VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

KNN - Projekt 2021/2022

Metody Actor Critic pro spojitá prostředí

Martin Kostelník, Michal Glos, Michal Szymik

Přiložené materiály: Github projektu

1. Úvod

Cílem tohoto projektu bylo vytvořit vlastní implementaci algoritmů posilovaného učení (RL), především Actor-Critic pro spojitá prostředí. Jsou využity prostředí knihoven OpenaI Gym[1] a dm_control [3]. S výslednou implementací je prováděno několik sad experimentů, které demonstrují úspěšnost implementace.

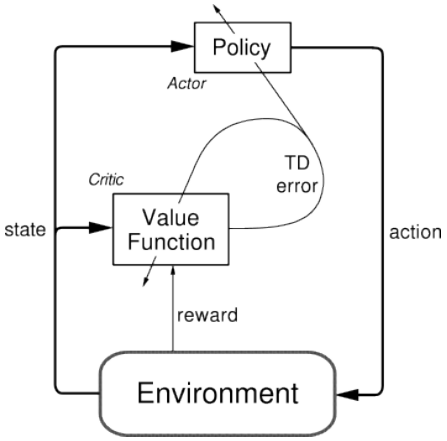
Prováděné experimenty srovnávají úspěšnost naší implementace s algoritmy z knihovny stable-baselines-3[2] která obsahuje řadu různých implementací algoritmů posilovaného učení. Během experimentů se zkoumá vliv jednotlivých parametrů na celkovou úspěšnost agenta, jako je architektura neuronových sítí, nastavení aktivačních funkcí a obecně různé nastavení parametrů specifických pro typ implementované metody.

V tomto projektu se zabýváme implementací TD3 algoritmu (viz 2), který vychází z jiného Actor-Critic algoritmu Deep Deterministic Policy Gradient Agents (DDPG), do kterého přináší několik změn aby nedocházelo k přeceňování value funkce.

2. Actor-critic a algoritmus TD3

Metody actor-critic jsou specifické tím, že mají oddělenou strukturu pro policy a value funkci. Policy síť se nazývá actor a síť aproximující value funkci se nazývá critic. Úkolem sítě actor je vybírat akce, které mu poté critic kritizuje, čili aproximuje očekávanou value funkci. Průběh učení agenta je dobře vidět na diagramu v obrázku 1.

Twin-delayed deep deterministic policy gradient (TD3) je jedním z actor-critic algoritmů. Jedná se o



Obrázek 1. Diagram znázorňující průběh učení agenta používajícího actor-critic metodu.

online a off-policy učící metodu, založenou na principu actor-critic. TD3 agent hledá takovou optimální policy, která maximalizuje očekávanou kumulovanou odměnu. Agent používá dvě critic sítě, kterými se učí dvě value funkce a pro aktualizaci policy používá minimální odhad value funkce. Policy se updatuje méně často než Q-funkce a během její aktualizace se přidává šum k vybrané akci, tudíž se na základě policy budou s menší pravděpodobností vybírat akce s vysokým odhadem value. [1]

3. Popis řešení

Projekt je implementován v jazyce Python3 s použitím knihoven `Gym`, `MuJoCo` [2], `stable_baselines3` a několika dalších podpůrných nástrojů.

3.1 Implementace TD3

Agenta lze spustit ve dvou módech: trénovacím a evaluačním. Při evaluačním módu se načte již existující a natrénovaný model a provede se několik běhů v daném

[1] https://www.gymlibrary.ml/

[2] https://stable-baselines3.readthedocs.io/en/master/index.html

prostředí. Z běhů je vypočtena průměrná odměna, která je vytištěna na standardní výstup. Součástí evaluace je taky vykreslování chování agenta do okna a vytvoření videa.

Zajímavější je trénování. Po nastavení všech parametrů uživatelem je vytvořeno celkem šest neuronových sítí. Dvě pro síť actor (actor a actor-target) a dvě pro každou ze dvou sítí critic (critic1,2 a critic1,2-target). Také se provede naplnění paměti agenta náhodnými akcemi v náhodných stavech prostředí s příslušnými odměnami. V tuto chvíli můžeme začít samotné trénování. To trvá daný počet kroků a v každém z nich provedeme akci vybranou sítí actor, ke které je přidán šum.

Také provedeme optimalizační akci, ve které se náhodně vybere několik vzpomínek z paměti. Vypočítáme hodnotu value funkce pro obě critic-target sítě vybereme z nich menší hodnotu a s pomocí chybové funkce MSE spočítáme loss sítí critic.

Optimalizace sítě actor se provádí pouze co N kroků. Zde nám stačí pro dané stavy z paměti vygenerovat akce a jako chybu vzít negativní průměr hodnot daných sítí critic. Na konci trénovacího kroku se provede aktualizace target sítí tak, že se provede lineární kombinace parametrů s non-target sítí s využitím parametru $\tau \in < 0,1 >$.

4. Experimenty

Pro vyhodnocení výsledného implementovaného algoritmu jsme se rozhodli provést řadu experimentů. V první sadě experimentů jsme prováděli porovnání našeho TD3 algoritmu s existující implementací z knihovny stable-baselines-3[3]. Využili jsme méně komplexních prostředí fyzikálního simulátoru MuJoCo[4]. Druhá sada experimentů se zabývá vlivem nastavení hyperparametrů na agentovo učení v prostředí Hopper.

4.1 Srovnání se stable-baselines-3

Pro porovnání našeho algoritmu s existující implementací z knihovny stable-baselines-3 jsme si vybrali tři prostředí z MuJoCo: `InvertedDoublePendulum-v2`, `Reacher-v2` a `HalfCheetah-v2`. Pro vyhodnocení jsme použili kritérium průměrné odměny z posledních 256 epizod. Pro každé prostředí byly nastaveny stejné hyperparametry u obou implementací. Výsledky učení pro 50000 časových kroků v každém prostředí lze vidět na obrázku 2.

