodně vybrány. Tento přístup se nazývá *Prioritized Experience Replay*.

Na závěr kapitoly je v algoritmu 1 uveden Soft Actor-Critic v jednotlivých krocích.

Algorithm 1 Soft Actor-Critic. Převzato z [9].

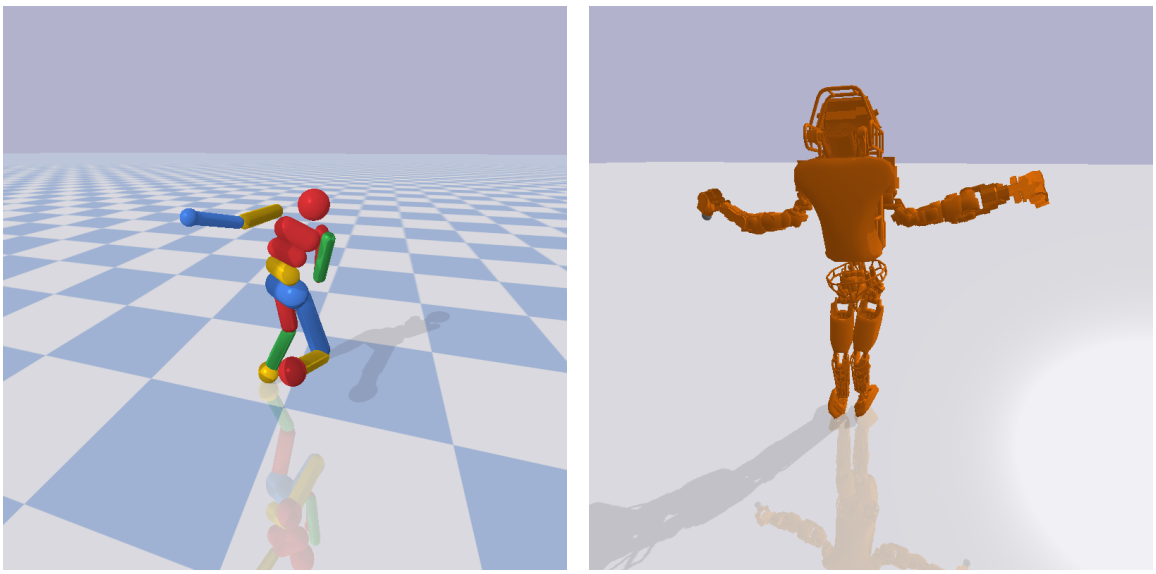
Inicializuj vektory parametrů ψ, $\hat{\psi}$, θ, ϕ.
for každá iterace **do**
  for každý krok prostředí **do**
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
  end for
  for každý krok gradientu **do**
    $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
    $\bar{\psi} \leftarrow \tau\psi + (1 - \tau)\bar{\psi}$
  end for
end for

Kapitola 4

Návrh řešení

Algoritmus Soft Actor-Critic dosahuje velmi dobrých výsledků v prostředích se spojitým prostorem akcí. Autoři algoritmu jeho výkonnost předvedli na simulovaných fyzikálních prostředích, kde předčil ostatní současné metody[9]. Šlo o úlohy se simulovanými roboty, avšak i prostředí, které bylo označeno za náročné, se stále skládalo ze zjednodušeného modelu humanoidního robota (viz obrázek 4.1 vlevo) s relativně málo klouby (viz tabulku 4.1). Z původní práce tedy není patrné, jak by si algoritmus vedl na prostředí, které je blízké skutečnému robotu, nebo dokonce v prostředí reálném. Tato práce si proto klade za cíl natrénovat algoritmus SAC na úloze pohybu modelu skutečného robota, jako je například Atlas, v simulovaném fyzikálním prostředí.

Tento úkol s sebou nese několik problémů. První, více zřejmý, souvisí se složitostí prostředí, neboť zvyšující se počet stupňů volnosti logicky klade větší nároky na algoritmus a snižuje jeho schopnost se efektivně učit. Druhý, méně zřejmý problém, je špatná dostupnost open-sourcových prostředí, ve kterých je možné roboty trénovat.



Obrázek 4.1: Příklady prostředí. Vlevo `HumanoidBulletEnv-v0` z balíčku PyBullet, vpravo opravené a funkční prostředí `AtlasPyBulletEnv-v0` z balíčku PyBullet Gymperium.

4.1 Prostředí PyBullet a OpenAI Gym

OpenAI Gym[4] je platforma pro vývoj a porovnávání algoritmů pro posilované učení. Jeho cílem je standardizovat prostředí použitá při vyhodnocování algoritmů ve výzkumných pracích[2]. Obsahuje široké spektrum úloh od jednoduchých hříček typu CartPole, kde agent vybírá jednu ze dvou možných akcí, přes hry z rodiny Atari 2600 až po robotický pohyb, ať už ve 2D či 3D. Pro robotiku je nutné nainstalovat zároveň fyzikální engine. Oficiálně je zřejmě doporučován program MuJoCo, který je ale proprietární a nabízí pouze zkušební verzi zdarma. Zároveň ale OpenAI na několika místech uvádí, že nyní doporučuje namísto toho PyBullet, který je open-source[1].

