Deep Reinforcement Learning with the MuJoCo Physics Simulator

Curtis Brinker
cjbzfd@mst.edu

Tanner May
tmay@mst.edu

Abstract—**Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.**

I. INTRODUCTION

Recent developments in machine learning have made huge strides in approaching problems that were previously unsolvable with traditional programming methods. In general, machine learning techniques are grouped into three categories: supervised learning, unsupervised learning, and reinforcement learning (RL) [1].

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

- First itemtext
- Second itemtext
- Last itemtext
- First itemtext

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi

sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

II. TESTING ENVIRONMENT

Each of the models were trained using Open AI's Gym library and DeepMind's physics simulator MuJoCo. The Gym toolkit implements numerous environments for testing RL, such as the MuJoCo physics simulator. MuJoCo stands for **Mu**lti-**Jo**int dynamics with **Co**ntact and was made for applications that require fast, accurate physics simulations such as robotics and machine learning [2]. Using MuJoCo, machine learning algorithms can learn how to control the movement of robotic models, such as a snake, ant, or humanoid.

Each of Gym's environments adhere to a strict structure that makes it easy to switch testing environments. One of the most important parts to keep consistent is the values returned after every step of training [3]:

- Observation: Represents the current state of the environment, as the agent sees it. In MuJoCo's case, the environment is fully observable and consists of the model's position, velocity, and various forces [4]
- Reward: The reward result of the previous action. The reward for the MuJoCo environment is quite simple: if the simulation is still running, give a reward.
- Done: A flag denoting if the environment needs to be reset. Each of the models in MuJoCo define a safe height range that the model is allowed to exist within; if the model leaves that range, the done flag is raised and the model is reset back to the start.
- Info: Some information useful for debugging.

¡Some sort of final words and transition into the background section.¿

III. BACKGROUND

The study of reinforcement learning is about training an *agent* to interact with an *environment*. An action by an agent can influence the environment, which may later affect the actions of the agent. For this reason, reinforcement learning algorithms must be able handle a changing environment that is causally influenced by the agent itself.

To formalize the interdependency between the environment and the agent is done with with with the *agent-environment interaction loop*. In which, at time t the environment is fully

described by its *state*, s_t. Then, an agent makes an *observation* of the environment, o_t, where $o_t \subseteq s_t$.[1] The agent responds to this observation with an *action* a_t and is given a *reward*, r_t. After the action is taken, the state of the environment changes with the new state denoted as s_{t+1}.

An environment state can be represented by a tensor of values describing individual aspects of the environment. The domain of possible observations of the agent is referred to as the *observation space*, which can be either continuous or discrete. Similarly, the domain of all actions that an agent can take is called the *action space* which can also be continuous or discrete. An action, a_t, is selected from the action space by agent's *policy*. The policy can select actions either deterministicly or stochastically. Typically, deterministic policies are denoted with $a_t = \mu(s_t)$ and stochastic policies are denoted as $a_t \sim \pi(\cdot|s_t)$. [5]

The reward given for the transition from s_t to s_{t+1} by action a_t is given by the reward function, where $r_t = R(s_t, a_t, s_{t+1})$. Though, readers should be aware that some literature refers to this value as r_{t+1}. Typically, this reward function is specified by a programmer in a way that rewards a specific goal. With this in mind, the process of training an agent can be formulated as maximizing the cumulative reward, called the *return*. To calculate the return, the *trajectory* is used which contains the sequence of states and actions taken by the agent. The trajectory is given by:

$$\tau = (s_0, a_0, s_1, a_1, ...)$$

Therefore, as the return is defined in terms of cumulative reward, the trajectory can be used to calculate the rewards as the agent interacts with the environment. Then, in its most simple form, the return is the summation of all rewards. However this introduces two problems: First the return for an action t is defined in terms of actions taken before it. Intuitively, we would to only consider rewards that will happen as a result of the current action. Second, The summation of all rewards weights all rewards equally, however, this means that very long term rewards may eventually dominate the summation. To encourage shorter term rewards we can introduce a discount factor, γ, that discounts the value of future rewards.

