USC Viterbi
School of Engineering

Continuous Control Using Deep Reinforcement Learning

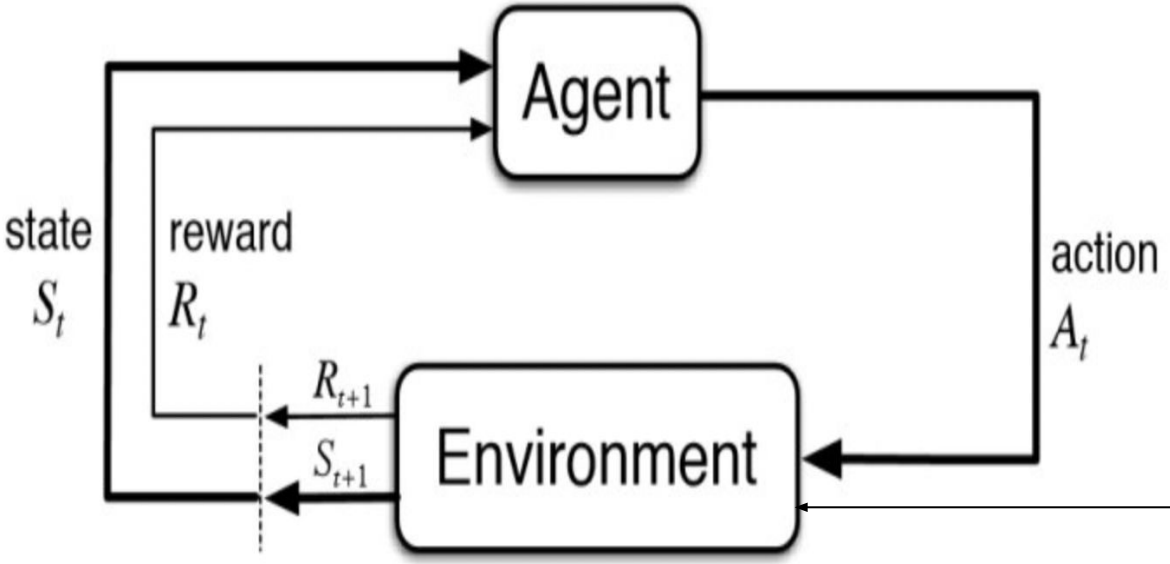
Aditi Bodhankar
Amulya Kallakuri
Nithyashree Manohar

9 December 2022

Introduction

- An **environment** in Reinforcement Learning contains all possible actions that an agent can take, the states an agent can achieve, and the reward an environment can give to an agent for performing those actions.
- We implemented two environments: **Shower and Asset Trading**
- **Deep Deterministic Policy Gradient (DDPG) algorithm** has been used in the shower environment
- **Proximal Policy Optimization** from the Stable Baselines3 library in the asset trading environment
- Custom environments have been created with **OpenAI Gymnasium.**
- Agent has been trained in continuous action space on existing environments, and then on a custom environment.

Reinforcement Learning Overview



In this project, we experimented with **two environments: Shower Head and Asset Trading**

Types of Environment

The types of custom environments we created are as follows

1. Showerhead environment

2. Trading environments:
   - Stock market
   - Bitcoin prices
   - Gold prices

Environment 1: Showerhead

Uses Deep Q-Network - Deep Deterministic Policy Gradient (DDPG) for executing this.

Goal:

Build RL model to adjust temperature automatically to get it within optimal range

Actions:

Turn knob between $-\pi$ and π

States:

Values between 0º to 100º

Reward function:

Rectangular, Parabolic, Triangular

RL for Environment 1: Deep Q-Network

- Model-free RL algorithm
  - No predefined model to make predictions, learns directly from data ⇒ isn't bound by the assumptions and limitations of a model
- Goal
  - Learn optimal Q-function by iteratively updating parameters of network based on data
- Called a "Q-Network" since it approximates Q-values (or expected long term reward) of actions in a given state
- "Experience Replay"
  - Instead of learning from each individual experience, it stores a "buffer" of past experiences and randomly samples from this
  - Helps the network to not be too strongly influenced by one experience