OpenAI v jednu chvíli vydal prostředí Roboschool, které mělo za cíl přenést úlohy z proprietárního MuJoCa do open-source. Projekt byl ale brzy opuštěn ve prospěch PyBullet. V tomto balíčku opravdu jsou některá prostředí k dispozici, zvláště ta jednoduchá. Například prostředí s Atlasem ale implementováno není.

Existuje ještě repozitář PyBullet Gymperium[6], který zřejmě implementuje některá další prostředí pro OpenAI Gym, ale byl podle všeho v průběhu vývoje opuštěn a je částečně zastaralý. Právě prostředí s Atlasem je zde ale částečně implementováno. Obsahuje však kritické chyby způsobující chybné chování fyzikálního modelu a není proto použitelné. Vzhledem k tomu, že je kód tohoto repozitáře veřejný, podařilo se mi jeho kód v rámci této práce nakonec opravit a prostředí `AtlasPyBulletEnv-v0` zprovoznit. Patch je součástí odevzdaného adresáře jako položka v souboru `requirements.txt`[1].

Prostředí pro OpenAI Gym sdílejí stejné rozhraní, takže pokud je kód napsán obecně, je možné jej použít na trénování libovolného prostředí předaného argumentem. Všechna prostředí se chovají podle principů posilovaného učení – přijmou akci a vrátí nový stav, respektive pozorování nového stavu, odměnu, a k tomu ještě informují agenta, zda přešel do koncového stavu nebo nikoliv. Přitom rozměr pozorování a akce se může mezi prostředími lišit. Jednodušší prostředí budou mít rozměry nižší. Tato práce se zabývá pouze prostředími, která mají spojitý prostor akcí, jelikož jsou obecně pro agenta náročnější. Jak je vidět v tabulce 4.1, algoritmus bude vyhodnocen na šesti různých prostředích, jejichž náročnost stoupá. Přitom prvních pět prostředí se shoduje s prostředími použitými v práci Haarnoji et al. Výsledky jsou ale porovnatelné jen orientačně, neboť ačkoliv se jedná o stejný typ úlohy, nejsou známy přesné parametry prostředí, jež byla použita při vyhodnocení.

4.2 Návrh agenta a neuronové sítě

Jako framework, nad kterým byl program postaven, byl vybrán stále populárnější PyTorch. Nabízí vhodné abstrakce nad nízkoúrovňovými komponentami, což umožňuje, aby kód nad ním napsaný byl jednoduchý a přehledný. Výpočetní graf vytvářený dynamicky za běhu umožňuje snadné debugování, což zase usnadňuje a urychluje implementaci.

Metoda Soft Actor-Critic používá velice jednoduchý model neuronové sítě. Všechny sítě Actor, Critic i Value jsou postaveny na obyčejných dvou plně propojených lineárních skrytých vstvách (`torch.nn.Linear`), každá po 258 skrytých neuronech, oddělených aktivačními funkcemi typu ReLU (Rectified Linear Unit).

Síť Value dostává na vstupu pouze stav a jejím výstupem je skalární veličina V popisující jeho hodnotu. Critic přijímá na vstupu dvojici stav a akce. To může být implementováno více způsoby. Critic může dostat dvojici na vstupu hned u první vrstvy, nebo může dostat

[1] https://github.com/hasdavid/pybullet-gym

v první vrstvě jen stav a až v další vrstvě akci. Na výkon modelu by to nemělo mít velký vliv. Výstupem sítě Critic je opět skalární veličina, tentokrát vyjadřující hodnotu Q dvojice stav–akce.

Teprve až síť Actor je trochu složitější. Na vstupu má opět stavový vektor, ale na výstupu má dvě hlavy. Jedna hlava totiž produkuje střední hodnoty μ a druhá směrodatné odchylky σ. Tyto hodnoty popisují normální rozdělení, z nichž jsou pak akce náhodně vybrány. Přitom každá hlava produkuje jednu hodnotu pro každý rozměr akce, výstupem je tedy dvakrát tolik hodnot, než kolik má akce rozměrů.