Commonly, the reward function R is co-opted to take a trajectory and produce the rewards at each timestep t. With these considerations in mind, the return can be defined as:

$$R_t(\tau) = \sum_{t'=t}^{T} \gamma^{t'} r_{t'}, \ \ \gamma \in (0,1)$$

Where T is the length of the trajectory and t is the current time for which the return is being calculated. If T is finite, then the return is said to be a *finite-horizon return*. Similarly, if T is infinite, then the return is said to an *infinite-horizon return*. The inclusion of γ makes this return a discounted return. This places more focus on near-term rewards and also helps with

---

[1]While the observation and the state are not necessarily equal, reinforcement learning literature frequently refers to an observation as the state itself.

convergence in infinite-horizon returns. Using t' makes the return only consider future rewards, which [6] calls the *reward-to-go* return.

However, an agent cannot act to directly maximize its return as it dependent on future actions and states. Instead, an approximation must be used to calculate the expected return, $\mathcal{J}(\pi)$. This this done with *value functions*. Formally, $\mathcal{J}(\pi)$ is defined as:

$$\mathcal{J}(\pi) = \mathop{\mathbb{E}}_{\tau \sim \pi} [R(\tau)]$$

The *state value function* for policy π, denoted as V^π, gives the expected value for policy π given state s. Alternatively, the *state-action value function* for policy π, denoted as Q^π, which can be used estimate the expected value for policy π in state s and immediately taking action a. Formally, these are defined as:

$$V^\pi(s) = \mathop{\mathbb{E}}_{\tau \sim \pi} [R(\tau) \,|\, s_0 = s]$$
$$Q^\pi(s,a) = \mathop{\mathbb{E}}_{\tau \sim \pi} [R(\tau) \,|\, s_0 = s, a_0 = a]$$

However, these functions are still dependent on the return. To avoid this, we can use the Bellman Equations, which defines both V^π and Q^π in terms of expected values of future states. [5], [7], [8] These equations are given as:

$$V^\pi(s_t) = \mathop{\mathbb{E}}_{a_t \sim \pi} [R(s_t, a_t, s_{t+1}) + \gamma V^\pi(s_{t+1})]$$
$$Q^\pi(s_t, a_t) = \mathop{\mathbb{E}}_{a_{t+1} \sim \pi} [R(s_t, a_t, s_{t+1}) + \gamma Q^\pi(s_{t+1}, a_{t+1})]$$

For the agent to maximize $\mathcal{J}(\pi_\theta)$, the agent must learn to improve the policy π_θ, where θ are the parameters for the policy and is normally updated by gradient ascent. Normally this involves using V^π or Q^π to better approximate the optimal value functions. Commonly, these approximation are done via deep neural networks. The exact details of how the policy and value function(s) are updated is dependent on the reinforcement learning algorithm used. Following is a brief discussion on different types of reinforcement learning algorithms followed by discussions on details of specific algorithms studied in this paper.

A. The REINFORCE Algorithm

B. Advantage Actor Critic (A2C)

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

C. Deep Deterministic Policy Gradient (DDPG)

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

D. Soft Actor Critic (SAC)

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

IV. METHODOLOGY

Each of the algorithms were implemented on Ubuntu 20.04 with Python 3.8 using the PyTorch library. We used PyTorch because we had previous experience with it and liked how flexible it is compared to other options. To train the models, we used the previously mentioned Gym library. To verify that the implementation of each of the algorithms worked correctly, we trained them using the Pendulum-v1 environment whose goal is to simply swing a pendulum so it stays upright. For training towards our goal, we used the MuJoCo environment with the Ant-v3 and Humanoid-v3 models. The ant model was chosen because it is relatively simple (four legs, two joints each) and will still teach the algorithm to walk in a three dimensional environment. The humanoid model was chosen because it is significantly more complex (two legs and a total of seventeen joints) and should showcase the learning power of the algorithms. Since the reward is given for every step that the simulation continues, a stopping condition had to be defined. For the Ant model, we defined the safe height range to be from 0.3 to 2.0; for the humanoid, we used the default of 1.0 to 2.0. If the model left its respective range, the environment would raise the 'done' flag and the model would be reset.

