Capital Normal University High School – Germantown Academy
Curriculum Partnership Program

PROJECT BASED LEARNING CAPSTONE

PROJECT REPORT

AP COMPUTER SCIENCE A & AP STATISTICS: MARS LANDER TRAINING THROUGH DEEP REINFORCEMENT LEARNING(DEEP Q NETWORK) AND STATISTICAL OPTIMIZATION

Muchen Liu

3rd July 2024

Chapter 1

Introduction

1.1 Context

The training of a Mars Lander Machine using Deep Reinforcement Learning and Statistical Optimization.

1.2 Motivation

The Project is motivated to enable human-free landers to adjust policy and achieve successful landing on the Mars.

1.3 Goals

The Project aims at two objectives. 1. Successfully train a Mar Lander under customized OpenAI Gym environmet(Brockman et al. 2016) 2. Optimize the training process in order to achieve models with better generalization abilities.(Farebrother, Machado and Bowling 2018)

1.4 The Environment

The training process was based on the OpenAI Gym's lunar lander environment(customized). The lunar lander environment was designed by OpenAI to train landers on the moon. See Table 1.1 for the defaulted parameters of this environment.

Table 1.1
Default Parameters

Continous	Gravity constant	Enable wind	Wind power	Turbulence power
False	1.6	False	15.0	1.5

During the training process, the environment was customized with the following parameter settings.

The agent(lander) receives 8 variables from the environment, namely, a 1x8 matrix. Each observed variable indicates following: 1. X coordinate 2. Y coordinate 3. Linear velocity in X 4. Linear velocity in Y 5. Angle 6. Angular Velocity 7. A boolean representing whether the left leg contacted ground 8.A boolean representing whether the right leg contacted ground.

TABLE 1.2
Customized Parameters

Continous	Gravity constant	Enable wind	Wind power	Turbulence power
False	3.71	True	15.0	1.5

1.5 THE MODEL

We used a Deep Q Network to train the agent. The agent's action space is a 1x4 matrix, indicating the Q-value for: 1. Do nothing 2. Fire the left engine 3. Fire the main engine 4. Fire the right engine.

The Model receives an input of the agent's observations and outputs a 1x4 matrix, each element indicating the expected Q-value of choosing each action.

1.6 OPTIMIZATION

Considering that we want a model that is workable and maintains at a good level, we want it to be fluctuating less. Considering the complexities and unpredictability of the Mars environment, we would like to find a model that has the best generalization ability, namely, one that is able to conduct successful landings in preferably most environments. This is usually reflected by the mean reward gained under different environments by the agent.
We propose to adopt algorithms to reduce the degree of fluctuation and consider the mean reward in different environments as our metric. We propose two methods to achieve the aforementioned objective. 1. By utilizing the standard deviation after the model has already gained sufficient and consecutive rewards into the reward function. 2. By stifling exploration after sufficient and consecutive rewards are gained.

Chapter 2

Methods

2.1 Training

2.1.1 The Deep Q Network

The Deep Q Network(DQN) is a neural network designed for training reinforcement learning agents. The network was invented in light of the rising tools of neural networks, and in the shortage of tabular methods for RL has demonstrated.(Z. T. Wang and Ueda 2021) The DQN was the optimized version of a Q table, in which we find the Q-value for each state-action pair.(Z. Wang, Schaul, Hessel, H. v. Hasselt et al. 2016) Under certain situations where the number of states of the environment are astronomical and incompatible, the Q-table no longer fits. The DQN, as a neural network, takes the observation of the environment(in this case, the 1x8 matrix) as input and outputs the Q-value for each possible step in the action spaces(in this case, the 1x4 matrix). A standard DQN with experience replay is shown in Figure 2.1. (Z. Wang, Schaul, Hessel, H. v. Hasselt et al. 2016)

Under this environment, the DQN is designed as in Figure 2.2.