[3] https://stable-baselines3.readthedocs.io/en/master/modules/td3.html

[4] https://mujoco.org/

Nejlepší natrénované modely obou algoritmů byly evaluovány na 1000 epizodách. Výsledky evaluace jsou v tabulce 1, která ukazuje průměrnou hodnotu odměň přes všechny provedené epizody.

	Inverted Double Pendulum	Reacher	HalfCheetah
Our-TD3	9349.20	−6.08	2856.33
SB3-TD3	4140.74	−4.84	2493.56

Tabulka 1. Průměrné hodnoty odměn při evaluaci algoritmů na 1000 epizodách.

Výsledky tohoto experimentu naplňují očekávání. Náš algoritmus se učí řešit prostředí s velice podobnými výsledky jako existující stable-baseline-3 implementace. Pro úplné porovnání by bylo třeba provést více experimentů s různým nastavením hyperparametrů, výše uvedené porovnání nám ale přišlo pro účely tohoto projektu dostatečné.

4.2 Nastavení hyperparametrů

Druhá sada experimentů se snaží zjistit závislost úspěšnosti agenta na nastavení hyperparametrů. Všechny experimenty budeme provádět v prostředí `Hopper-v2` a budeme sledovat průměr odměn z posledních 256 kroků a chybové funkce obou sítí.

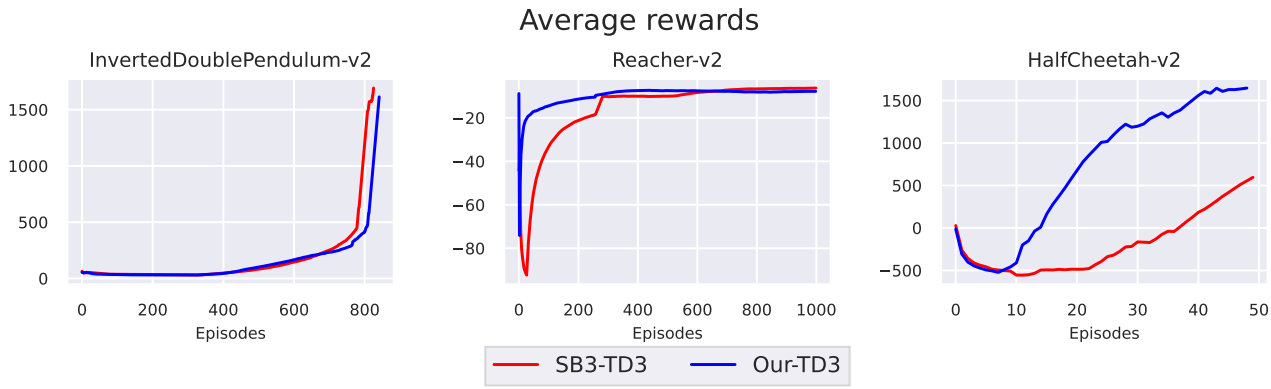
Budeme sledovat efekt discount factoru γ, kombinačního parametru τ, různé architektury sítí a různá nastavení frekvence učení sítě actor. Tyto všechny výsledky budeme porovnávat s výchozím nastavením parametrů, které je následující:

- $gamma = 0.99$
- $\tau = 0.005$
- $net = \{400, 300\}$
- $delay = 2$

Budeme vždy trénovat na 200000 kroků a budeme testovat následující možnosti parametrů:

- $gamma \in \{0.0, 0.5, 0.9, 1.0\}$
- $\tau \in \{0.0, 0.05, 0.1, 0.4, 0.8, 1.0\}$
- $net \in \{\{64, 64, 64\}, \{512, 256, 128\}\}$ a různé velikosti sítí actor a critic
- $delay \in \{1, 4, 10\}$

Výsledky experimentů můžeme vidět na obrázcích na konci tohoto dokumentu počínaje obrázkem 3. Můžeme si všimnout, že i při dlouhém tréningu nenastalo zlepšení, ale agent se nejlépe choval už po méně než 5000 epizodách. Parametr γ je nejlepší nechat co nejvyšší, velice blízký hodnotě 1.0. Zdá se, že architektury jednotlivých sítí na výsledek neměly



Obrázek 2. Srovnání průběhu učení agenta pro naši implementaci a pro implementaci ze stable-baselines-3.

nějaký značný vliv a všechny experimenty se chovaly velice podobně.

Frekvence provádění optimalizace sítě actor je lepší ponechat na nižších hodnotách, experimentálně dopadl nejlépe výchozí parametr s hodnotou 2. U parametru τ se jeví nejlepší ponechat ho velice blízko nule, zajímavého výsledku si můžeme všimnou na obrázku 18, kde se dlouho nic nedělo a ke konci učení odměny explodovaly. Při vyšších nastaveních se agent neučí vůbec.

5. Závěr

Hlavní náplní této práce bylo naprogramovat vlastní implementaci actor-critic algoritmu. Vytvořený algoritmus svými specifiky odpovídá metodě TD3 a je schopný řešit úlohy z prostředí MuJoCo. Učení agenta používajícího náš algoritmus dosahuje dobrých výsledků a je velice podobné učení agenta používajícího existující implementaci TD3 ze stable-baselines-3.

Dále jsme v této práci ukázali, že je možné v naší implementaci nastavovat různé hyperparametry jednotlivých komponent algoritmu, což jsme následně demonstrovali na několika experimentech, které ukazují vliv hyperparametrů na učení agenta.

Porovnání průběhu učení agenta s existující implementací proběhlo na třech MuJoCo prostředích a ve všech případech lze pozorovat velice podobný tvar křivky průměrných odměn. Lze tedy předpokládat, že náš algoritmus dosahuje podobné úrovně kvality jako již existující implementace.