RL for Environment 1: Deep Deterministic Policy Gradient

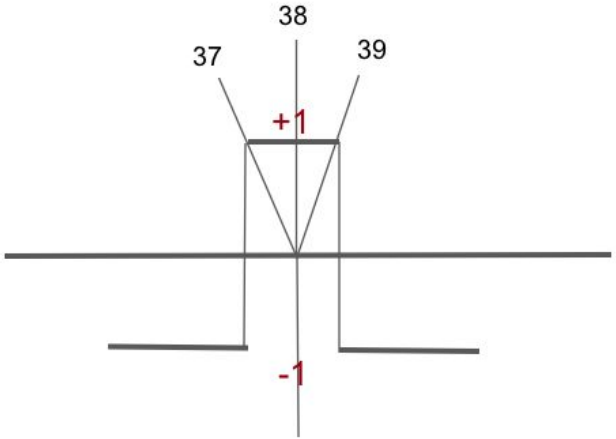
- Model-free
- Uses two neural networks:
  - Actor: Decides what actions to take in an environment.
  - Critic: Evaluates actions and providing feedback to actor.
- Ornstein-Uhlenbeck to model random exploration
- Uses “Function Approximation”
  - Instead of learning the perfect model, it approximates function that maps states to actions ⇒ can work with high-dimensional and continuous action spaces.
- This is useful in situations where the environment is unknown or difficult to model.