2.1.2 Training process and evaluation

At each step, when a mini-batch is sampled, a gradient descent algorithm is performed with loss function MSE on temporal difference error.

$$L = [r + \gamma \underset{a'}{\mathrm{argmax}} Q(s', a') - Q(s, a)]^2 \tag{2.1}$$

where

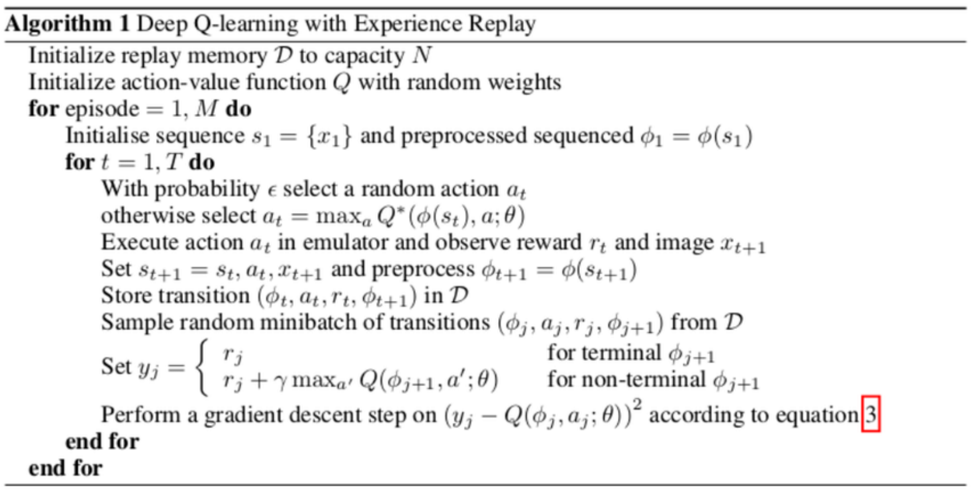
$$r + \gamma \underset{a'}{\mathrm{argmax}} Q(s', a') \tag{2.2}$$

is the expected Q-value for the current state-action pair, that is, the current value and the discounted maximum Q-value of the next state-action pair.

The training is iterated 10000 times, and the reward function is shown in Figure 2.3.

In order to evaluate the training function and the performance of a model, we propose 1. To sample every 100 episodes of training and observe the distribution of these 100 samples' mean and standard deviation. 2. To deploy the globally optimized model under one training method into 10 different Mars environments, differing in the power of wind and the power of turbulence.

The distribution is shown in Figure 2.4.



Algorithm 1 Deep Q-learning with Experience Replay

```
Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize action-value function $Q$ with random weights
for episode $= 1, M$ do
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ do
        With probability $\epsilon$ select a random action $a_t$
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
    end for
end for
```

FIGURE 2.1
Deep Q Network with experience replay

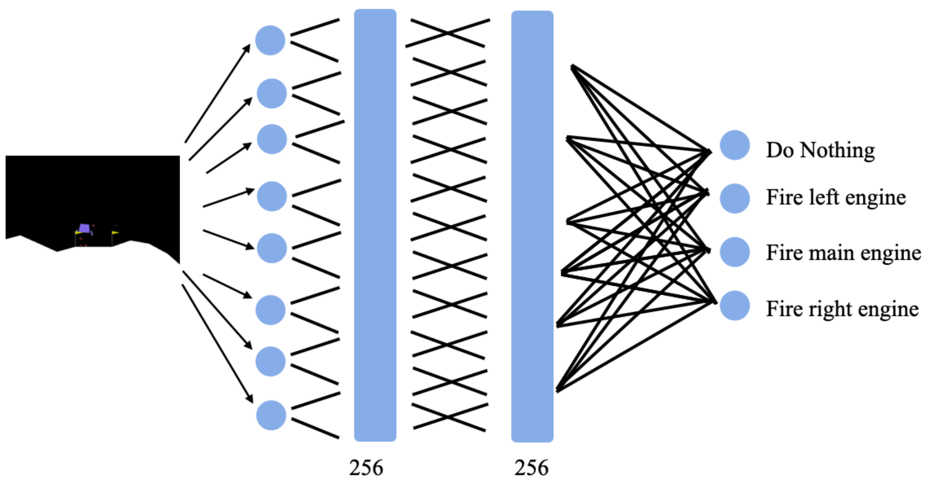


FIGURE 2.2
DQN design

It can be easily observed that the model experienced a severe drop in episodes around 5000 to 7500, climbed back from 7500 to 7800, experienced another drop at around 8000 episodes and finally re-climbs back.