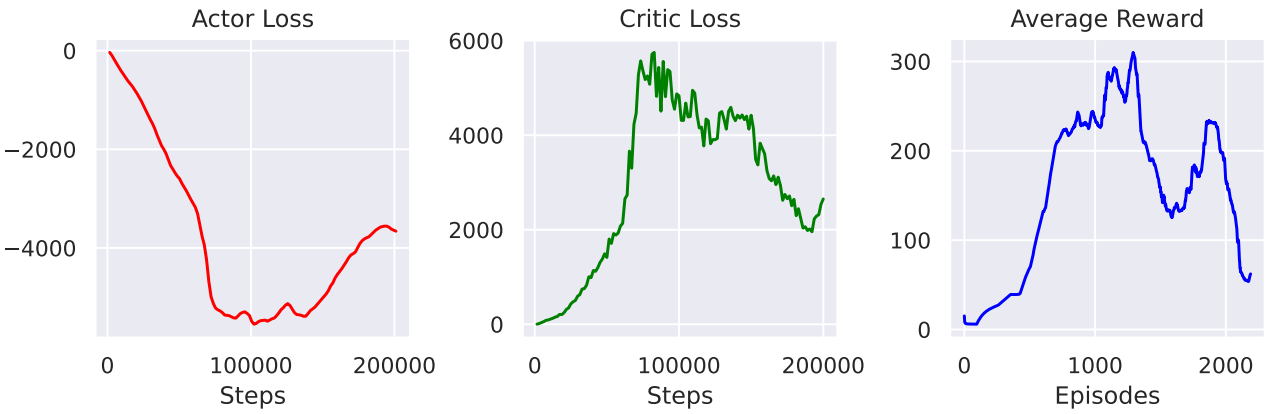
Implementace našeho algoritmu umožňuje experimentování ať už na nejnižší úrovni se samotnou metodou TD3, tak na vyšší úrovni s trénováním agenta s různými hyperparametry na libovolných prostředích knihovny OpenAI Gym (tedy nejsou limitovány pouze na MuJoCo). Výsledky agentova učení lze porovnávat s jinými existujícími implementacemi.

Literatura

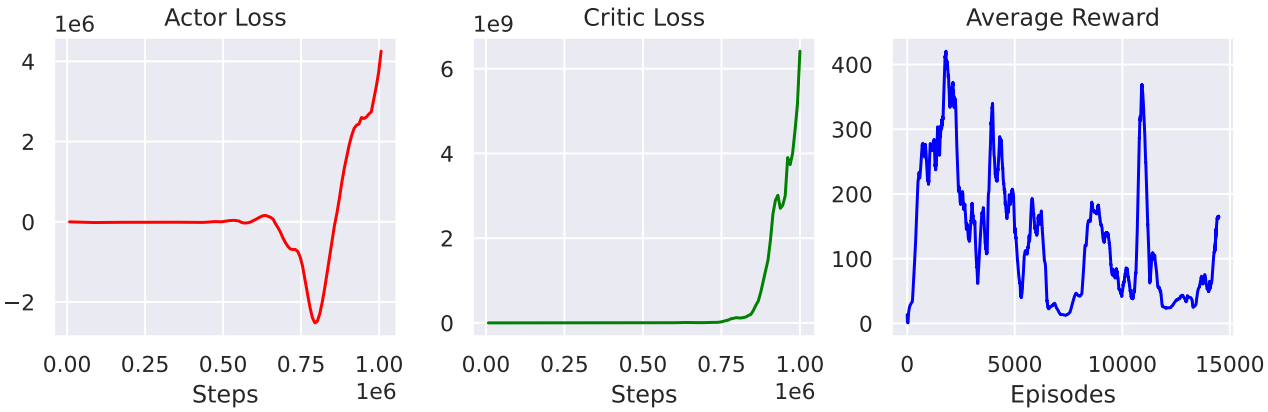
[1] THE MATHWORKS, I. *Twin-Delayed Deep Deterministic Policy Gradient Agents* [documentation (english)]. Březen 2022. Dostupné z: https://www.mathworks.com/help/reinforcement-learning/ug/td3-agents.html.

[2] TODOROV, E., EREZ, T. a TASSA, Y. MuJoCo: A physics engine for model-based control. In: IEEE. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, s. 5026–5033.

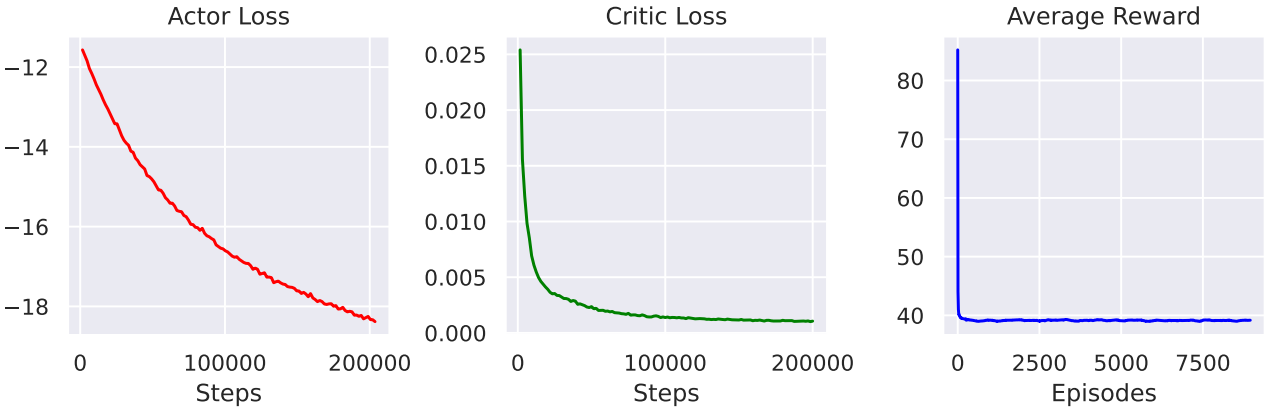
[3] TUNYASUVUNAKOOL, S., MULDAL, A., DORON, Y., LIU, S., BOHEZ, S. et al. Dm_control: Software and tasks for continuous control. *Software Impacts*. 2020, roč. 6, s. 100022. Dostupné z: https://www.sciencedirect.com/science/article/pii/S2665963820300099. ISSN 2665-9638.