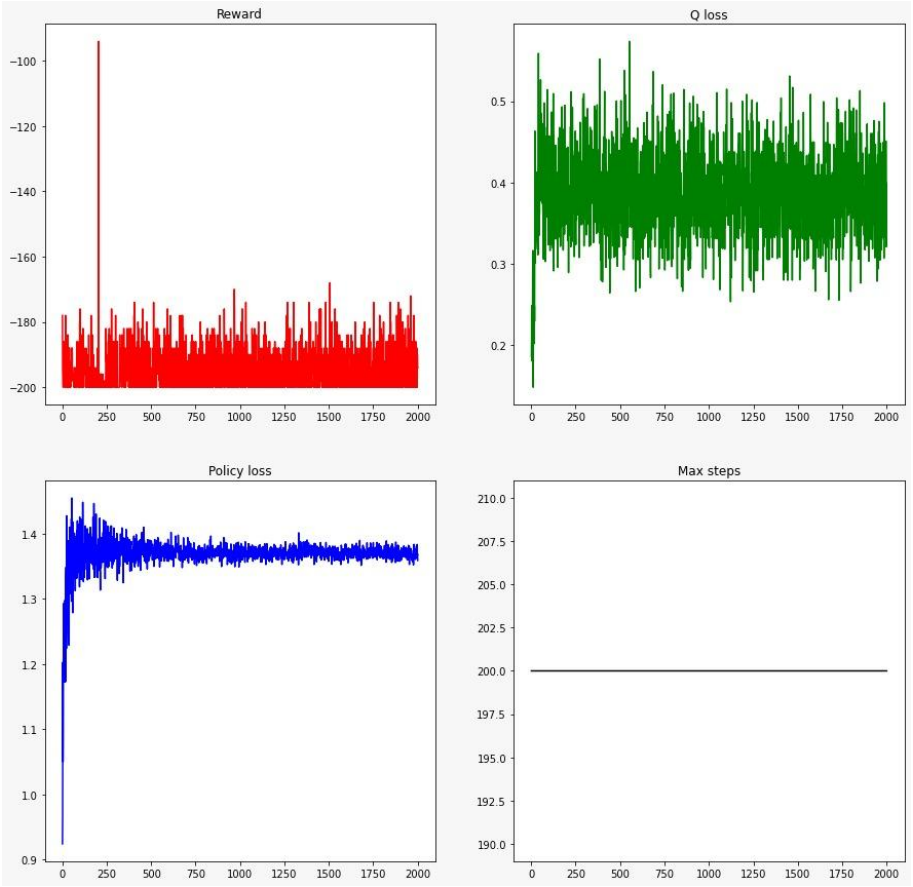
Environment 1: Results

Rectangular Reward Function

Rectangular: +1 $37^{o} \leq T \leq 39^{o}$

-1 elsewhere





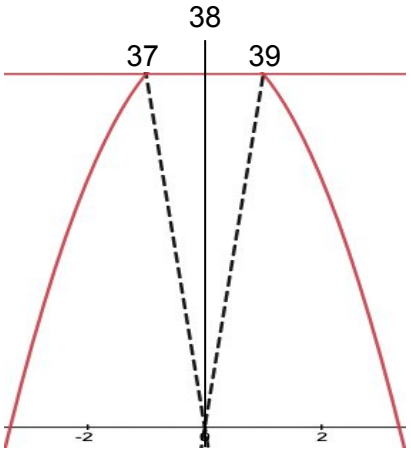
USC Viterbi School of Engineering

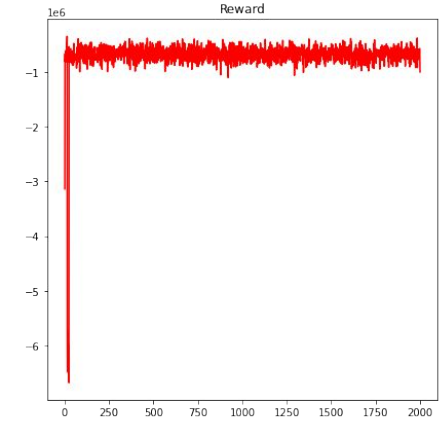
Environment 1: Results

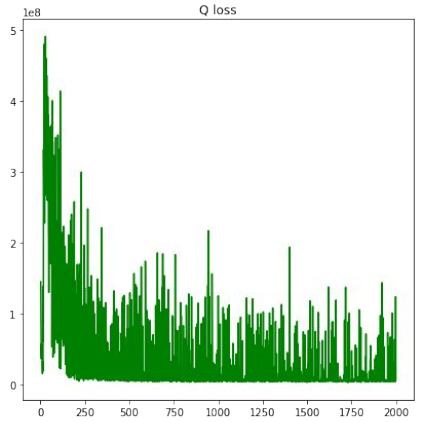
Parabolic Reward Function

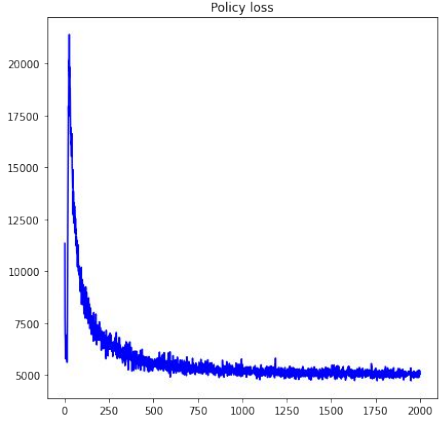
Parabolic: 10 $37^{o} \leq T \leq 39^{o}$