We suspect an overfitting occurred (Zhang, Ballas and Pineau 2018), therefore, the L2 regularization was performed.

2.2 OPTIMIZATION

2.2.1 METHOD 1: L2 REGULARIZATION

The L2 regularization was referring to adding the squared of the parameters in the neural networks in to the loss function in order to punish high values of parameters which tends to incur overfitting. Namely,

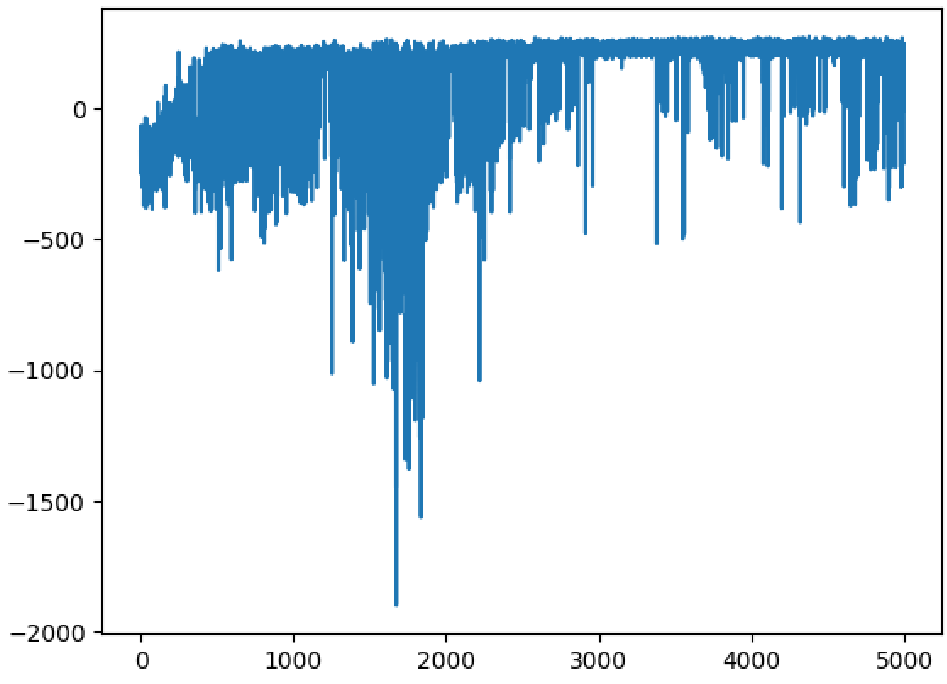


FIGURE 2.3
Reward function

the L2 regularization modifies the loss function into

$$L = [r + \gamma \underset{a'}{\text{argmax}} Q(s', a') - Q(s, a)]^2 + \lambda \sum_{i=0}^{N} W_i^2 \tag{2.3}$$

We then propose a new statistical method to attain less fluctuating models.

2.2.2 METHOD 2: REWARDS FOR LESS FLUCTUATION

As the goal was to attain a model with less fluctuation after it reached a sufficiently satisfactory level, we propose that, after reaching such a level, the reward function be added with a number that, if the current reward deviates no bigger than 40, than the previous consecutive 7 rewards, equals 40. The reward function is now

$$R(s) = \begin{cases} R(s) + 40, \dfrac{\sum_{i=s-7}^{s-1} R(s)}{7} > 200 \ \ \& \ \ \sigma(R(s)...R(s-7)) < 40 \\ R(s), else \end{cases} \tag{2.4}$$

In the light of another possible reason that incurred the severe drop - exploration, we propose the third method that represses exploration after a consecutive episode of success.

2.2.3 METHOD 3: SUCCESS REPRESSION AND REWARDS FOR LESS FLUCTUATION

In reinforcement learning, at each step, the agent needs to know whether it should explore, that is, to take a random step, or to exploit, that is, to take the optimal step produced by the model. The function with which an agent can achieve that is called $\epsilon - greedy$.