Následné zpracování výstupu sítě Actor také není triviální. Je nutné se totiž postarat o to, aby při výběru akce z normálního rozdělení nedošlo k přerušení grafu toku gradientů, čímž by se zabránilo zpětné propagaci a nemohlo by dojít ke korektní aktualizaci vah sítě. Náhodný výběr přitom právě graf přeruší. Použije se zde proto takzvaný *reparametrizační trik*, jak je zmíněno v sekci 3.2. Ten spočívá v tom, že místo toho, aby se vzorek z rozdělení pravděpodobnosti vybral náhodně, použije se přidaná veličina, která slouží jako šum. Ta samotná může být už vybrána náhodně. Dále není žádoucí, aby výsledná akce měla příliš divoké hodnoty. Využita je proto funkce tanh(), která výsledek hyperbolicky omezí na interval $(-1; 1)$. Pak je možné výsledný interval snadno upravit, pokud si prostředí žádá jiný rozsah. Výsledná funkce pro zpracování výsledku sítě Actor může být zapsána takto:

$$\widetilde{a}_\theta(s, \xi) = \tanh(\mu_\theta(s) + \sigma_\theta(s) \odot \xi), \text{ kde } \xi \sim \mathcal{N}(0, 1). \tag{4.1}$$

Přidaná veličina ξ je vybrána z Gaussova rozdělení[3]. Ve frameworku PyTorch je možné si ještě zjednodušit práci a použít vestavěnou funkcionalitu třídy `Distribution`. Ta umožňuje vybrat vzorek buď běžným způsobem pomocí metody `sample()`, nebo právě pomocí reparametrizačního triku metodou `rsample()`, která by se dala přeložit jako „sample using reparametrization trick“.

Implementovány byly všechny optimalizace popsány v sekci 3.3: přehrávací paměť s výchozí kapacitou 10^6, dvojitá síť Q_{θ_1}, Q_{θ_2} a soft-aktualizace parametrů cílové Value sítě $V_{\bar{\psi}}$ pomocí parametru τ, jehož výchozí hodnota je 0.005. Cílová síť je v každém kroku aktualizována podle rovnice $\bar{\psi} = \tau\psi + (1-\tau)\bar{\psi}$ a tvoří tak průběžný exponenciální průměr lokální sítě V_ψ. Hodnota parametru τ podle počátečních experimentů není příliš významná a byla ponechána po celou dobu na výchozí hodnotě.

Byly testovány architektury s jednou a třemi skrytými vrstvami a různým počtem neuronů. Byla vyzkoušena i implementace například bez reparametrizačního triku nebo s Actorem, který přímo generoval akce, namísto parametrů normálního rozdělení. Krátké experimenty ukázaly, že tyto změny mají buď nezřetelný, nebo i záporný vliv na efektivitu tréninku.

Pokud je trénování agenta z nějakého důvodu přerušeno, agent uloží svůj stav a program pak může pokračovat tam, kde naposledy skončil. Není však reálné, aby vždy uložil na disk celou svou přehrávací paměť a ta byla tedy vždy restartována. To se ukázalo jako problém, protože agentovo průměrné skóre obvykle po restartu hluboce propadlo. Jednou změnou, která toto pomohla vyřešit, bylo zavedení počátečního průzkumu prostředí (v programu `INITIAL_LEARNING`) a nastavení na hodnotu 10000. Agent se tedy začal znovu učit až ve chvíli, kdy se jeho paměť naplnila aspoň na tento počet vzorků, což zabránilo ostrému propadu úspěšnosti jeho strategie.

Ačkoliv je trénování možné provádět i na CPU, stroj s grafickou kartou a podporou frameworku CUDA je praktická nezbytnost. Pro trénování modelu k tomuto projektu byla s užitkem využita platforma Google Cloud[2]. Uživatel si zde může vytvořit virtuální stroj

[2] https://cloud.google.com/

Tabulka 4.1: Parametry trénovaných prostředí

Prostředí	Rozměr pozorování	Rozměr akce
`HopperBulletEnv-v0`	15	3
`Walker2DBulletEnv-v0`	22	6
`HalfCheetahBulletEnv-v0`	26	6
`AntBulletEnv-v0`	28	8
`HumanoidBulletEnv-v0`	44	17
`AtlasPyBulletEnv-v0`	70	33

a používat jej k libovolnému účelu, například pro trénování neuronových sítí, což je jeden z případů užití, který je v platformě podporován out-of-the-box. Uživatel se tak nemusí zdržovat instalací potřebných knihoven, což je zrovna u systému CUDA mnohdy časově náročný proces. Virtuální stroj v cloudu se hodí i v případě, že uživatel sám vlastní použitelný stroj. Cloud může využít na paralelní trénování dvou agentů a rychleji tak zvalidovat některé parametry. Dalším potenciálním použitím je třeba také distribuované trénování.

Kapitola 5

Experimenty

V této kapitole je výše navržený agent Soft Actor-Critic podroben experimentům a jsou prezentovány jejich výsledky. Měřítkem je odměna, kterou se agentovi podařilo od prostředí získat, načrtnutá oproti uběhlým krokům. Původní nápad vypisovat odměnu proti uplynulým epizodám se ukázal trochu nešťastný, protože délka epizod někdy divoce kolísá a často se drasticky zvyšuje, což graf zkresluje. Grafy jsou nakonec vykresleny proti počtu uběhlých kroků.

