Continuous Control using Deep Reinforcement Learning

Amulya Kallakuri
University of Southern California
kallakur@usc.edu

Aditi Bodhankar
University of Southern California
bodhanka@usc.edu

Nithyashree Manohar
University of Southern California
nithyash@usc.edu

Abstract

Deep Reinforcement Learning (RL) deals with implementing RL algorithms and policies on environments using agents, states, and actions. An agent is an observer, learner and decision-maker in the environment. A state is a representation of the environment that the agent is in, which includes all relevant information about the environment that the agent needs to know to make a decision. An action space is a set of all actions that the agent can take in the current state. In this project, we focus on action spaces that are continuous in nature with the set being of infinite dimension. We implement the Deep Deterministic Policy Gradient (DDPG) and the Proximal Policy Optimization (PPO) algorithm in two different environments.

***K*eywords** Reinforcement Learning · Environment · Agent · Action · State · Reward

1 Introduction

1.1 Problem Statement and Goal

The main focus of this project was creating and implementing the algorithms on two custom environments. The environments were created using OpenAI Gymnasium and the details are as follows:

- Environment 1: Shower Head
  - Goal: Build RL model to adjust temperature automatically to get it within the optimal range.
    Optimal range: 37º to 39º.
  - Actions: Turn knob between -π and π
  - States: Values between 0º to 100º
  - Reward functions: Rectangular, Parabolic, Triangular
  - Policy used: DDPG
- Environment 2: Asset Trading
  - Goal: Build the RL model to determine the appropriate amount to buy or sell based on the values in the observation space.
  - Actions: Buy, sell, hold, do nothing, continuous spectrum of amounts to buy/sell
  - Observation Space: open price, high price, low price, close price, daily volume
  - Reward function: Account balance multiplied by the number for time steps taken so far
  - Policy used: PPO

1.2 Background and Literature Review

There was a steep learning curve associated with this project. The following subjects were extensively researched upon:

- Literature Review of RLLib and OpenAI Gymnasium

- Implementation of environments in general
- Translation and differences between discrete and continuous action space environments

1.3 Timeline

- 2 Nov 2022: Literature Survey and study on RLLib and OpenAI Gym
- 7 Nov 2022: Gather data about environments in general, then on discrete action space environments
- 16 Nov 2022: Translate discrete action space to continuous action space environments
- 22 Nov 2022: Begin implementation of both custom environments
- 29 Nov 2022: Optimize and analyze results of both environments
- 3 Dec 2022: Brainstorm new reward functions and implement; compile results

1.4 Challenges

There were a number of challenges surrounding this project, mainly due to the extensive learning curve that we had chosen.

- The usage of RLLib versus OpenAI Gymnasium. We first tried to implement the custom environments using RLLib but given the time constraint, going with OpenAI Gym was the more logical choice. It was over a week of studying RLLib when we decided to go to Gym instead.
- Working on OpenAI Gym with existing environments. In order for us to implement the custom environments, we first had to learn and implement Gym with existing environments.
- Literature survey on creating custom environments with discrete action spaces. Before moving on to sparsely implemented continuous action space environments, the knowledge of implementing it on a relatively more researched discrete space was required.
- Creating the custom environment. We tried to create an environment from scratch and succeeded to a decent extent after which we had to compare this environment to other existing environments and refer to how they were implemented so that we could do the same.

1.5 Extensions

- Shower environment: The extension to this environment was mainly done in the additional reward functions. This took us relatively lesser time after we understood completely what the environment was doing and what we could do to optimize it. The initial rectangular reward function made sense in a theoretical way but it was more practical for us to consider a parabolic and triangular reward function. This paid off as we can see in the results below and led us to far better reward results.
- Asset Trading environment: The extension for this was done by translating the original discrete space to a continuous space by modifying the data that we used. We included two additional columns, one for the return indicating the percent change between the close price of the current day and the close price of the previous day; and the other for the cumulative product value which would tell us what our gains would be if we invested the $100 that we had and left it until the end of the data period. This way we could compare weekly returns and teach the model when is a good time to invest and when is not. This gave us a chance to analyze the implications of investing in the long run versus a shorter period of time, in this case, a week.

