Happiest Minds

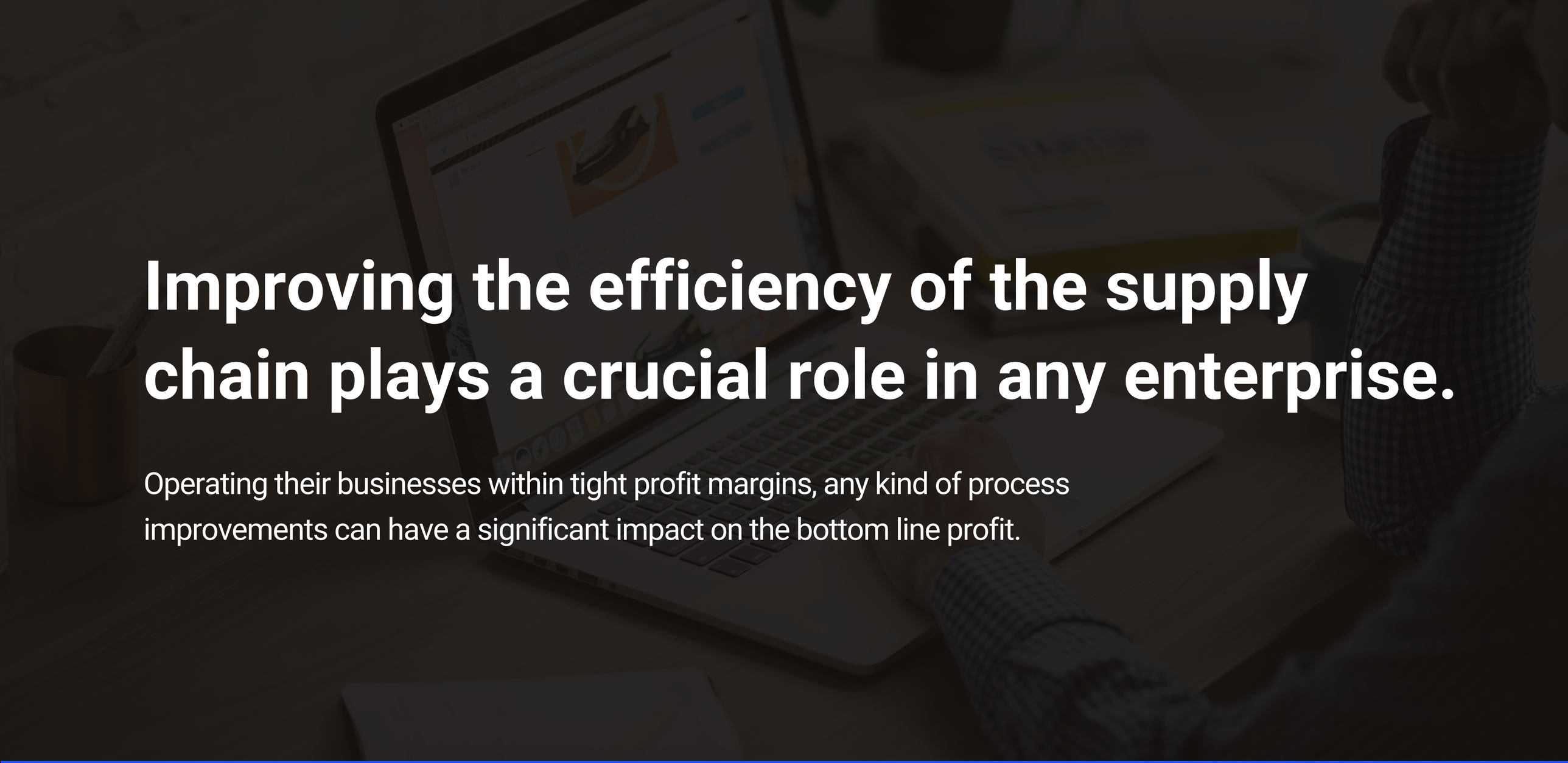
Reinforcement Learning in Supply Chain Management

Introduction

Supply Chain Management

Supply chains are highly complex systems consisting of hundreds if not thousands of manufacturers and logistics carriers around the world who combine resources to create the products we use and consume every day.