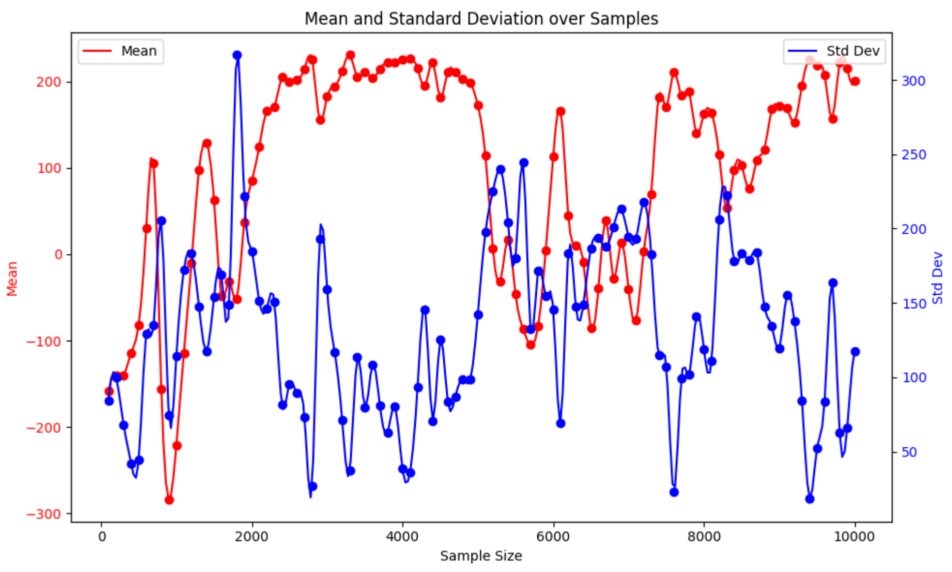


FIGURE 2.4
Mean and Standard deviation distribution

Before training, ϵ will be assigned a number, usually 0.99; β, the epsilon decay constant, is assigned a number, usually 0.95. At each step, a random number $\alpha \in [0.0, 1.0)$ will be generated.

If $\alpha < \epsilon$, Explore
If $\alpha >= \epsilon$, Exploit

After each episode, ϵ is updated by

$$\epsilon := \epsilon \times \beta \tag{2.5}$$

Using the $\epsilon - greedy$ method, the model will tend not to explore that much as episodes go on. Yet, we still discovered some frequent and severe drops as episodes increased. Therefore, we propose to repress the tendency for exploration after the model has achieved 3 successes. By success, we refer to 7 consecutive rewards, each of which is greater than 200. If the model achieved 1 success, we set

$$\epsilon := \epsilon \times 0.55 \tag{2.6}$$

If it reached 3 successes, we set

$$\epsilon := \epsilon \times 0.01 \tag{2.7}$$

In the meantime, the Method 2 and Method 1 is also adopted, with extra credit reward being 80. Therefore, after 3 successes, the model will not, at sufficiently great probability, further its exploration and instead stick to its current policy.

CHAPTER 3

RESULTS

3.1 FLUCTUATION RESULTS

The resultant distribution of mean and standard deviation for every 100 episodes for **Method 1** is shown in Figure 3.1

We observe that the fluctuation decreased dramatically compared to the one shown in Figure 2.4. Yet we still observe a standard deviation between 80 and 150, even some reached 400; we ascribe this to potential 'bad' explorations.

The resultant distribution of mean and standard deviation for every 100 episodes for **Method 2** is shown in Figure 3.2

We observe even less fluctuation compared to M1, with a standard deviation between 60 and 120; only one of them reached 200, but it was in a very early stage. The degree of fluctuation dropped significantly and successfully.

The resultant distribution of mean and standard deviation for every 100 episodes for **Method 3** is shown in Figure 3.3

We observe that the fluctuation of Method 3 is the least in all methods, with a standard deviation between 30 and 140. This suggest that the method 3 we proposed worked dramatically well in reducing the degree of fluctuation.

However, it's important that we focus also on the generalization abilities of each method.

3.2 GENERALIZATION RESULTS

In order to test the generalization abilities of the models trained by each method, we customized ten environments. See Figure 3.4 for details.

By deploying the optimal model trained by each method into 10 environments, we will be able to compare, using the mean of rewards for 50 episodes in each environment, each model. See figure 3.5 for results.