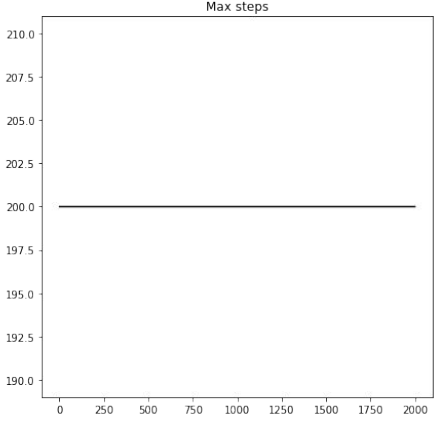
$-x^2+10$ elsewhere









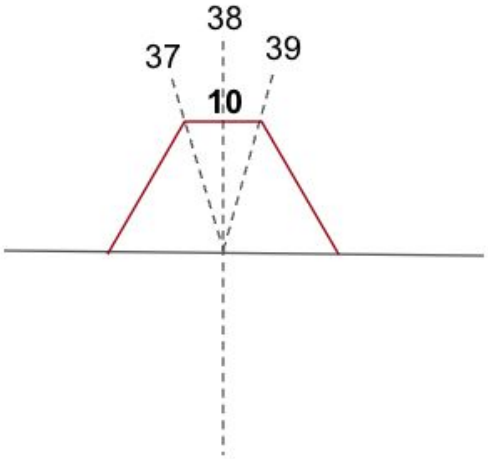


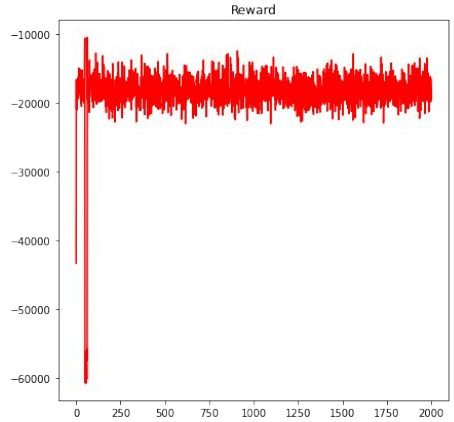
Environment 1: Results

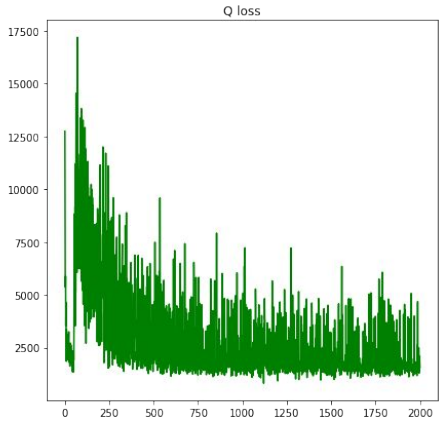
Triangular Reward Function

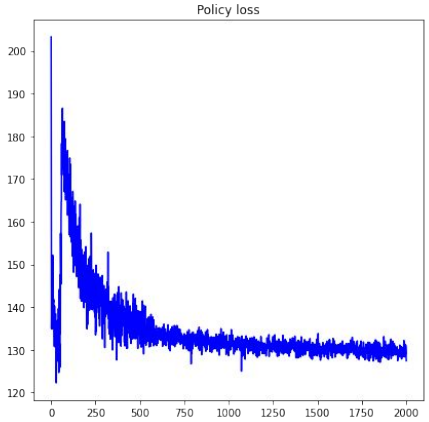
Triangular:

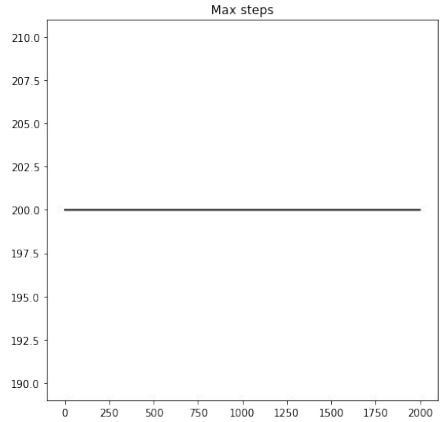
10	$37^o \leq T \leq 39^o$
10 - 2x	$39^o \leq T$
10 + 2x	$T \leq 39^o$











Environment 2: Asset Trading

Uses Proximal Policy Observation to implement this

Observation Space

- We want the agent to see data points:
  - Open price
  - High price
  - Low price
  - Close price
  - Daily Volume
- And other details such as:
  - Account balance
  - Current stock positions
  - Current profit

Action Space

- Buy
- Sell
- Hold
- Do nothing
- Continuous spectrum of amounts to buy/sell

Environment 2: Asset Trading

Reward:

We want to incentivize profit that is sustained over long periods of time.

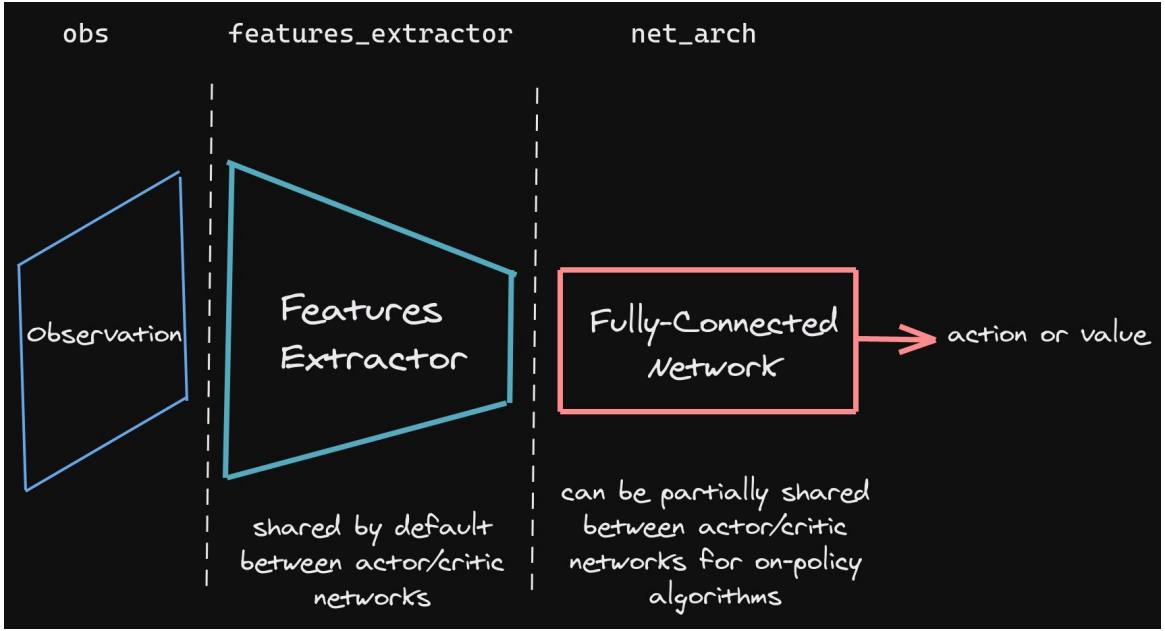
At each step, we will set the reward to the account balance multiplied by some fraction of the number of time steps so far.