- Better ability to predict and meet customer demands
- Fewer process inefficiencies and less product waste
- Improved cash flow and more efficient logistics



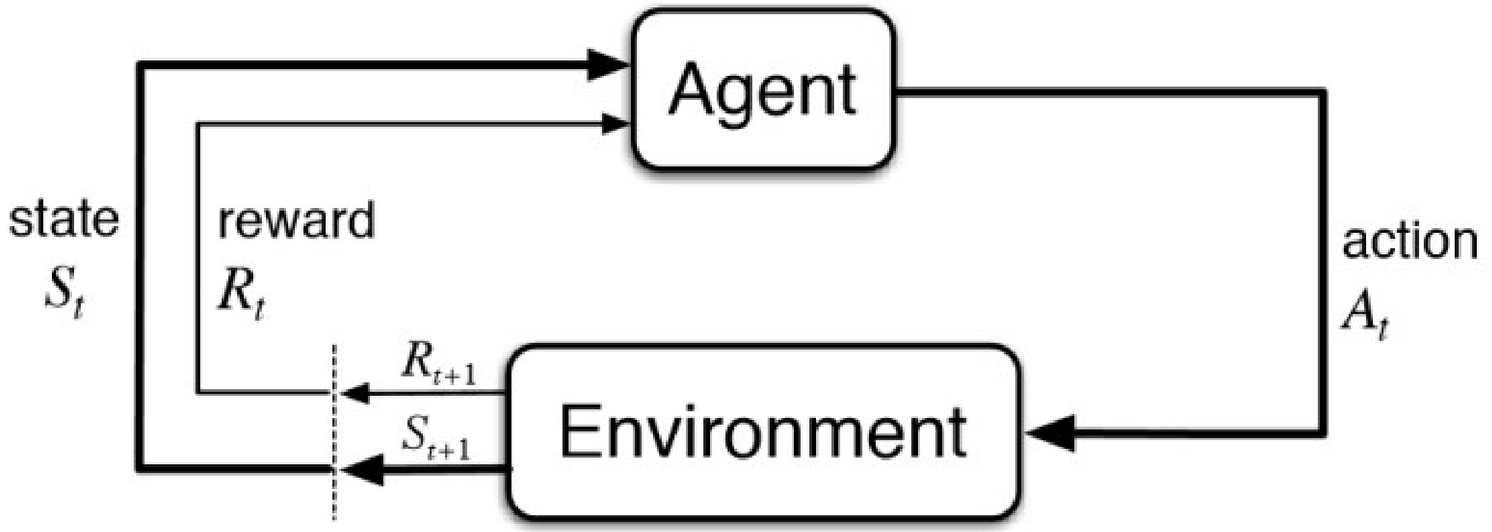
Improving the efficiency of the supply chain plays a crucial role in any enterprise.

Operating their businesses within tight profit margins, any kind of process improvements can have a significant impact on the bottom line profit.

Reinforcement Learning

The goal of Reinforcement Learning (RL) is to learn a good strategy for the agent from experimental trials and relative simple feedback received.

With the optimal strategy, the agent is capable to actively adapt to the environment to maximize future rewards



Where Reinforced Learning comes in Inventory Management

Reinforcement learning is developed to solve sequential decision making problems in dynamic environments. In sequential decision making, a series of decisions are to be made in interaction with a dynamic environment to maximize overall reward. This makes reinforcement learning an interesting method for inventory management.

By being able to handle dynamic environments, it can be possible to create a variable inventory policy that is dependent on the current state of the system, instead of a fixed order policy. Sequential decision making and delayed rewards are also relevant for inventory management.

Problem Statement

Multi-Echelon Inventory Management System

We specifically look at Multi-Echelon Inventory Management System. The project defines a complex environment with a factory and multiple warehouses and aims at optimizing the returns while taking into consideration multiple locations, transportation issues, seasonal demand changes and manufacturing costs.