1.6 Inference and Exploration

Our inferences and exploration done during the course of this project are as follows:

- Shower environment: The practical effect of reward functions gave us significantly better results than a theoretical one. We explored the idea of using gradual decrement reward functions which led to our best results. We came to the idea of using newer and better reward functions fairly soon once we considered the practical aspect of what the environment was trying to do, instead of pursuing it in a purely theoretical manner. From our observations, the shower environment worked best with an inverse parabolic reward function which was as we had expected. Once we added noise and took into account the Ornstein-Uhlenbeck process, our agent was able to perform the exploration in a much better and random manner instead of converging at local minima, which is what is ideal for a reinforcement learning environment.

- Trading Assets environment: Once we had done an exploratory data analysis and taken into account our domain knowledge of the stock market, we were able to explore which factors could be considered and which could be dropped and added two key columns which led to our best results. Despite this, as can be seen in the graph below, certain agents worked well while others did poorly. Our model gave us significantly better results with the returns column and the cumulative product column along with the fractional reward function than it did with just the open price, close price, and a static reward function that multiplied only the prices with a fractional amount of how much we can buy and sell.

2 Implementation

For the implementation of our continuous control problems, we employ two different neural architectures for two different environments. For the physics-based environment, we implement the Deep Deterministic Policy Gradient (DDPG) algorithm using the Actor-Critic model and for the financial-trading environment, we look at a Multilayer Perceptron (MLP) based architecture. Before we move on to the architecture and the training, we first look at each environment in more detail in the following sub-sections.

2.1 Environment

In this project, we look at two different types of environments. One is a physics-based continuous control environment and the other is a asset trading based environment.

2.1.1 Shower Head Environment

For the physics-based environment, we formulated a shower environment, where the movement of the shower knob would result in different water temperatures. The observation space (also called the state space) comprises different temperatures ranging from 0ºto 100º and each state changes in an integral fashion. For a continuous action space, the shower knob moves within a range of $-\pi$ to $+\pi$ radians. The reward function is defined in such a way that for an optimal temperature range of 37º to 39º, the reward obtained for the agent is highest leading to positive reinforcement. When the temperature of the shower drops below 37º i.e., when the water gets too cold with every turn of the knob to the left, the reward drops from the maximum value. Similarly, when the temperature of the shower increases beyond 39º, the reward again drops to a smaller value. In both these cases, a negative reinforcement makes the agent take a state which is within the optimal temperature. In evaluating the policy and the Q-function, we implemented a Deep Deterministic Policy Gradient (DDPG) algorithm using an Actor-Critic Model.

2.1.2 Trading Environment

To visualize live results, we created an asset trading environment where we train our agent to be a profitable trader. When a human trader makes trades, the human looks at charts and technical indicators, combines prior knowledge with available information to make an informed decision. In the case of our agent, we want the agent to look at the stock data points before making or not making a trade. This forms the observation space which contains information such as open, low, high, close prices and daily volume over the last five days. Once the agent has learned about the environment, we want the agent to take an action and hence, the action space consists of three possibilities - buy, sell or do nothing. We also want to know the amount to buy or sell, this is a continuous spectrum of amounts to buy or sell. Once the agent has taken an action, we would like to evaluate the action which is done in the form of a reward function. Here, the reward function is the amount of balance in the account multiplied by the number of time steps taken by the agent so far.