Please note 1. NF G1 refers to No Fighting 2. F CS G2 refers to Fighting using Computer Science methods(L2) 3. F STATS G3 refers to Fighting using statistical method 1 3. F STATS G4 refers to Fighting using statistical method 2.

Given the randomness of the Environments and the closeness of average rewards of G3 and G4, a separate experiment was performed to investigate whether G3 was better at generalization than G4. The training

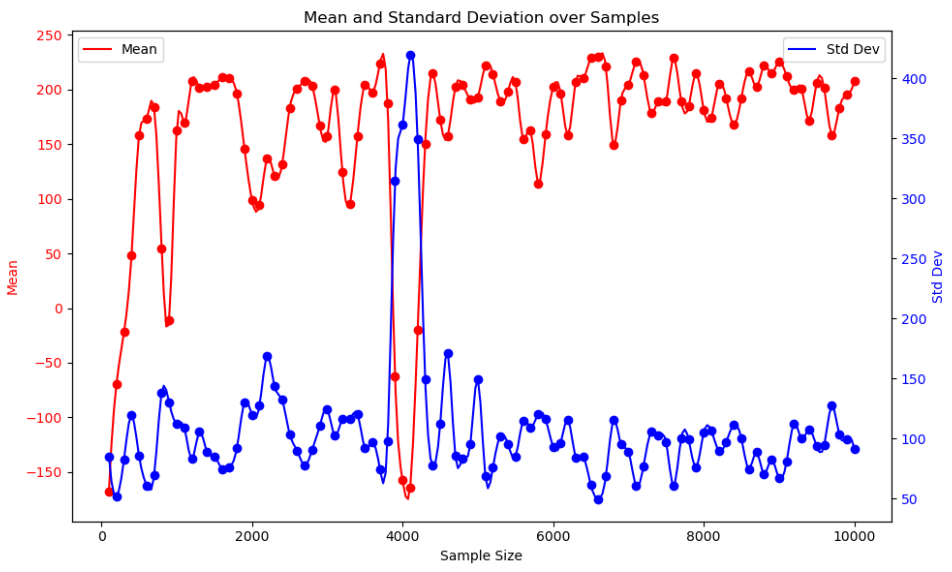


FIGURE 3.1
M1 distribution

of G3 and G4 was performed, and the optimal model of two method-trained models was selected and put into 10 environments to calculate the mean of rewards for 50 episodes in each environment for each model. See Figure 3.6 for the result.

From Figure 3.6 , we see G3 is better at generalization than G4.

Out of curiosity, we also explored the question of whether a model performs better with more training or not. We selected G4 for this experiment. We separated G4 into 3 groups: episode 0-3300, episode 3301-6600, and episode 6601-10000. We selected an optimal model from each of these 3 groups and compared the mean of rewards for 50 episodes in each of the 10 environments for each model. See Figure 3.7 for result.

From Figure 3.7, generally, in terms of average reward and generalization, a model performs better with more training.