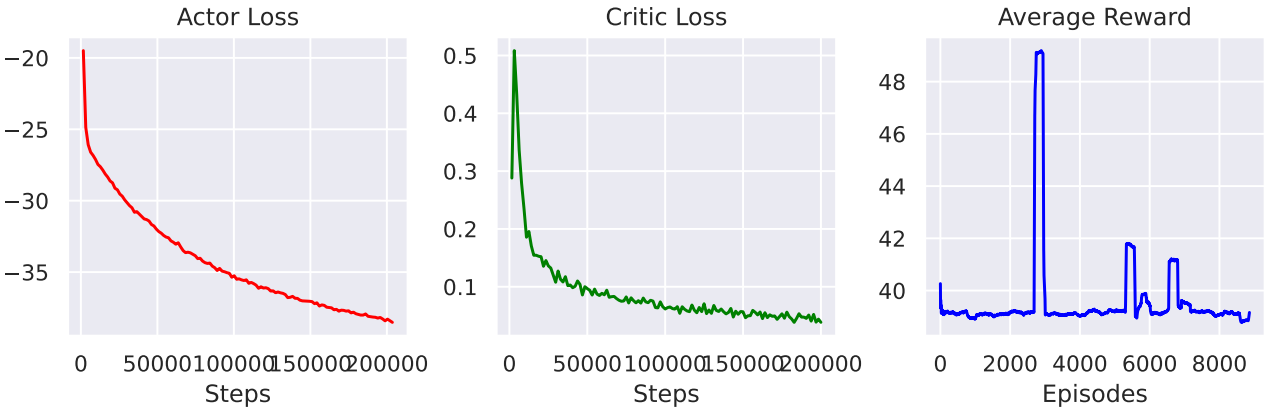
Obrázek 3. Výchozí nastavení parametrů.



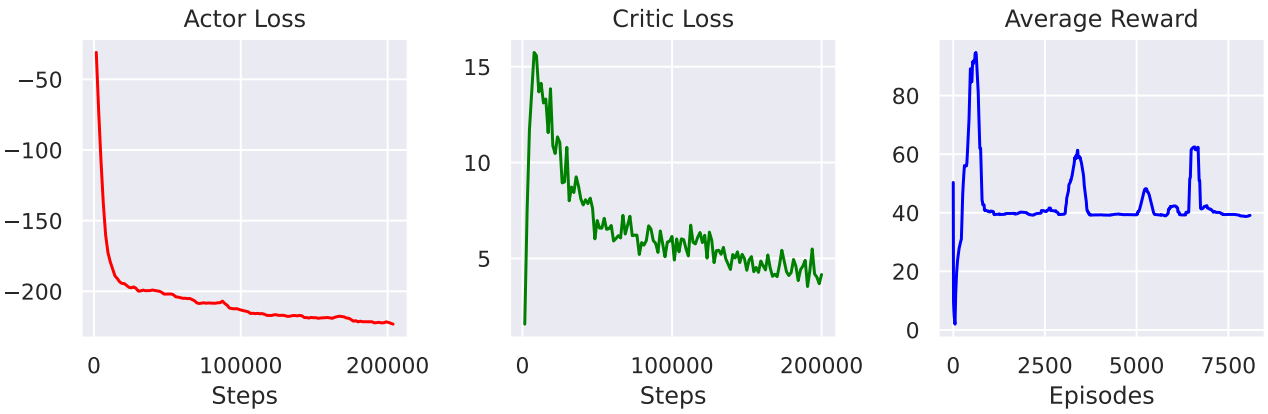
Obrázek 4. Výchozí parametry pro 1M kroků.



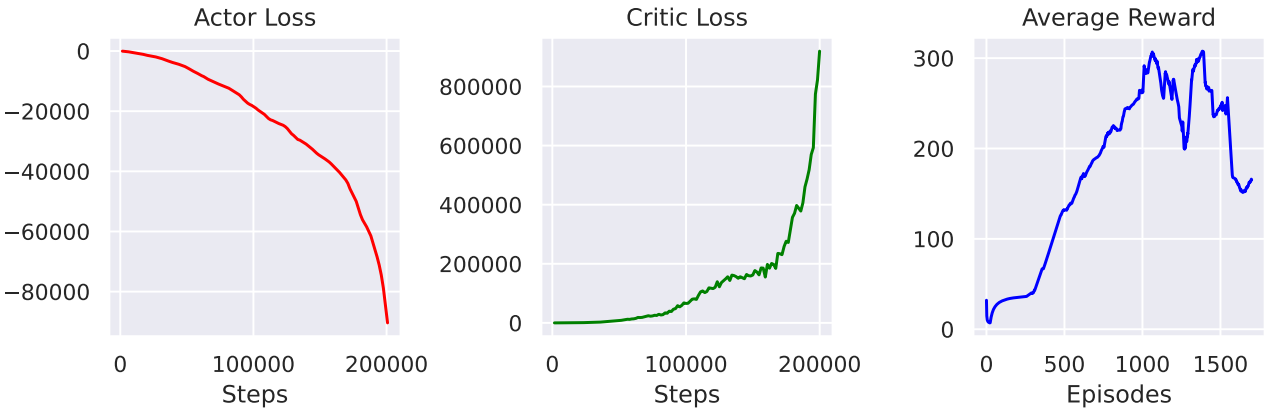
Obrázek 5. $\gamma = 0.0$.