Experimenty jsou vyhodnoceny na šesti různých úlohách. Ve všech prostředích dostává agent odměnu za uraženou vzdálenost vpřed. Detaily se liší, ale dotyk země něčím jiným než chodidlem obvykle agenta penalizuje a většinou i ukončuje epizodu. Všechna prostředí také končí epizodu po 1000 krocích.

Hopper je nejjednodušší z použitých prostředí. Jedná se jen o jednu končetinu se třemi klouby. Ideální pohyb je skákání dopředu. Prostředí je ve 2D. Pád ukončuje epizodu.

Walker2D je o něco složitější než Hopper, co se týše prostoru akcí. Prostředí je stále ve 2D, ale agent už má k dispozici obě nohy. Ideální pohyb je chůze nebo běh. Pád ukončuje epizodu.

HalfCheetah je zvíře mající jednu nohu vepředu a jednu vzadu. Prostředí je 2D. Agent by se měl naučit sprintovat. Pokud se tělo dotýká země, dostává postih.

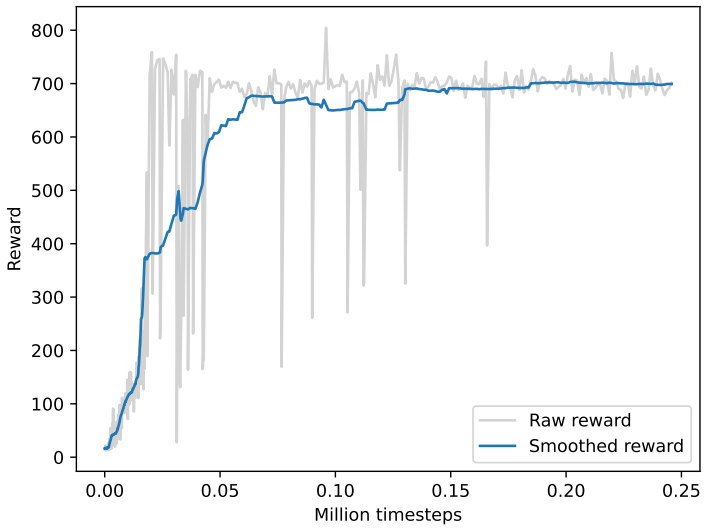
Ant je první 3D prostředí. Jedná se o kulovité tělo vybavené čtyřmi ortogonálními nohami. Za dotyk země tělem dostává negativní odměnu. Model má úmyslně zvýšenou váhu, aby agent musel používat všechny končetiny. Dřívější verze prostředí, kde byl lehčí, vedly na strategie, kdy agent vyhodil dvě protilehlé nohy do vzduchu a pohyboval se pouze zbylýma dvěma.

Humanoid je jednoduchá postavička připomínající stickmana ve 3D. Agent se musí naučit chodit po dvou. Toto prostředí má i složitější varianty jako Flagrun, kdy postava nemá za cíl běžet dopředu, ale k vlajce, jejíž pozice se změní vždy, když k ní doběhne. Agent se v tomto prostředí naučí zatáčet a zvedat po pádu ze země. Existuje také HardcoreFlagrun, kde je agent navíc bombardován projektily. Tato dvě pokročilá prostředí nejsou v této práci testována a jsou uvedena jen pro zajímavost.

Atlas je podobně jako Humanoid dvounohý robot. Cílem je naučit se chodit po dvou co nejrychleji, co nejdál a neztratit přitom rovnováhu.

Trénování probíhalo na instanci Google Cloud s grafickou kartou NVIDIA Tesla T4 a na vlastním stroji s kartou NVIDIA GeForce GTX 1660 Ti.