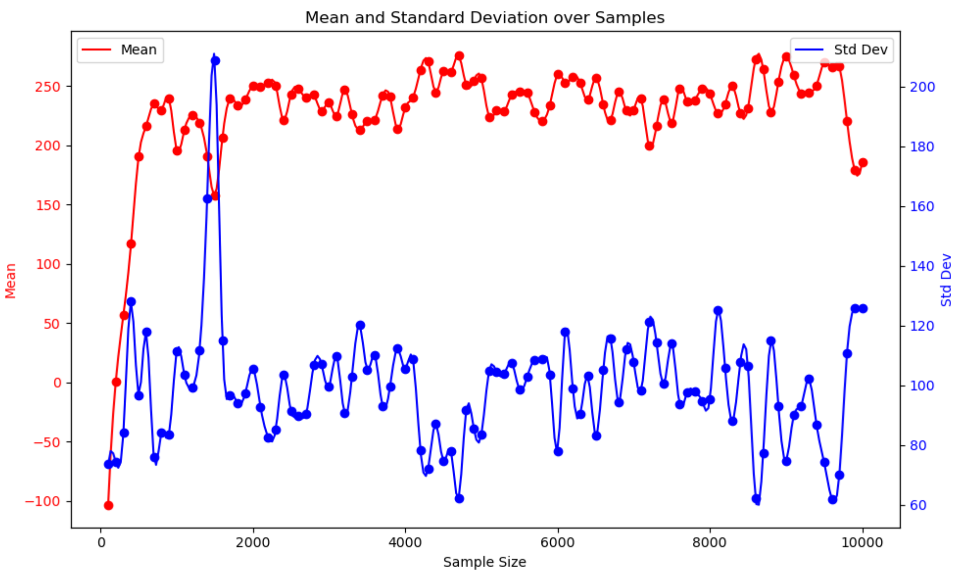


FIGURE 3.2
M2 distribution

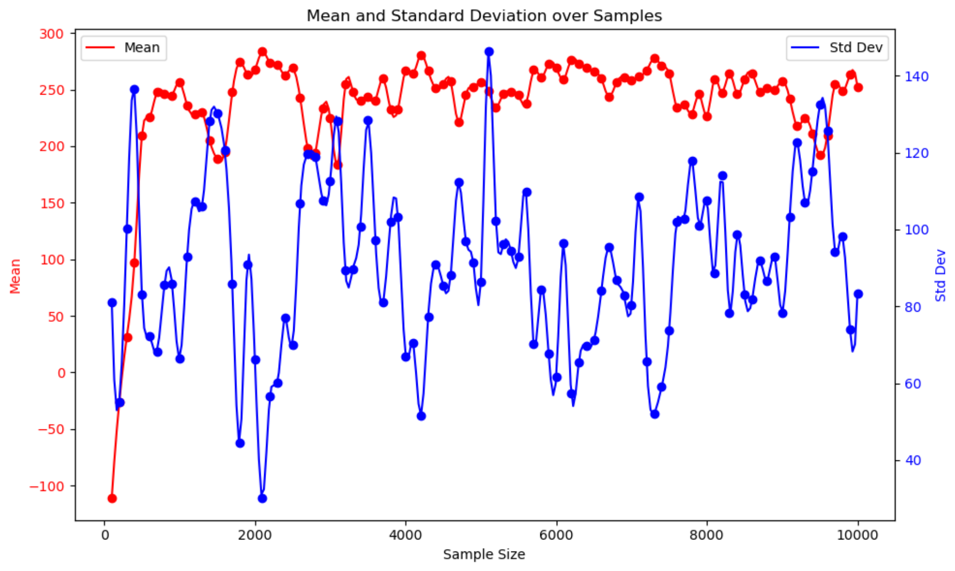
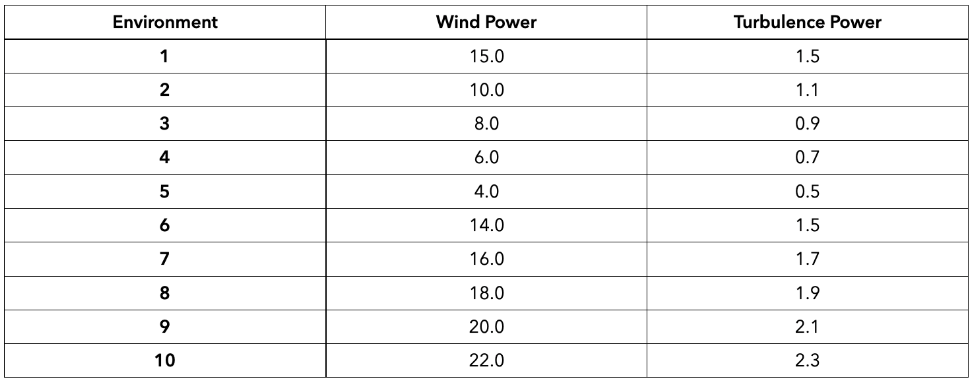


FIGURE 3.3
M3 distribution



Environment	Wind Power	Turbulence Power
1	15.0	1.5
2	10.0	1.1
3	8.0	0.9
4	6.0	0.7
5	4.0	0.5
6	14.0	1.5
7	16.0	1.7
8	18.0	1.9
9	20.0	2.1
10	22.0	2.3