2.2 Analytic Decisions

2.2.1 Shower Head Environment

The optimal range was chosen with the normal human body temperature in mind, making the environment a more realistic one. For the action space, the movement of the shower knob was considered to be continuous within the specified range. Additional noise in terms of the Ornstein-Uhlenbeck process was added to make the convergence of the reward function stable over extended episodes. An added gaussian noise also resulted in more randomized state space for the agent to learn from and result in an optimal policy. The actor and critic both are modeled as MLPs with two hidden layers, with the actor having ReLU activations through all the hidden layers and a tanh activation for the output. In the case of the critic, the action produced by the actor was added to the input of the second hidden layer to compute the Q-function. Overall, all the hidden layers and the output layer activation functions were formulated as ReLU. For the reward functions, we looked at three different kinds of functions - rectangular function where the reward dropped to -1 from +1 when the states were not in the optimal temperature range. Second, we looked at the parabolic reward function, where the maximum reward of +10 was within the optimal range of states, and as the temperature dropped below 37º or increased beyond 39º, the reward gradually decreased in a parabolic fashion. In the third case, we looked at a triangular reward function, where the reward reduces abruptly when the temperature is not within the optimal range.

2.2.2 Trading Environment

The reward function is the account balance multiplied by the number of time steps the agent has taken so far. This reward function has chosen to give the agent enough time to explore strategies before optimizing on a single strategy too deeply and too early in the process without exploration. Also to reward agents that maintain higher balances for longer periods of time. We used algorithms imported from the stable baselines3 library to train our reinforcement learning agent. Stable baselines3 provides a set of reliable implementations of reinforcement learning algorithms in PyTorch. The algorithm we used was the Proximal Policy Optimization which is a recent advancement in the field of Reinforcement Learning, and provides an improvement on Trust Region Policy Optimization (TRPO). As the agent is learning from the environment, PPO ensures that the updated policy is not too much different from the old policy to ensure low variance in training. The underlying algorithm used is the Multilayer Perceptron which has 2 layers and each layer consists of 64 neurons. We chose the MLP algorithm since it is suitable for time-series and tabular data.

2.3 Model Architecture and Implementation

2.3.1 Shower Head Environment

We first begin with a brief understanding of the DDPG algorithm and how it is different from the Deep Q-network used in most of the RL-based control problems. In the former, the actor maps each state into a specific action directly rather than to a probability distribution across the entire continuous action space. DDPG also uses target networks to increase the overall stability of the learning as they are time-delayed copies of the original network. Additionally, DDPG uses an experience replay to add experience to update the neural network parameters. As the trajectory of each roll-out changes, we save the experience tuple given by -

$$(state, action, reward, nextstate)$$

into a finite-sized 'Replay buffer'. To update the policy and Q-function networks, each mini-batch is sampled from the replay buffer.

To understand the Actor (policy) and the Critic (Q-function) networks in more detail, we start with the Bellman equation for the Q-function update -

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$$

Then we minimize the loss as -

$$Loss = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$$

And when it comes to the policy, our aim is to maximize the expected return, this is calculated as -