Reasons:

- Explore sufficiently before optimizing a single strategy.
- Reward agents that maintain a higher balance for longer.

Environment 2: Algorithm

Proximal Policy Optimization: Stable Baselines3



Source: https://stable-baselines3.readthedocs.io/en/master/guide/custom_policy.html

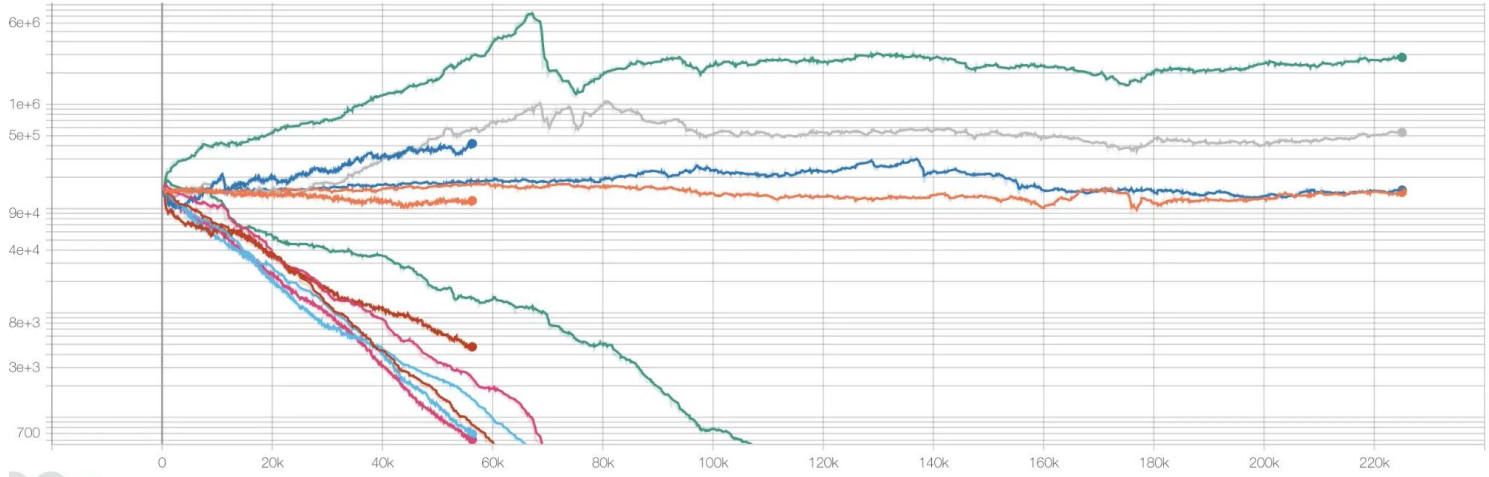
Environment 2: Multilayer Perceptrons

- Multilayer Perceptrons, or MLPs, are a classical type of neural network.
- Policy object implements actor-critic using an **MLP of 2 layers each consisting of 64 neurons**
- Reasons to use MLP Policy:

  1. Time series data
  2. Tabular datasets

Environment 2: Results

Discounted rewards of many agents over 200,000 time steps



- Visualized results using tensorboard
- It can be seen that only a few agents perform well

Environment 2: Extensions

- New columns
  - Return: Percent change of Close Price
  - P: Cumulative product of reward

- Maximize:
  - $\omega R + (1-\omega)C$
  - ω = Fraction of money we put into the stock
  - R = Return from previous day
  - C = Cash Holding

Conclusion

- **Showerhead environment**
  - Created a custom environment with continuous action (shower-knob) space.
  - Explored different kinds of reward functions over the observation (Temperature) space.
  - Improved the agent's performance using Actor-Critic model (DDPG).

- **Trading Environment**
  - Upgrade the existing model to use a recurrent, LSTM policy network with stationary data.
  - Improve the agent's reward system to account for risk, instead of simply profit.
  - Fine tune the model's hyper-parameters using Bayesian optimization.