Methodology

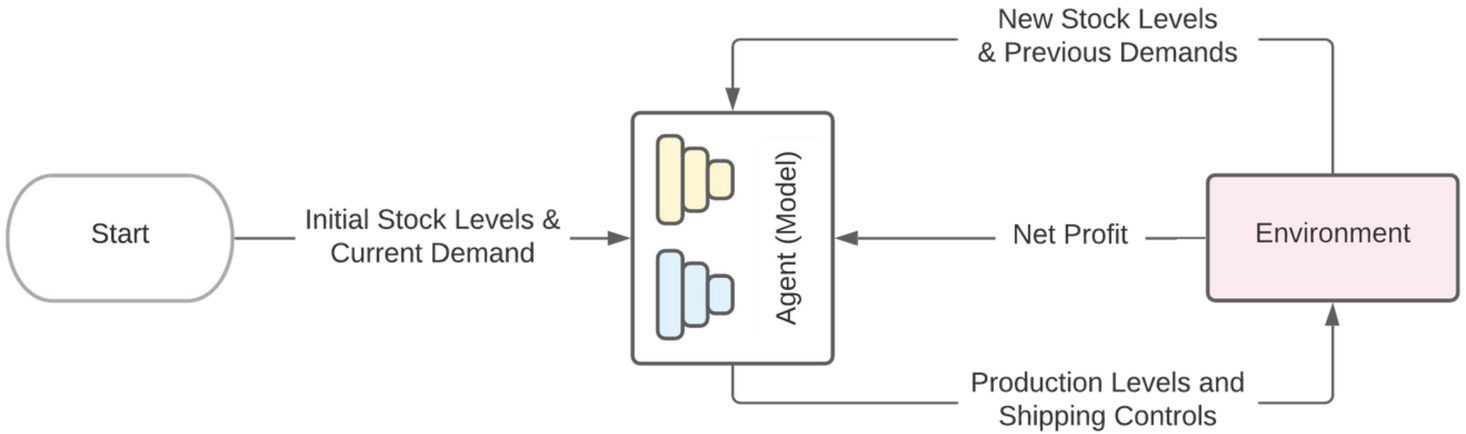
Objectives

Our roadmap is marked by four small goals.

1. Formulate the problem as a MDP
2. Define the environment parameters
3. Train a Deep RL model on the defined environment
4. Compare the results to baselines

Formulation of problem (MDP) for Reinforcement Learning

The state at time t is the tuple of all current stock levels, and demand values of all warehouses for τ previous steps. The action vector consists of production and shipping control



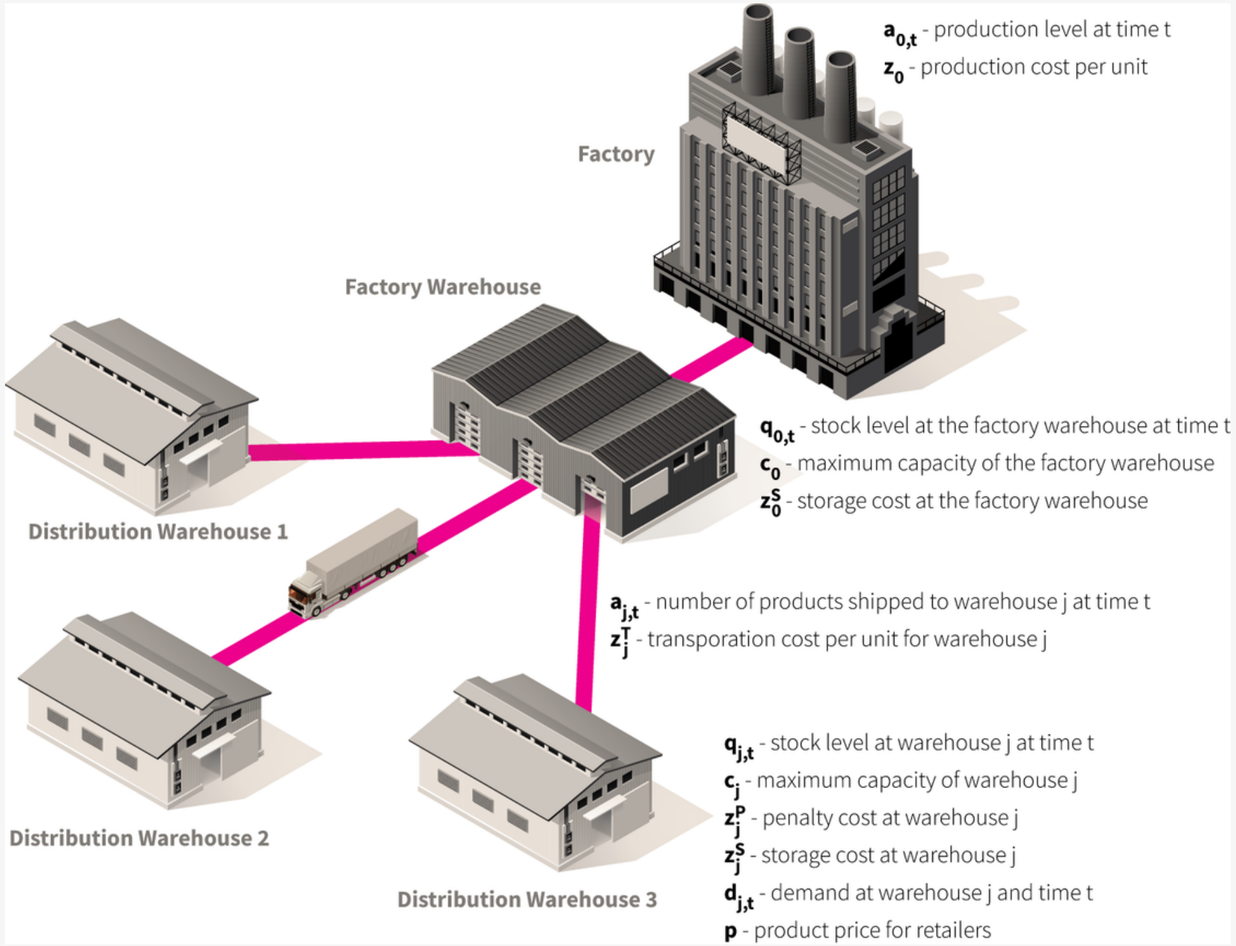
Deep RL

We use two Deep Neural Nets in our algorithm!