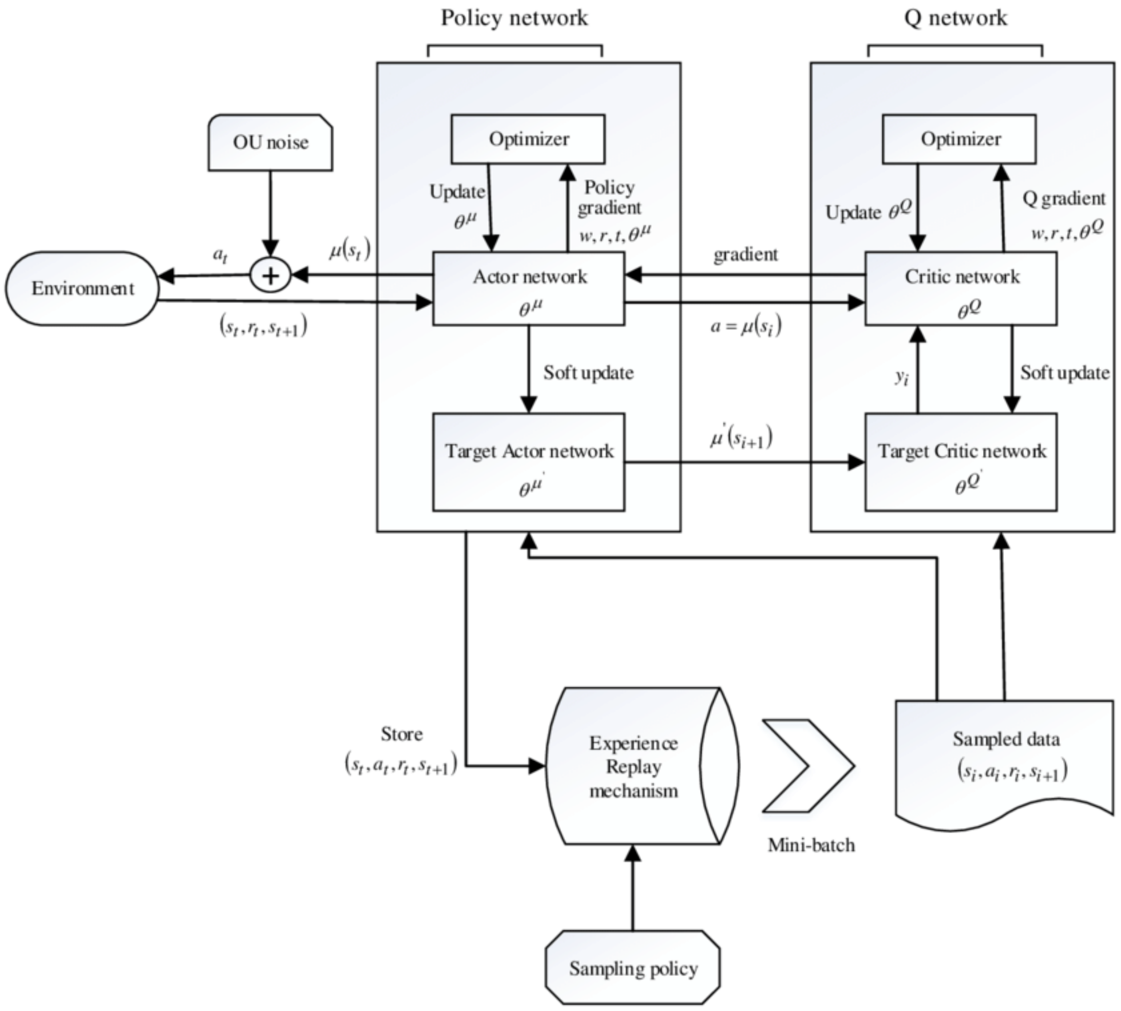
$$J(\theta) = E[Q(s,a)|_{s=s(t),a=\mu(s(t))}]$$

We minimize this policy loss since policy updates are happening in an off-policy way with each mini-batch of experience.

In the case of target networks, the parameter updates are similar to the 'soft updates' usually done in the gradient descent approach.

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$$
$$\text{where } \tau << 1$$

In the case of discrete action spaces, exploration is done by probabilistically selecting a random action. For continuous action spaces, exploration is done by adding noise (mostly gaussian) to the action itself. Further, the DDPG algorithm uses Ornstein-Uhlenbeck Process to add noise to the action output. The reason for this stochastic approach is that the noise generated correlates with the previous noise, to prevent the 'freezing' of the overall dynamics. The final DDPG algorithm with Actor-Critic Model can be visualized from figure:1



Policy network
Q network
OU noise
Optimizer
Update θ^{μ}
Policy gradient w, r, t, θ^{μ}
Optimizer
Update θ^{Q}
Q gradient w, r, t, θ^{Q}
a_t
$\mu(s_t)$
Environment
Actor network θ^{μ}
gradient
Critic network θ^{Q}
(s_t, r_t, s_{t+1})
$a = \mu(s_i)$
Soft update
y_i
Soft update
Target Actor network $\theta^{\mu'}$
$\mu'(s_{i+1})$
Target Critic network $\theta^{Q'}$
Store (s_t, a_t, r_t, s_{t+1})
Experience Replay mechanism
Mini-batch
Sampled data (s_i, a_i, r_i, s_{i+1})
Sampling policy