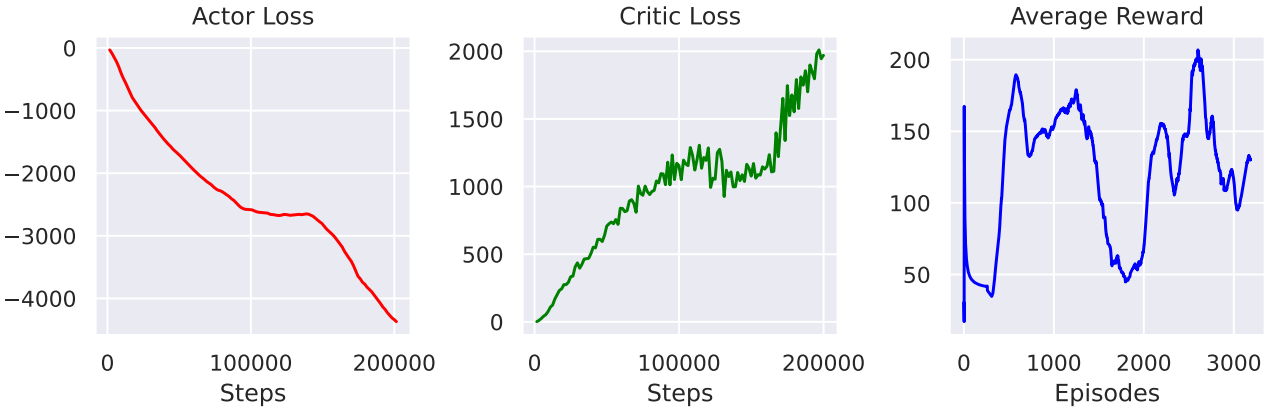
Obrázek 6. $\gamma = 0.5$.



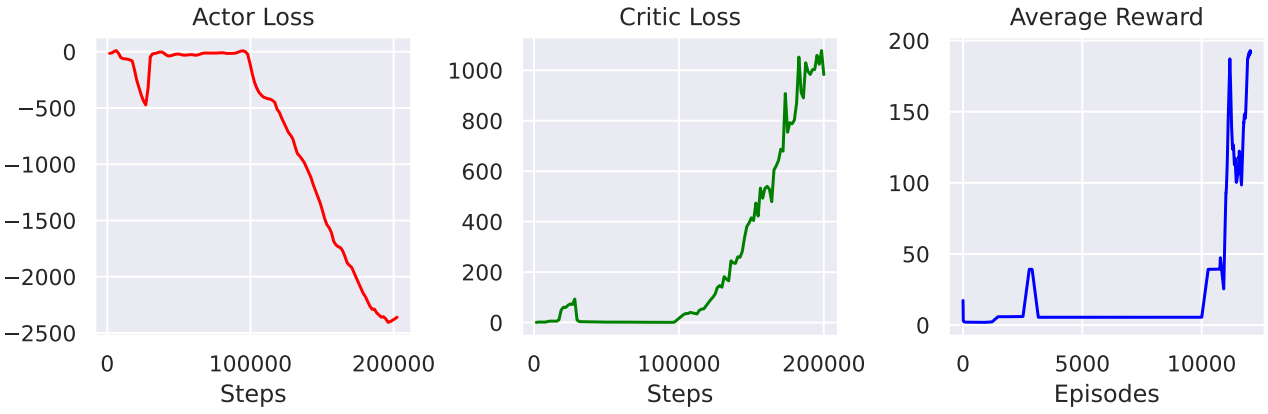
Obrázek 7. $\gamma = 0.9$.



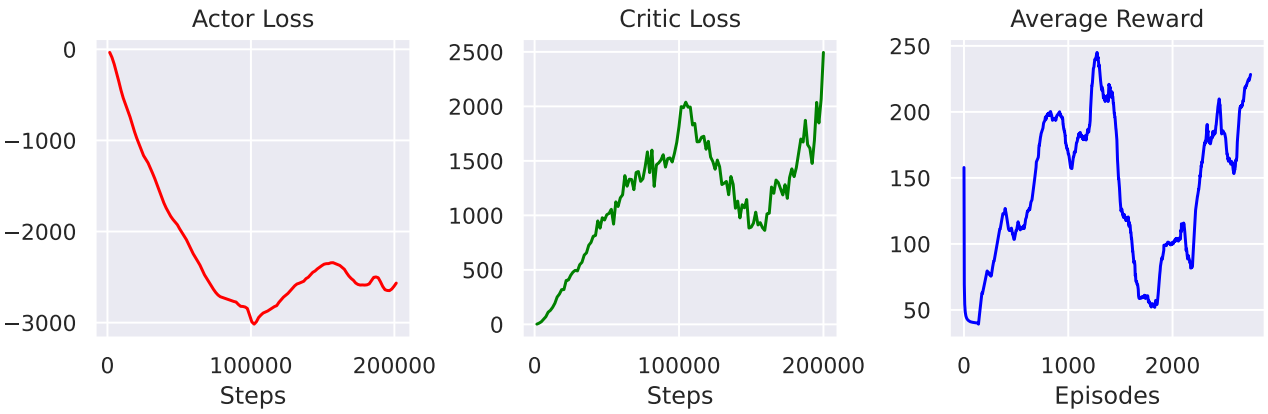
Obrázek 8. $\gamma = 1.0$.