Environment Description

One factory, One central factory warehouse, and other distribution warehouses.

This is for demonstration purposes as a proof of concept only.



Environment Assumptions

Besides the various constant parameters of our environment, we make the following assumptions

1. Factory produces a product with a fixed cost
2. Warehouses have a fixed capacity
3. Tangible transportation and storage costs
4. Any unfulfilled demand is carried over to the next time step with a penalty

Environment Parameters

- **State** = [current stock levels at warehouses, list of demand at distribution warehouses for previous predefined amount of time steps]

- **Action** = [production level at factory, shipment level to distribution warehouses]

- **Reward** = Income - expenditure + Penalties
  = Selling Price * (Total demand) - (Production Price * (production level) + Storage Costs + Shipment Costs) + Penalty Cost

Environment Parameters

- $d(t) = \langle d_1(t), ..., d_W(t) \rangle$
- $S_t = \langle q_0(t), q_1(t), ..., q_W(t), d(t-1), d(t-2), ..., d(t-\tau) \rangle$
- $A_t = \langle a_0(t), a_1(t) ..., a_W(t) \rangle$

$$R = p \sum_{j=1}^{W} d_j - z_0 a_0 - \sum_{j=0}^{W} z_j^S \max\{q_j, 0\} - \sum_{j=1}^{W} z_j^T a_j + \sum_{j=1}^{W} z_j^P \min\{q_j, 0\}$$

Technologies Used

OPEN AI GYM

Creating Standardized Environments for simulating interacitons

PyTorch

Implementation of the Internal Deep Neural Networks

Stable Baselines

Implementation of the RL Algorithms (What about RLlib?)

Models & Algorithms

Features of the Model

This is a high level overview of the components and architecture of the model

Actors

- Policy Gradient Algorithms
- A Deep Neural Network that approximates the Policy Function Directly.
- Given a state, results in the optimal action to be taken
- Very Complicated Problem, models usually naive

Critics

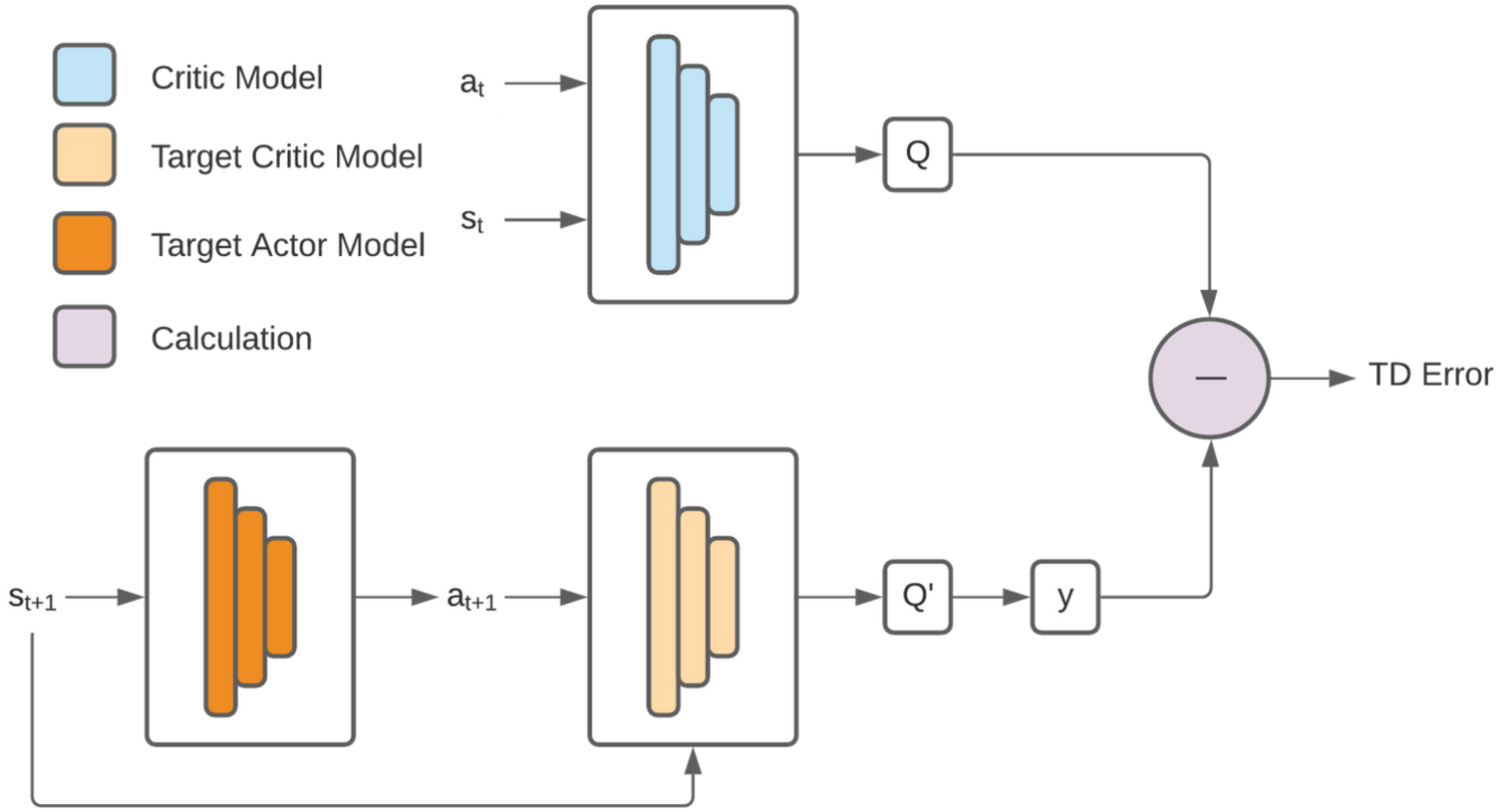
- Deep Q Learning Algorithms
- A Deep Neural Network that approximates the Q Value Function
- Given a state, the model gives a set of actions and their associated Q Values, and we choose the action with the highest expected reward
- Inefficient if the actions are in a continuous space.