Figure 1: DDPG Architecture

2.3.2 Trading Environment

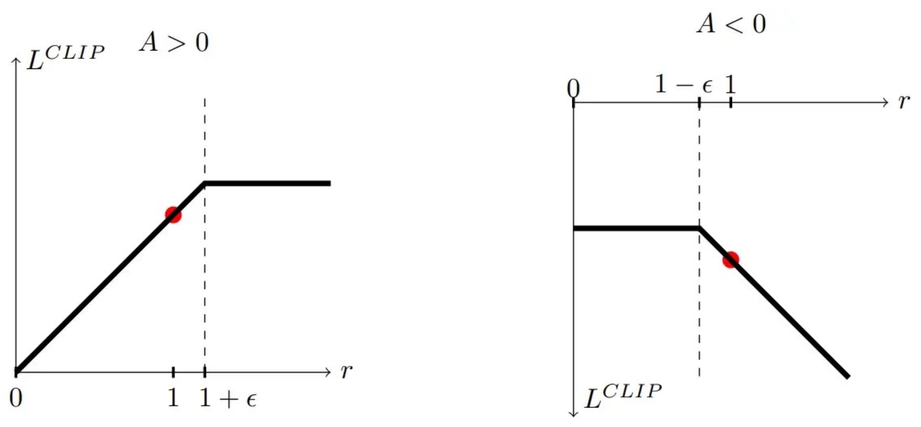
The algorithm used was the Proximal Policy Optimization which builds on Trust Region Policy Optimization. PPO has two variants - PPO penalty and PPO clip. In this project, we have used the PPO clip variant.

The PPO clip variant outperforms the penalty variant and it is simple to implement. The other variants change penalty over time, PPO clip has a range within which the policy can change. When the agent achieves advantages by updates outside the clipping range, they are not used when we update the policy. This provides the agent an incentive to stay relatively close to the existing policy.

$$\mathcal{L}^{CLIP}_{\pi_\theta}(\pi_{\theta_k}) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\left[\min\left(\rho_t(\pi_\theta,\pi_{\theta_k})A_t^{\pi_{\theta_k}}, \text{clip}(\rho_t(\pi_\theta,\pi_{\theta_k}),1-\epsilon,1+\epsilon)A_t^{\pi_{\theta_k}}\right)\right]\right]$$

The terms (1- ϵ)A and (1+ϵ)A do not depend on θ, yielding a gradient of 0. The samples outside the trusted region are tossed, eliminating overly large updates. In conclusion, we don't explicitly constrain the policy update itself, but simply eliminate advantages stemming from overly deviating policies. We use an optimizer like ADAM to perform the updates.

From the graphs below it can be seen that in this variant of PPO, the surrogate advantage is clipped. If the updated policy deviates from the original one by more than ϵ, the sample yields a gradient of 0. This mechanism avoids overly large updates of the policy, retaining it within a trusted region



There are six non-trivial cases with corresponding update behavior summarized in the table below.

Ratio	Advantage	Update signal	Clipped objective	Gradient $\neq 0$
$\frac{\pi_{\theta_k}(s_t,a_t)}{\pi_\theta(s_t,a_t)} \in [1-\epsilon, 1+\epsilon]$	$A_t > 0$	$\frac{\pi_{\theta_k}(s_t,a_t)}{\pi_\theta(s_t,a_t)} A_t$	No	Yes
$\frac{\pi_{\theta_k}(s_t,a_t)}{\pi_\theta(s_t,a_t)} \in [1-\epsilon, 1+\epsilon]$	$A_t < 0$	$\frac{\pi_{\theta_k}(s_t,a_t)}{\pi_\theta(s_t,a_t)} A_t$	No	Yes
$\frac{\pi_{\theta_k}(s_t,a_t)}{\pi_\theta(s_t,a_t)} < 1-\epsilon$	$A_t > 0$	$\frac{\pi_{\theta_k}(s_t,a_t)}{\pi_\theta(s_t,a_t)} A_t$	No	Yes
$\frac{\pi_{\theta_k}(s_t,a_t)}{\pi_\theta(s_t,a_t)} < 1-\epsilon$	$A_t < 0$	$(1-\epsilon)A_t$	Yes	No
$\frac{\pi_{\theta_k}(s_t,a_t)}{\pi_\theta(s_t,a_t)} > 1+\epsilon$	$A_t > 0$	$(1+\epsilon)A_t$	Yes	No
$\frac{\pi_{\theta_k}(s_t,a_t)}{\pi_\theta(s_t,a_t)} > 1+\epsilon$	$A_t < 0$	$\frac{\pi_{\theta_k}(s_t,a_t)}{\pi_\theta(s_t,a_t)} A_t$	No	Yes