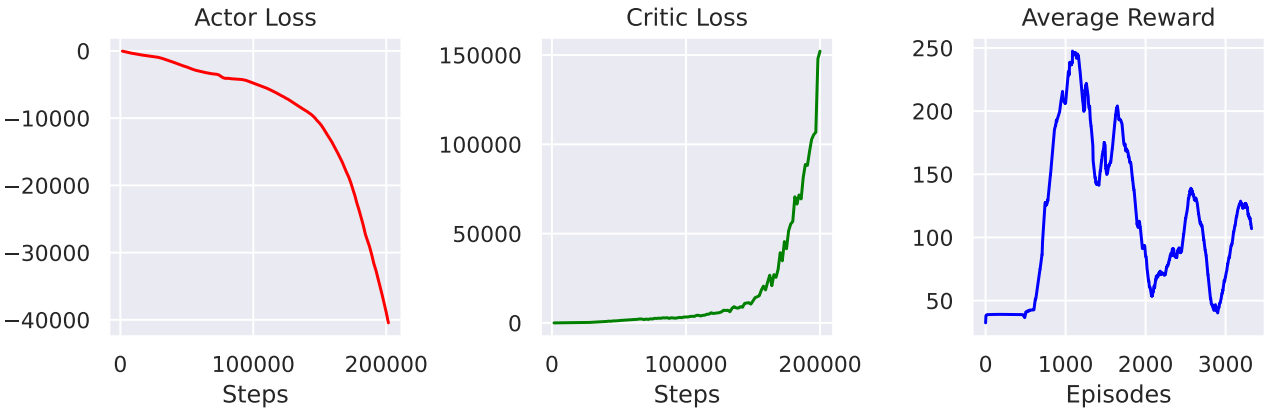
Obrázek 9. $nets = \{64, 64, 64\}$.



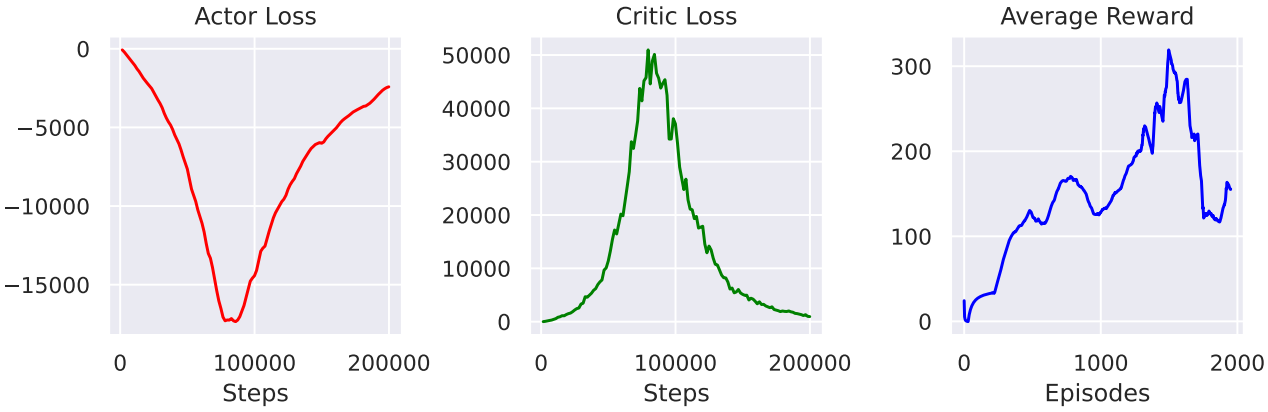
Obrázek 10. $nets = \{512, 256, 128\}$.



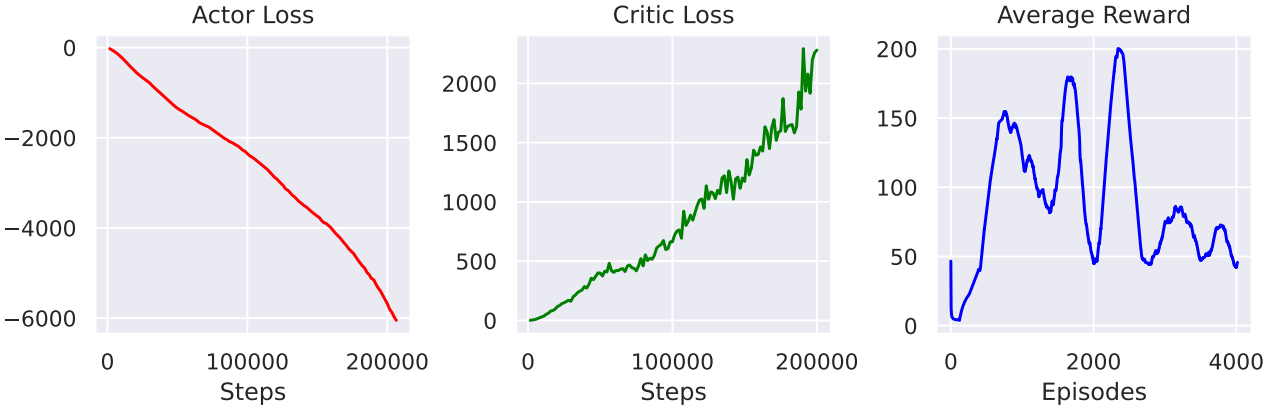
Obrázek 11. $actor = \{256, 256\}, critic = \{256, 256, 256\}$.



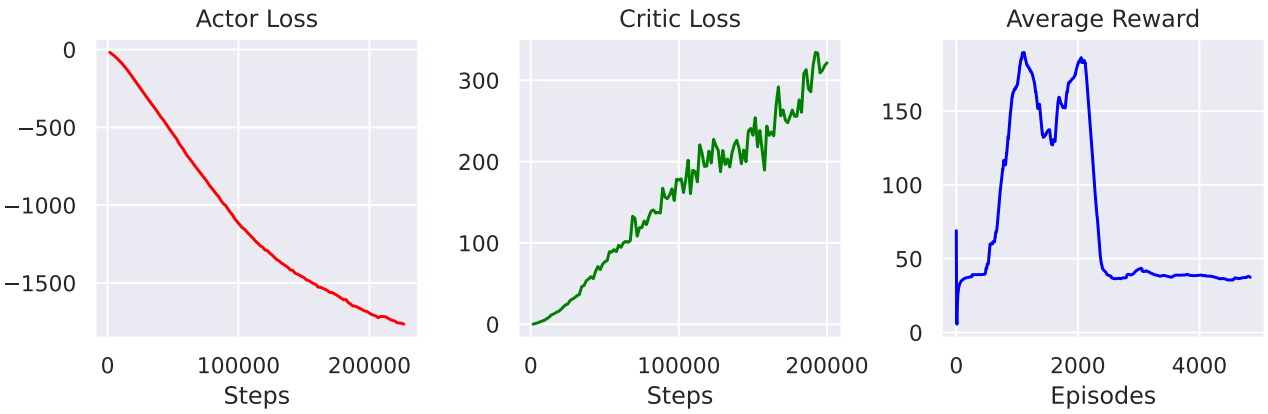
Obrázek 12. $actor = \{256, 256, 256\}, critic = \{256, 256\}$.