Step 1: Deep Deterministic Policy Gradient Algorithm

DDPG is an algorithm with an Actor and a Critic model. The Actor takes the actions, and the critic evaluates how good those actions were.

Model Visualization

Note: TD Error = Temporal Difference Error



What could be the possible solutions? Feel free to pitch in!

Shortcomings of the DDPG Algorithm

Our roadmap is marked by four small goals we need to achieve every three months.

1. Instability during training
2. Highly Sensitivity to the choice of hyperparameters
3. Over Estimation bias of the Critic Network
4. Residual failure to converge using different noise processes

Introduce TD3!

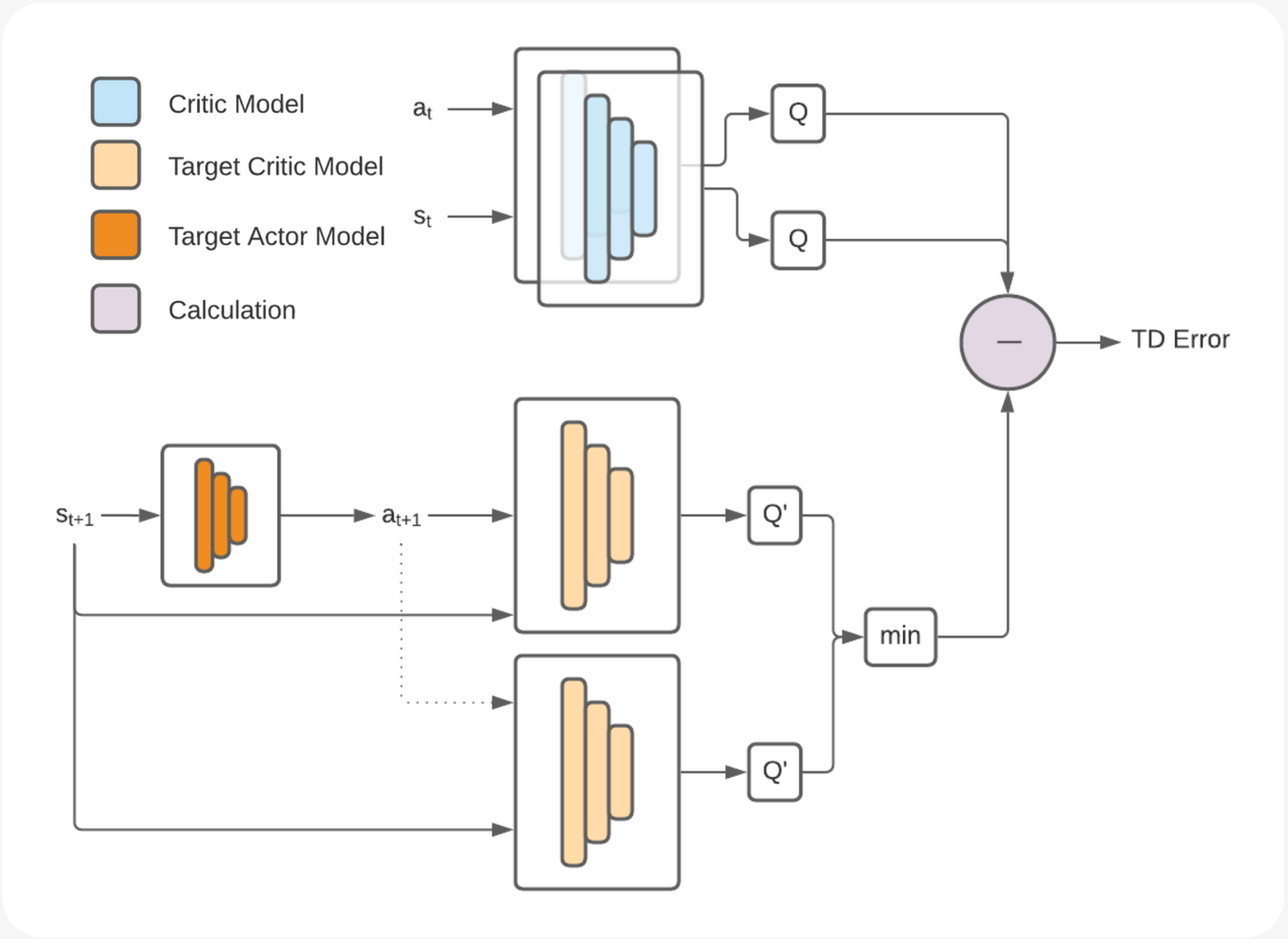
Twin Delayed DDPG overcomes most problems of the DDPG Algorithm