To achieve the desired behavior we should tune ϵ, serving as an implicit restriction on KL divergence. Empirically, ϵ=0.1 and ϵ=0.2 are values that work well. These values drift a bit from the theoretical foundations of natural policy gradients but empirically it gets the job done.

The algorithm for PPO clip is as follows.

- Input: Initial policy parameter and clipping threshold
- For k = 0,1,2,... do
- Collect set of partial trajectories on policy
- Estimate advantages using any advantage estimation algorithm
- Compute policy update by taking K steps of minibatch SGD (Adam)
- End

2.4 Outcomes & Results

2.4.1 Shower Head Environment

The results for the custom environment using different reward function are as follows:

- Rectangular Reward Function
- Parabolic Reward Function
- Triangular Reward Function

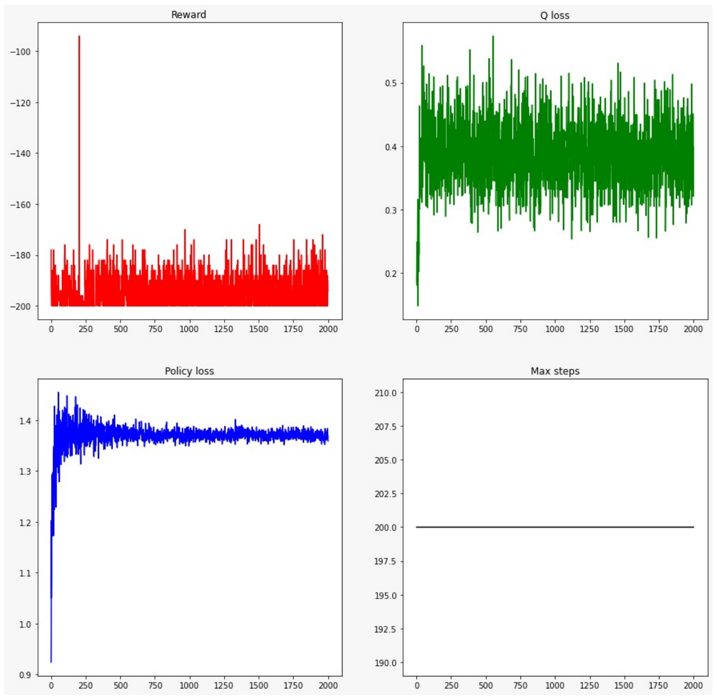


Figure 2: Rectangle: Top left to bottom right: Reward Function, Q-Loss, Policy Loss, Maximum step size per episode