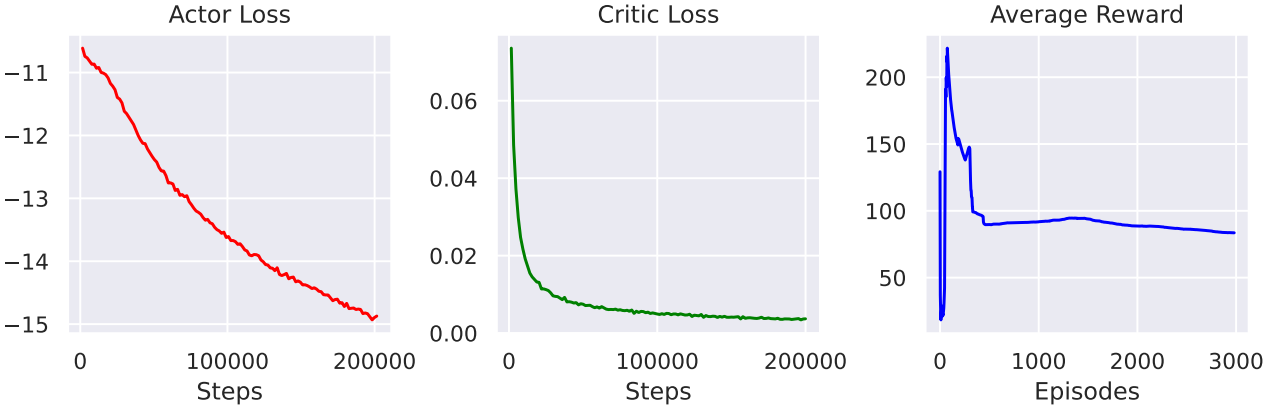
Obrázek 13. $delay = 1$.



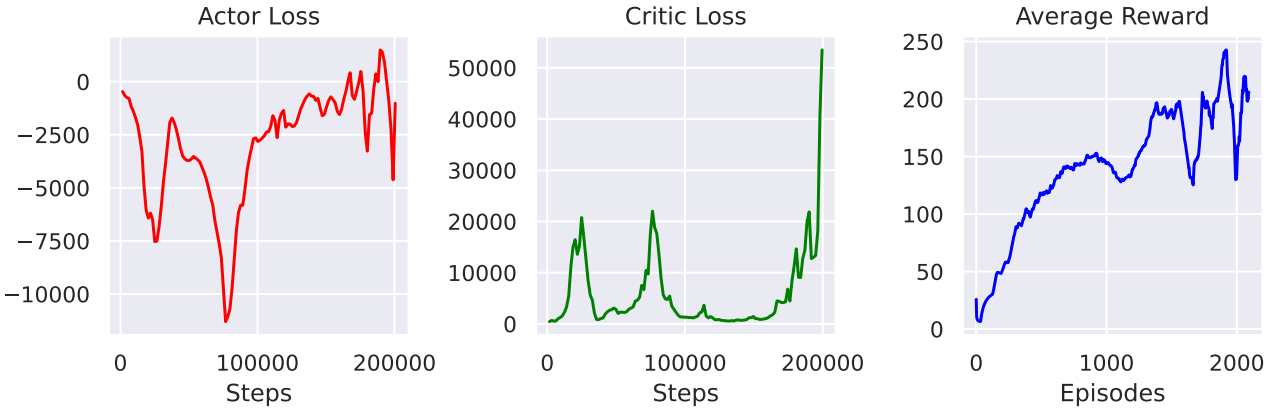
Obrázek 14. $delay = 4$.



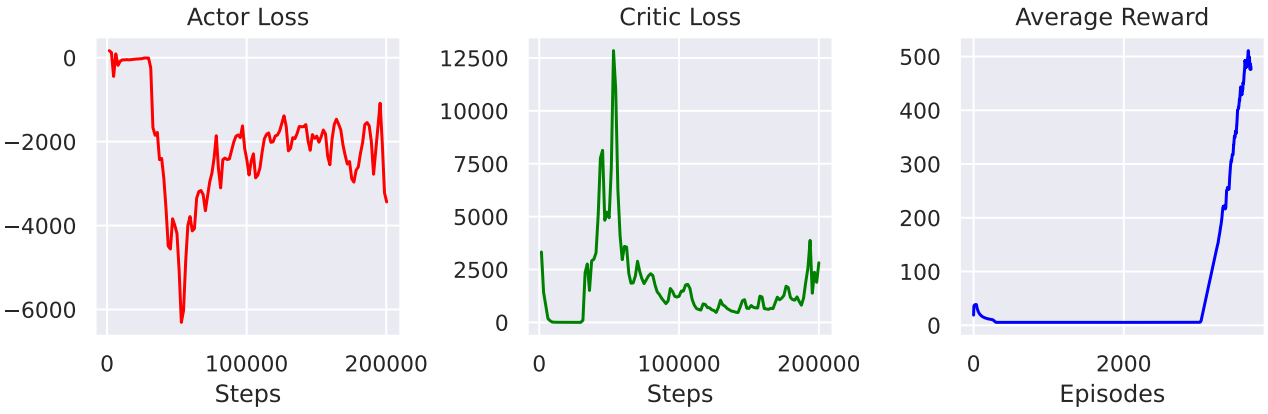
Obrázek 15. $delay = 10$.