5.1 Hopper

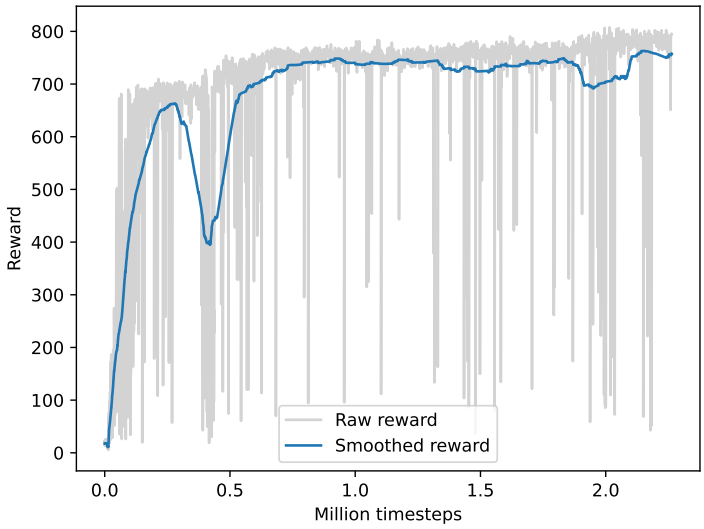


Obrázek 5.1: `HopperBulletEnv-v0`

Nejjednodušší z trénovaných prostředí. Jak je vidět z tabulky 4.1, agent má k dispozici pouze tři klouby, se kterými může manipulovat. Jedná se pouze o jednu končetinu, která se navíc může pohybovat pouze dopředu a dozadu, prostředí jí nedovolí spadnout na stranu. Je s podivem, že ačkoliv agent dosáhl stabilní relativně dobré odměny, nepodařilo se mu přesáhnout pomyslnou hranici 1000.0. Při sledování jeho pohybu je vidět, že se pouze sune pomalu dopředu aniž by spadl, což mu zajišťuje konstantní signál odměny, ale nenaučil se skákat, přestože mu to název prostředí napovídá. Agent nedosáhl hodnot, jakých na tomto prostředí vykazují Haarnoja et al., což je také zvláštní, protože by oba agenti měli mít podobné hyperparametry. Agent by možná dosáhl lepších výsledků s vyšší hodnotou *reward scale*. Sami autoři algoritmu jej označují za jediný hyperparametr, který je nutno ladit, a tento výsledek to může potvrzovat. Tento agent se učil s *reward scale* 2.0, zatímco referenční implementace s hodnotou 5.0. Náš agent se také učil kratší dobu, ale vzhledem k tomu, jaký tvar má učící křivka, zřejmě by delší učení o moc lepších výsledků nedosáhlo.

Nicméně, i na tomto prostředí je vidět počáteční velmi rychlé učení.

5.2 Walker2D

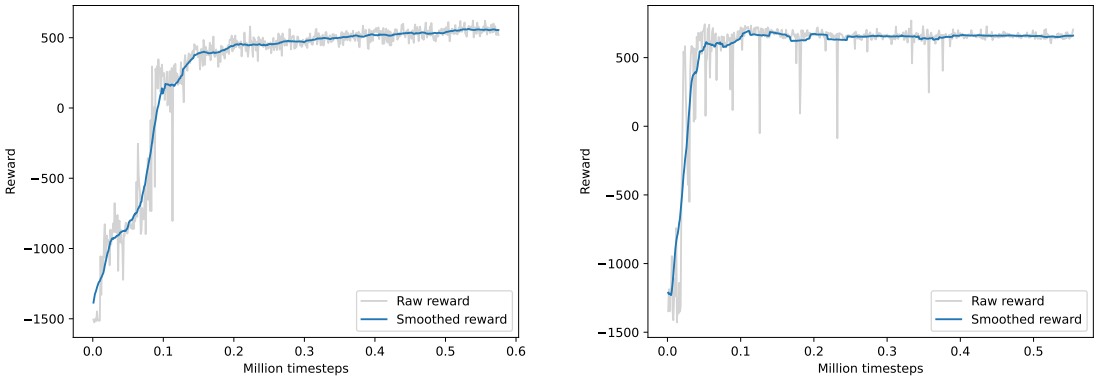


Obrázek 5.2: `Walker2DBulletEnv-v0`

Agent zde dostal druhou končetinu, stále se ale jedná o pohyb ve 2D prostředí. Výsledek je velmi podobný jako u předchozího úkolu – agent se rychle přiblíží tisícové hranici, není ji však schopen překročit, přestože referenční implementace s tím problém nemá. Při sledování, jak si daří, se ukazuje, že se drží vzpříma a téměř se nehýbe, jen šoupe nohama dopředu. Jako v předchozím případě je možné, že se agent zasekl v lokálním maximu a zvýšená hodnota *reward scale* by mu pomohla se z něj dostat.

Mám ale zároveň pocit, že toto prostředí není příliš dobře definované, neboť agent dostává vysoké odměny za téměř nulový pohyb.

5.3 HalfCheetah



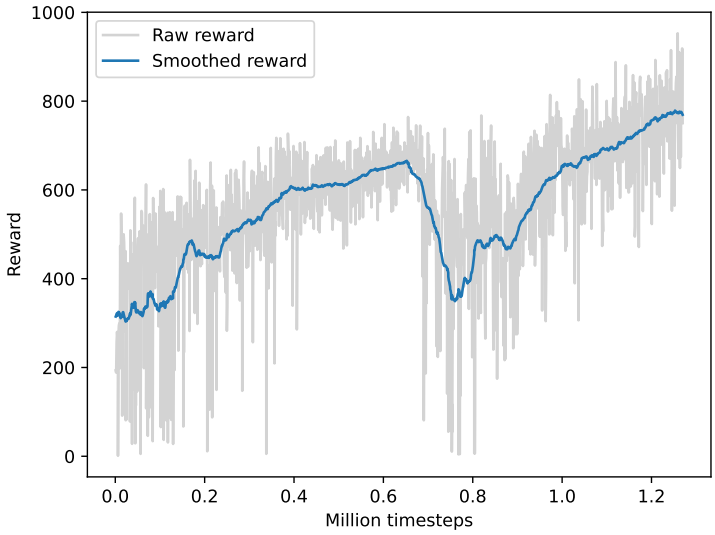
Obrázek 5.3: `HalfCheetahBulletEnv-v0`

Stále 2D prostředí, tentokrát agent manipuluje s dvounohým zvířetem.