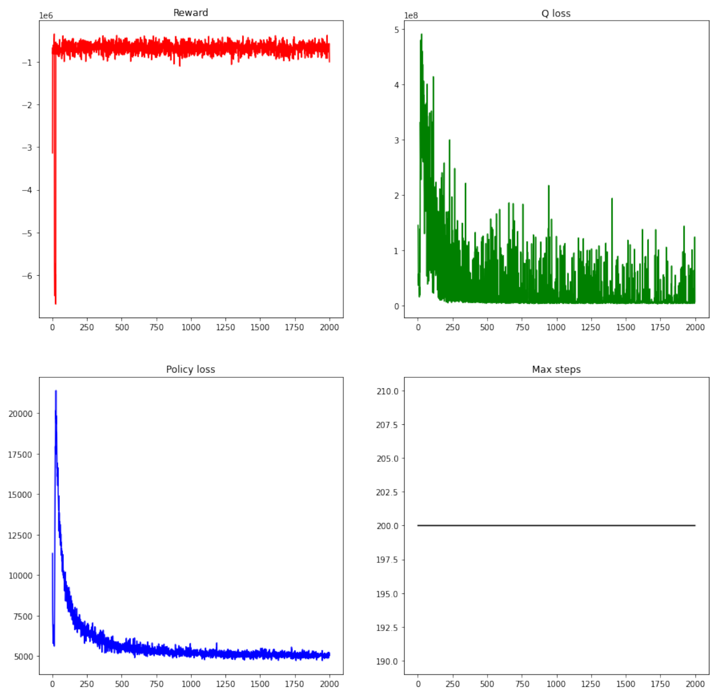


Figure 3: parabola: Top left to bottom right: Reward Function, Q-Loss, Policy Loss, Maximum step size per episode

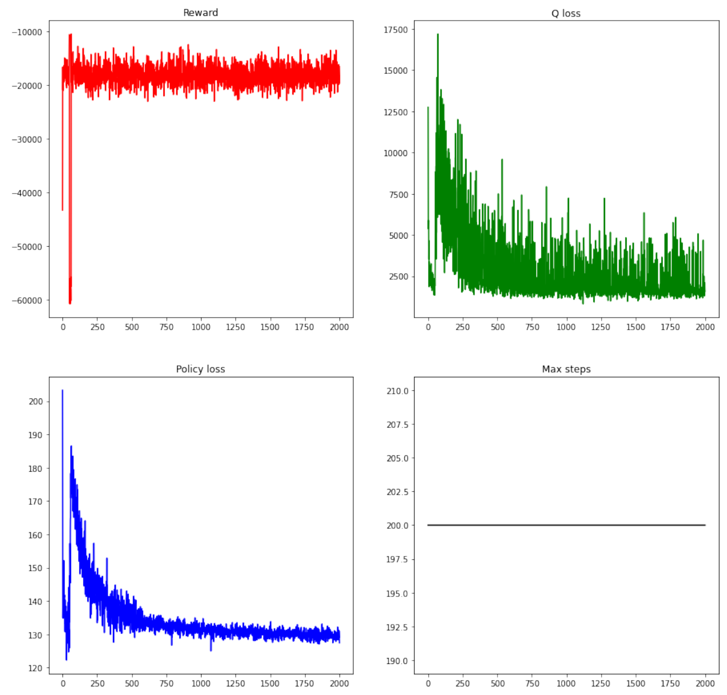


Figure 4: Triangle: Top left to bottom right: Reward Function, Q-Loss, Policy Loss, Maximum step size per episode

As seen from the above results, the rectangular reward function, which had discrete reward values and abrupt discontinuities resulted in a bad return for the agent and this is justified by the Q-loss and the policy loss, which were relatively high. The triangular reward function gave a much-improved return in comparison to the rectangular reward function. The best performance was when the reward was a parabolic reward function, where the reward was reduced gradually for every non-optimal state.

2.4.2 Trading Environment

The result below was visualized using tensorboard. The graph indicated the reward of agents over 200,000 time steps. It can be seen that only a few agents perform well but the ones that perform well have achieved a good account balance.