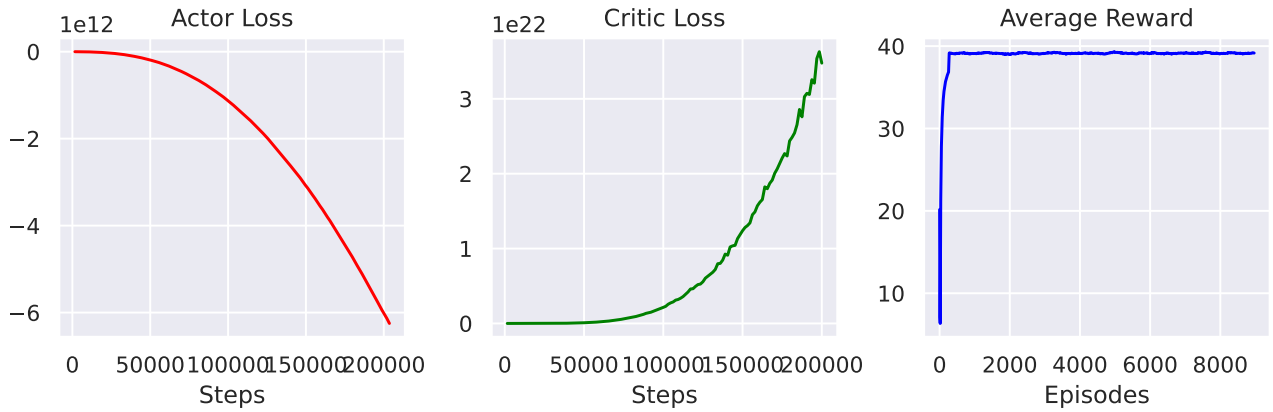
Obrázek 16. $\tau = 0.0$.



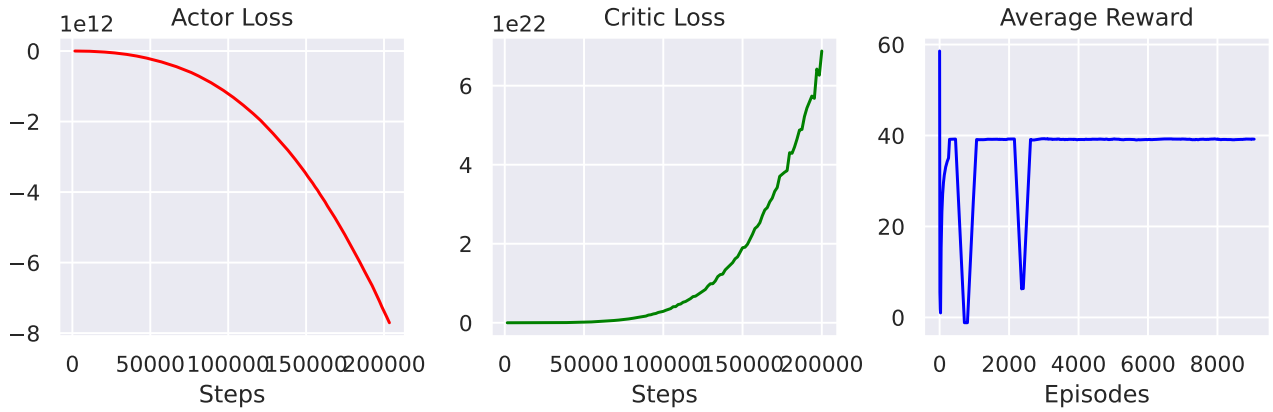
Obrázek 17. $\tau = 0.05$.



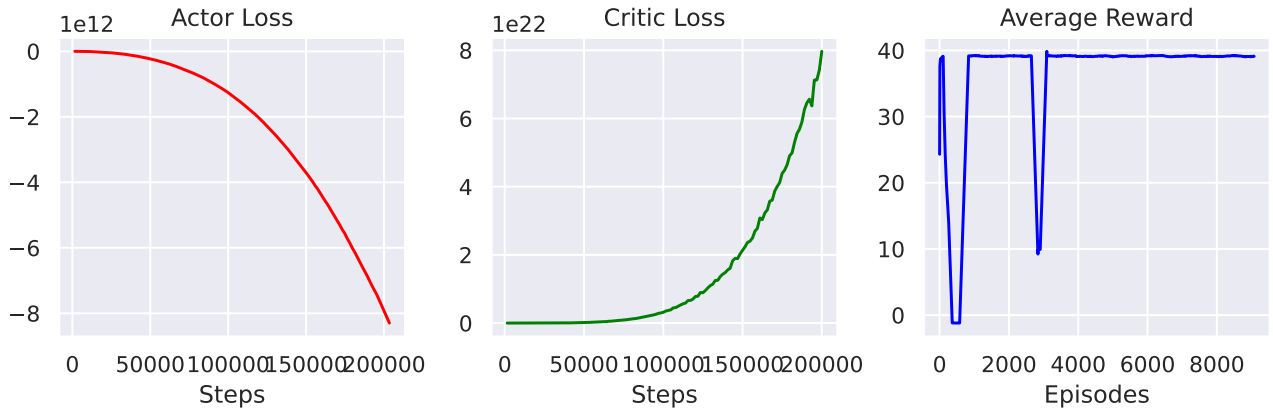
Obrázek 18. $\tau = 0.1$.



Obrázek 19. $\tau = 0.4$.



Obrázek 20. $\tau = 0.8$.



Obrázek 21. $\tau = 1.0$.