Since the models have vastly different complexities the amount of training steps varied between them, but not between RL algorithms. Each of the algorithms was able to train with the ant model for 500,000 steps; after 500,000 steps, training was stopped and the most recent saved model was used. We chose 500,000 steps since it seemed to produce good results (a model that moved a non-trivial distance) for all of the algorithms tested. Training with the humanoid model was conducted for 1,250,000 steps. But, since training took so long, we only trained the SAC algorithm because it produced the most interesting results during the ant training phase.

The hyperparameters for each algorithm were as follows:

- A2C: Learning rate of $3e^{-4}$ and $\gamma = 0.99$.
- DDPG: Update freq. of 64 steps, update threshold of 4096 steps, batch size of 128, learning rate of $1e^{-3}$, $\gamma = 0.99$, $\tau = 0.995$, and the noise distribution was Normal with a standard deviation of 0.1.
- SAC: Update freq. of 64 steps, 64 updates per update step, an update threshold of 4096 steps, batch size of 128, $\alpha = 0.5$, learning rate of $5e^{-4}$, $\gamma = 0.99$, $\tau = 0.995$, and an $alpha$ decay of 1.

Each of those parameters were chosen because they resulted in the best performing model.

Since our goal is to teach Deep RL algorithms how to walk, we defined the networks for each of the algorithms to have two layers, each with 256 neurons. Originally we tried 128 neurons each but algorithms using the 256 neuron configuration had better performance. More layers and neurons can, of course, be used but would have greatly increased the computational cost of training.

V. RESULTS

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

A. CartPole-v0

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.
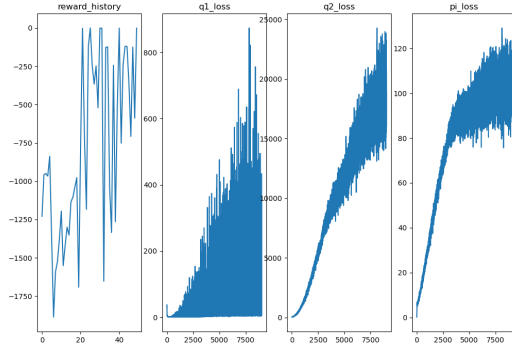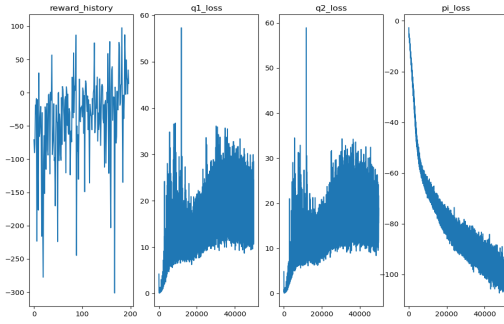
Fig. 1: SAC in the Pendulum-v1 Environment

Fig. 2: SAC in the Ant-v3 environment

Fig. 3: DDPG in the Ant-v3 environment

B. Pendulum-v1

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

C. Ant-v3

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi
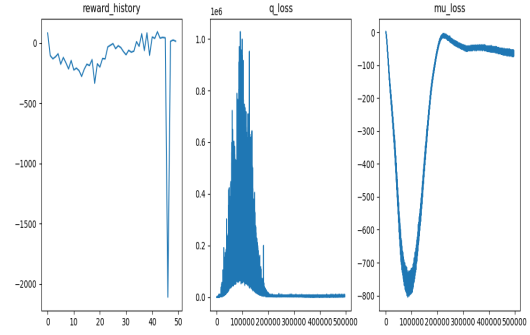
sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

D. Humanoid-v3

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

VI. Future Work and Conclusions

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla

tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

REFERENCES

[1] W. Qiang and Z. Zhongli, "Reinforcement learning model, algorithms and its application," in *2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)*, 2011, pp. 1143–1146.

[2] DeepMind, "Mujoco documentation." [Online]. Available: https://mujoco.readthedocs.io/en/latest/overview.html

[3] OpenAI, "Gym documentation." [Online]. Available: https://gym.openai.com/docs/

[4] ——. Gym library. v0.21.0. [Online]. Available: https://github.com/openai/gym

[5] (2018) Spinning up: Key concepts in rl.

[6] (2018) Spinning up: Intro to policy optimization.

[7] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[8] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.