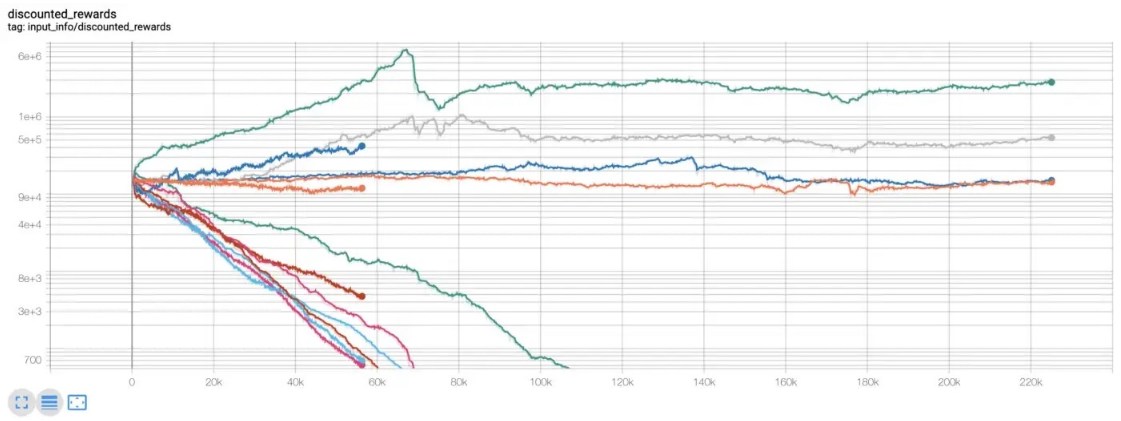


Figure 5: Reward of several agents over 200,000 time steps

Testing the agents in a test environment yields the following results

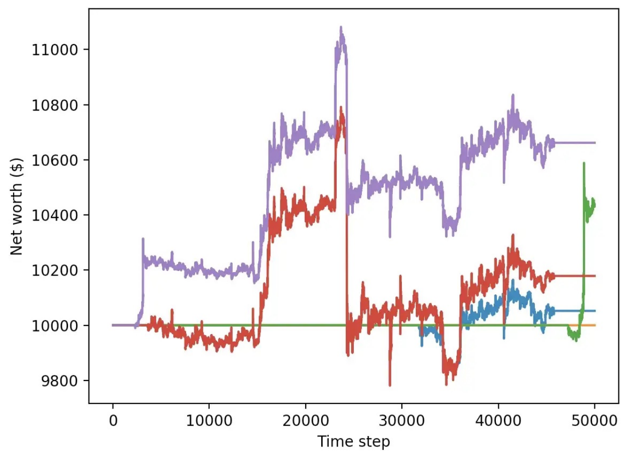


Figure 6: Account balance vs time steps

2.5 Challenges

2.5.1 Shower Head Environment

One of the challenges was finding the best mathematical formulation for the reward function which could best generalize the physics problem at hand. The second was to tune the overall network architecture to result in optimized learning of both the policy and the value function.

2.5.2 Trading Environment

One of the main challenges faced was to render the live output to the screen. We wrote a function to render the output, but as google colab does not support render we were not able to visualize the environment.

3 Conclusion

The DDPG algorithm with the Actor-Critic based learning has helped the agent perform significantly better in the shower head custom environment. We have explored different kinds of reward functions to make the agent's interactions with the environment more robust. Out of the three functions, a gradual change in the rewards, given by the parabolic reward function, resulted in a significant learning curve for the agent. In case of the trading environment, we would like to use as LSTM network in place of the MLP network since recurrent networks are capable of maintaining internal state over time, we no longer need a sliding "look-back" window to capture the motion of the price action. Instead, it is inherently captured by the recursive nature of the network. We would like to modify the reward function to account for risk factor instead of just profit. We plan on tuning the hyper parameters of the model using Bayesian Optimization.

Acknowledgments

This was supported in part by Prof. Brandon Franzke and TA Shashank Karkala.

4 References

- Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. and Wierstra, D. Continuous Control with Deep Reinforcement Learning.arXiv preprint. arXiv:1509.02971, 2015
- Chris Yoon Deep Deterministic Policy Gradients Explained, Towards Data Science, 2019
- Wouter van Heeswijk, PhD Proximal Policy Optimization (PPO) Explained, Towards Data Science, 2022
- Adam King Create custom gym environments from scratch — A stock market example, Towards Data Science, 2019
- Antonio Cappiello Physics control tasks with Deep Reinforcement Learning, Paperspace Blog, 2018

- Caleb M. Bowyer, Ph.D. Candidate What is State in Reinforcement Learning? It is What the Engineer Says it is!, Medium, 2022