Solutions introduced by TD3

- Two Clipped Critic networks
- Delayed Policy Updates
- Action Noise Regularisation

Novel Solutions introduced

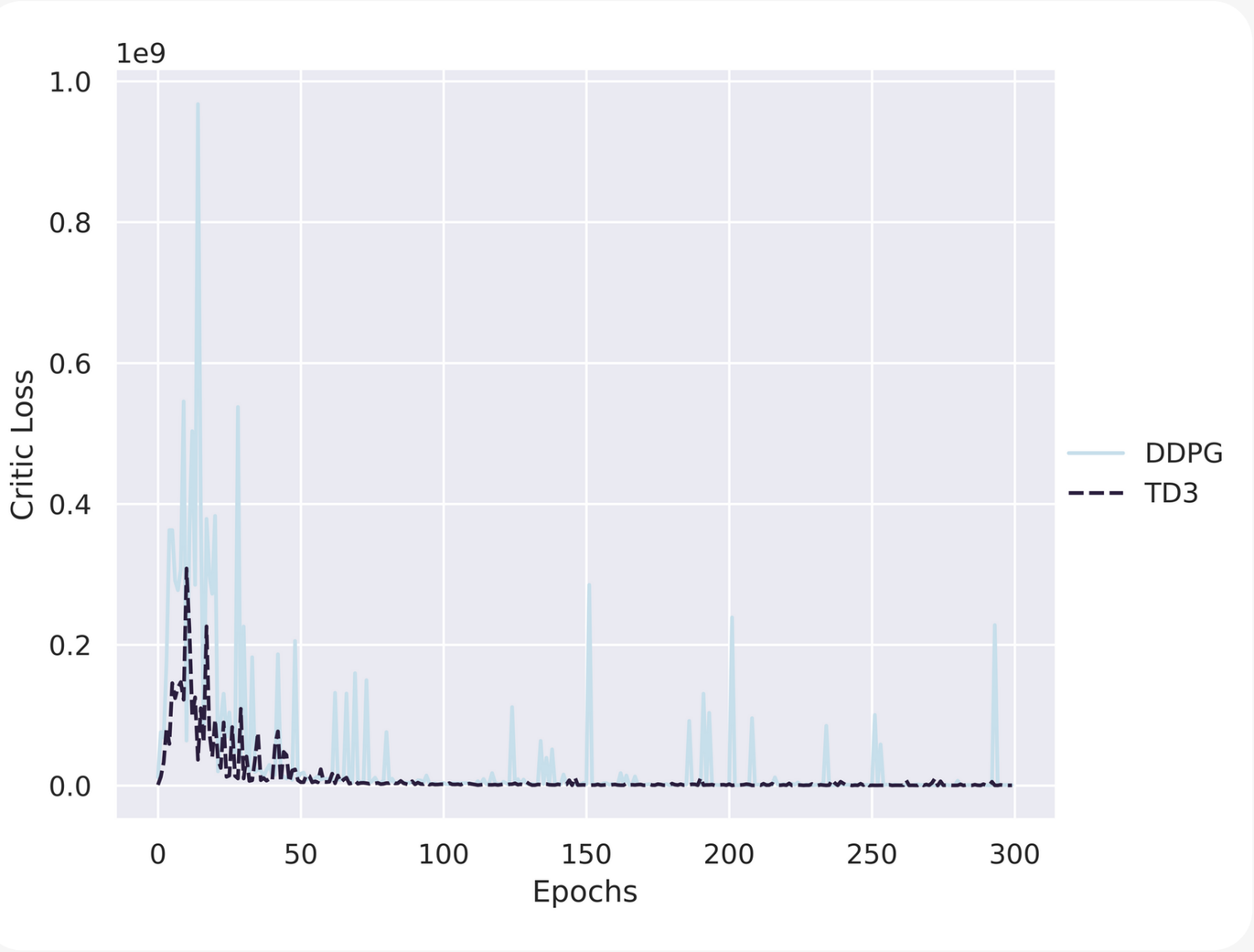
- Use Gaussian Distribution to sample noise rather than Ornstein-Uhlenbeck noise