Po zvláštních zkušenostech s předchozími dvěma prostředími bylo toto prostředí učeno dvakrát poměrně dlouhou dobu s dvěma odlišnými hodnotami parametru *reward scale* za účelem zjistit, jestli dosáhnou jiných výsledků. Graf vlevo ukazuje učení s hodnotou 1.0 a graf vpravo s hodnotou 10.0. Je dobře vidět, že vyšší hodnota umožnila agentovi učit se rychleji, ale stále mu neumožnila najít efektivní metodu pohybu. Prostředí penalizuje agenta, pokud se jeho hlava dotýká země. Agent si tedy někdy dřepne s hlavou těsně nad zemí a plazí se dopředu.

Za povšimnutí stojí i to, že vysoká hodnota reward scale vede k téměř deterministické strategii a agent už se nic neučí, jeho křivka je rovná. Naopak agent s nízkou hodnotou parametru stále vykazuje vzestupný trend.

5.4 Ant



Obrázek 5.4: `AntBulletEnv-v0`

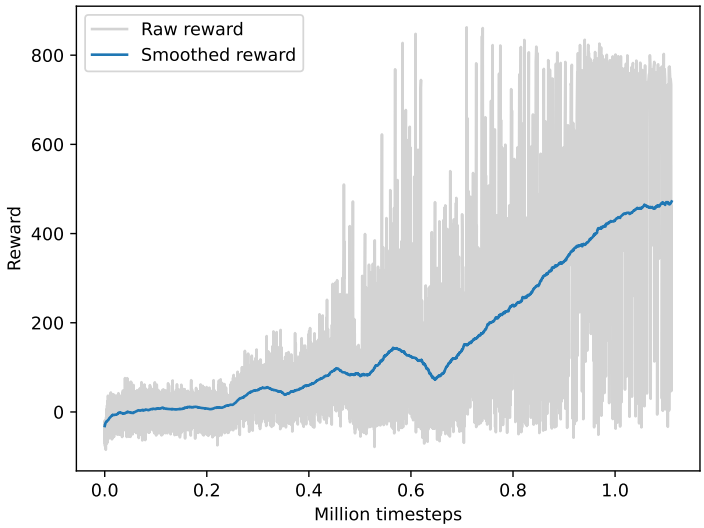
Čtyřnohý živočich ve 3D prostředí.

U tohoto experimentu je poprvé vidět velmi neintuitivní trend, který prochází všechna prostředí, na kterých jsem algoritmus trénoval. Vypadá to, že čím složitější prostředí, tím lepších výsledků agent dosáhne. Všechny předchozí pokusy vyvrcholily v neuspokojivé strategie, přestože jejich složitost byla nižší. Teprve v tomto prostředí agent ukazuje zajímavou strategii a jeho snažení vyústilo v byť neohrabanou, ale přece jen, chůzi.

Za vysvětlení stojí propad úspěšnosti v druhé půlce učení. Ten způsobilo přerušení učení kvůli změně reward scale. Vypnutí agentovi vymazalo paměť a po znovuzapnutí se začal učit s prázdnou. Dobře je na tom vidět užitečnost přehrávací paměti. Jakmile se agent začal učit na pouze recentních korelovaných vzorcích, jeho strategie se o hodně zhoršila. Následně bylo nastaveno minimální zaplnění paměti na 10.000 vzorků.

Je vidět, že učení by klidně mohlo ještě pokračovat a agent by se ještě zlepšil.

5.5 Humanoid



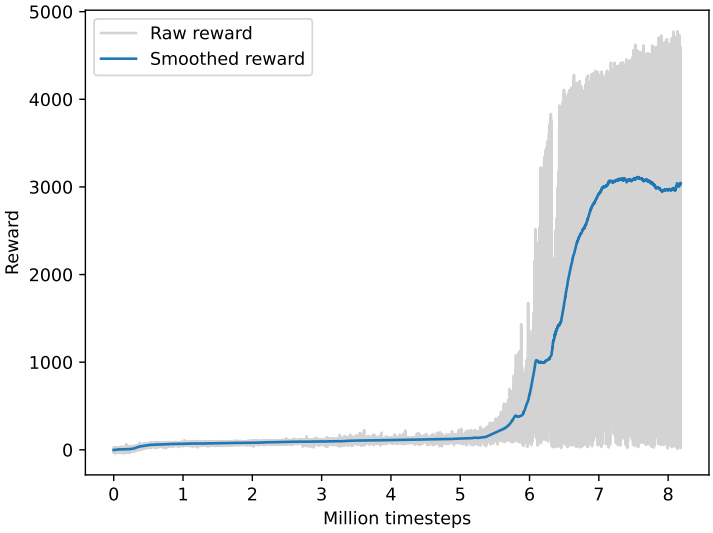
Obrázek 5.5: `HumanoidBulletEnv-v0`

Dvounohý jednoduchý humanoidní robot.

Tomuto prostředí jsem věnoval zvýšenou pozornost, jelikož jsem si kvůli jeho složitosti nebyl jist, zda se agent bude schopen něco zajímavého naučit. Je vidět, že začátek byl poměrně pomalý, ale to je u složitějšího prostředí očekávatelné. Poklesy v úspěšnosti ukazují přerušení učení a tím pádem vymazání zásobníku. Účelem zase bylo upravit hodnotu reward scale a zjistit, zda to bude mít na učení vliv. Z grafu to nyní vypadá, že změna hodnoty vliv možná ani neměla, a agent by se bez přerušení dostal na stejné místo, jen rychleji. Na konci je vidět pokles úhlu stoupání. To indikuje, že agent možná dosahuje svých limitů v tomto prostředí.

Při renderování naučeného modelu jsem byl překvapen, že agent skutečně chodí, přestože po maličkých krůčcích.

5.6 Atlas



Obrázek 5.6: `AtlasPyBulletEnv-v0`

Trénování Atlase bylo extrémně zajímavé. Nevím ani o žádné referenční implementaci, takže nebylo jisté, jestli se nejedná o příliš složité prostředí. Přece jen má asi dvojnásobný akční prostor oproti předchozímu prostředí, které je samo o sobě docela složité. Navíc jsem prostředí Atlase musel sám částečně upravit, takže nebylo otestované.

Agent se trénoval sám asi týden v Google Cloudu. Neočekával jsem, že bude konvergovat k nějakému zajímavému výsledku, ale velice pomalý zvyšující se trend šel pozorovat. Nechal jsem ho tedy běžet, protože jsem chtěl vidět, k jaké hodnotě se dostane. Přitom jsem předpokládal, že se naučí nanejvýš možná spadnout na správnou stranu.

Opravdu jsem nečekal, že po pěti milionech krocích začne zničehonic sám od sebe konvergovat. Pravděpodobně se v jednu chvíli musel naučit klíčovou schopnost, která mu umožnila objevovat nové vysoce odměňované stavy.

Při sledování jeho pohybu se ukazují další zajímavé poznatky. Atlas se například naučil velice šikovně využívat své flexibilní ruce k udržování rovnováhy. Ve výchozí poloze je má natažené v pravém úhlu rovně do stran od těla. Využívá rychlé svižné pohyby lokty a pěstmi, aby se švihem posunul potřebným směrem a zabránil ztrátě rovnováhy. Atlas je takto schopen udržet rovnováhu téměř po celou dobu běhu epizody a přitom slušnou rychlostí jít dopředu, defacto tím úlohu jako jediný z robotů, kteří byli v rámci této práce trénováni, vyřešil.

Zajímavé je i to, že se naučil úplně opačné chování, než Humanoid, který drží ruce blízko u těla.

Tři viditelné propadliny v pravé části grafu jsou opět způsobeny restartováním učení.

5.7 Vyhodnocení

Závěry experimentů jsou pro mě překvapující. Dopadly přesně naopak, než jsem očekával. Z jednoduchých prostředí, se kterými jsem začínal, nebyl žádný pokus příliš úspěšný. Naopak robot Atlas si jako jediný vedl velmi dobře a jeho učení se dá považovat za úspěch.

Několik poznatků je dokonce záhadných. Bylo by zajímavé zjistit, jak přimět agenty na prostředích Hopper, Walker a HalfCheetah, aby objevili efektivní strategii. Přitom by se neměli moc lišit od referenční implementace, což také přidává na záhadnosti.

Jedním z důvodů, proč byl Atlas úspěšnější než ostatní prostředí je jistě doba trénování. První tři prostředí, ale vypadala, že dosáhla stropu a delší doba učení by jim nejspíš nepomohla.

Kapitola 6

Závěr

Cílem této práce bylo použít algoritmus Soft Actor-Critic na prostředí simulující pohyb robota a natrénovat v něm agenta, který bude prostředí umět řešit. Zaměření padlo na robota Atlase a manuálně upravené prostředí `AtlasPyBulletEnv-v0`. Byl navrhnut a implementován agent používající algoritmus SAC a navzdory očekávání na složitém prostředí velmi úspěšně natrénován. Experimenty sestávaly i z trénování agentů na dalších, jednodušších prostředích, na které se nedostalo tolik výpočetní síly a jejich příslušní agenti si s nimi trochu paradoxně poradili méně úspěšně.

V teoretické části práce byly probrány základy posilovaného učení a neuronových sítí. Detailně je rozebrána vybraná metoda Soft Actor-Critic.