Comparison of Loss Curves during Training

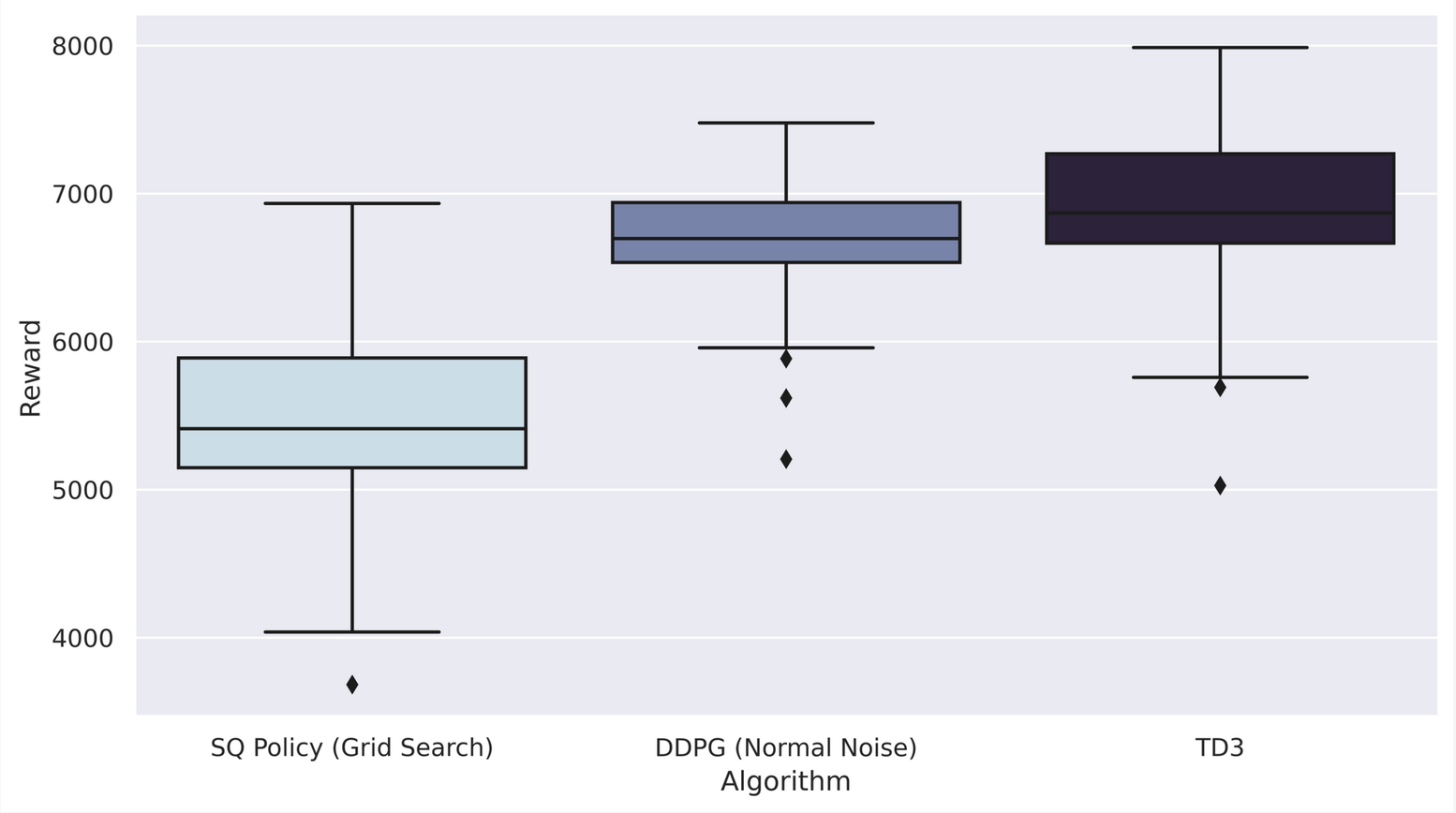
Critic Loss

- Unstable Critic Loss for DDPG Algorithm
- Result of the combination of Ornstein-Uhlenbeck Noise and model architecture
- Large Random Spikes in the Critic Loss affecting final resultant rewards.