Agent naučený na robotovi Atlas dosáhl skvělého průměrného výsledku přes 3000 skóre po 7 milionech krocích. Další dva agenti trénovaní na 3D úlohách, Humanoid a Ant, dosáhli průměrného skóre 500, popř. 800, což není špatný výsledek vzhledem k tomu, že byli trénovaní jen 1 milion kroků.

Bylo by velmi zajímavé zanalyzovat proces učení Atlase a zkusit přijít na to, co vedlo k tak atypické učicí křivce. Zároveň bych chtěl zjistit, jak by si vedl nový čerstvý agent trénovaný na tomto prostředí. Jestli bude konvergovat rychleji, stejně, nebo třeba vůbec. Každopádně mám v plánu nechat Atlase ještě nějakou dobu trénovat a pozorovat, zda se jeho výsledek ještě zlepší. Další záhadou je, proč se agenti ve 2D prostředích pokaždé zasekli na určité hranici, zatímco stejní agenti na složitějších 3D úlohách takový problém neměli.

Použitý algoritmus je možné vylepšit různými optimalizacemi. Jedna z nich je třeba prioritizovaná přehrávací pamět (prioritized replay buffer). Porovnání se současným stavem by mohlo přinést zajímavé výsledky.

Literatura

[1] Roboschool. *OpenAI* [online]. [cit. 2022-04-23]. Dostupné z: https://openai.com/blog/roboschool/.

[2] Getting Started with Gym. *OpenAI / Gym* [online]. [cit. 2022-04-22]. Dostupné z: https://gym.openai.com/docs/.

[3] Soft Actor-Critic. *OpenAI Spinning Up* [online]. 2018 [cit. 2022-04-22]. Dostupné z: https://spinningup.openai.com/en/latest/algorithms/sac.html.

[4] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J. et al. OpenAI Gym. Leden 2016, [cit. 2022-04-17]. Dostupné z: https://arxiv.org/abs/1606.01540.

[5] Di Palo, N. Making a robot learn how to move. *Towards Data Science* [online], 16. července 2017 [cit. 2022-04-18]. Dostupné z: https://towardsdatascience.com/making-a-robot-learn-of-to-move-intro-2bcf3c3330df.

[6] Ellenberger, B. *PyBullet Gymperium*. 2018-2019. Dostupné z: https://github.com/benelot/pybullet-gym.

[7] Fujimoto, S., Hoof, H. van a Meger, D. Addressing Function Approximation Error in Actor-Critic Methods. Únor 2018, [cit. 2022-04-23]. Dostupné z: https://arxiv.org/abs/1802.09477.

[8] Goodfellow, I., Bengio, Y. a Courville, A. *Deep Learning*. MIT Press, listopad 2016 [cit. 2022-03-26]. Adaptive Computation and Machine Learning. ISBN 978-0-262-03561-3.

[9] Haarnoja, T., Zhou, A., Abbeel, P. a Levine, S. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. Leden 2018, [cit. 2022-04-22]. DOI: 10.48550/ARXIV.1801.01290.

[10] Kaelbling, L. P., Littman, M. L. a Moore, A. W. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*. Květen 1996, sv. 4, s. 237–285, [cit. 2021-05-23]. DOI: 10.1613/jair.301.

[11] Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H. et al. Model-Based Reinforcement Learning for Atari. Březen 2019, [cit. 2022-04-23]. DOI: 10.48550/ARXIV.1903.00374.

[12] Mnih, V., Kavukcuoglu, K., Silver, D. a al. et. Human-level control through deep reinforcement learning. *Nature*. 2015, s. 529–533, [cit. 2022-04-23]. DOI: 10.1038/nature14236. ISSN 0028-0836.

[13] Otterlo, M. a Wiering, M. Reinforcement Learning and Markov Decision Processes. *Reinforcement Learning: State of the Art*. Leden 2012, s. 3–42, [cit. 2021-05-17]. DOI: 10.1007/978-3-642-27645-3_1.

[14] Sutton, R. S. a Barto, A. G. *Reinforcement Learning: An Introduction*. 2. vyd. MIT Press, listopad 2018 [cit. 2022-03-26]. Adaptive Computation and Machine Learning. ISBN 978-0-262-03924-6.

[15] V.Kumar, V. Soft Actor-Critic Demystified: An intuitive explanation of the theory and a PyTorch implementation guide. *Towards Data Science* [online], 8. ledna 2019 [cit. 2022-04-18]. Dostupné z: https://towardsdatascience.com/soft-actor-critic-demystified-b8427df61665.

Příloha A

Obsah přiloženého paměťového média

```
├── bp/         Zdrojové soubory technické zprávy.
├── sac/        Zdrojové soubory vytvořeného programu.
│   ├── core/               Jádro programu.
│   ├── trained/            Adresář s natrénovanými modely.
│   ├── main.py             Vstupní bod programu.
│   ├── README.md           Popis instalace a použití programu.
│   └── requirements.txt    Soubor s instalačními požadavky.
└── projekt.pdf             Tato technická zpráva.
```