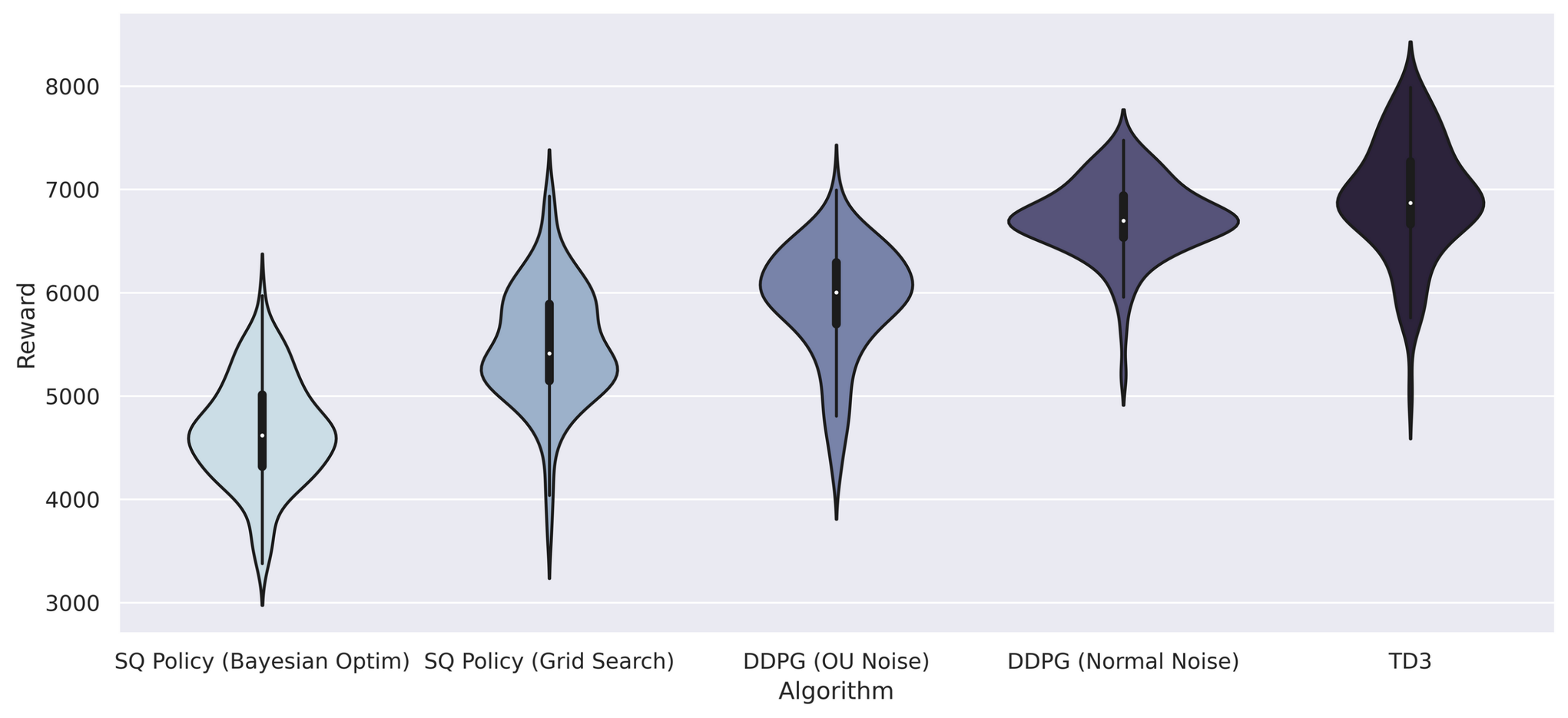


Results

- Mean Rewards:
  - SQ Policy: 5488.21
  - DDPG: 6705.04
  - TD3: 6908.81
- Standard Deviation of all Algorithms relatively Close
- **Our Solution outperforms the best SQ Policy by 25.88% on average.**



Extensive Violin Plot of the Rewards



Conclusion

Experiments with sequence models such as RNNs and LSTMs might greatly improve the performance of such agents, since the data is inherently a time series in nature and the Markovian State Space but a retrofit. This opens the door to using more advanced architectures such as Transformers with Multi-headed attention, and Gated Transformers. This is bound to increase the metrics over our current results. However, for our current solution, the results reach almost the theoretical maximum reward and testing needs to be done with a larger and more complicated environment.


Thank you!