FIGURE 3.4
Customized parameters for 10 environments

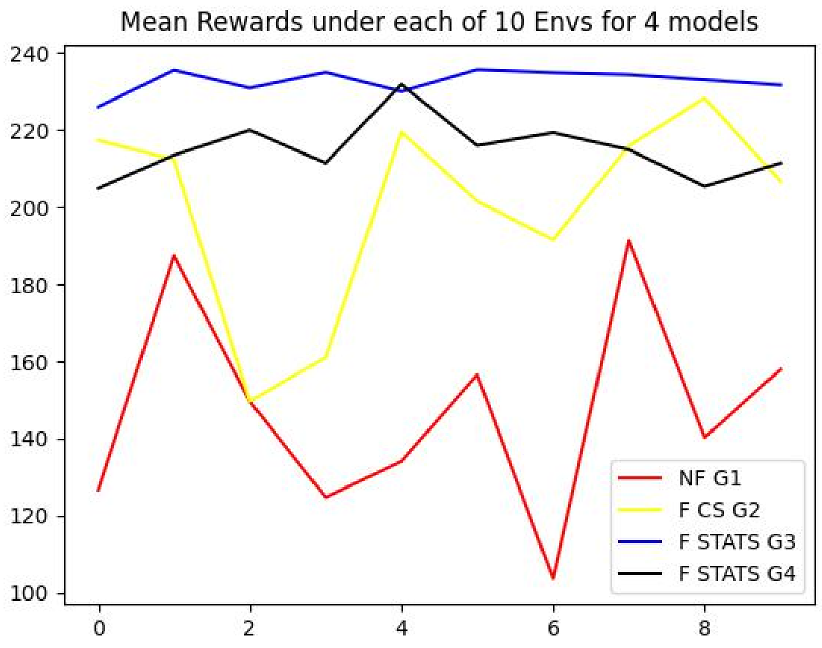


FIGURE 3.5
Mean Rewards under each of 10 Envs for 4 models

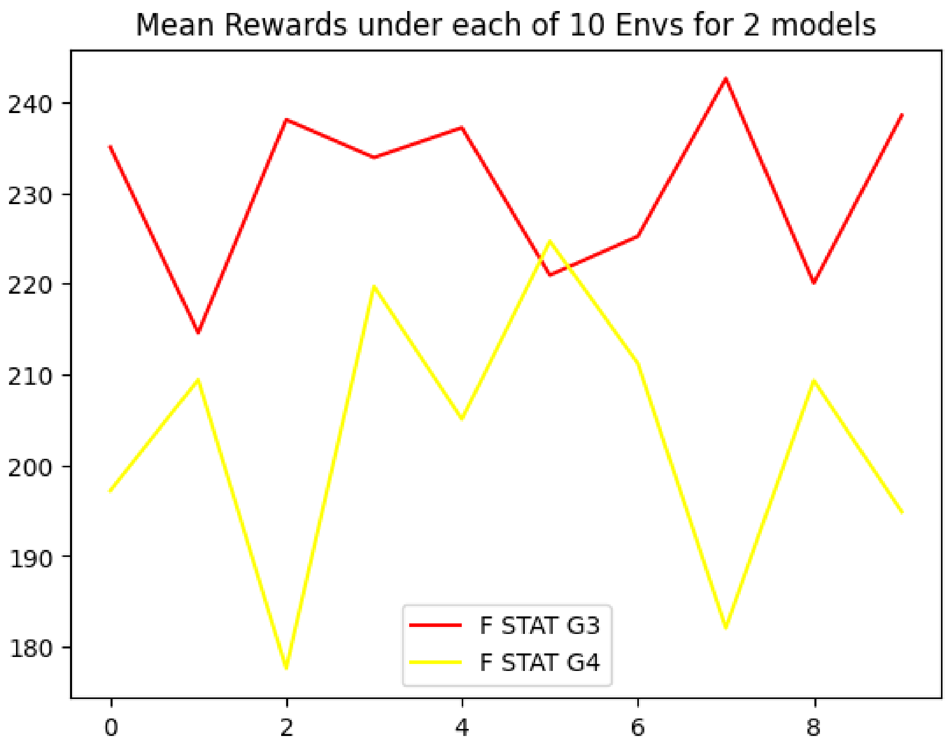


FIGURE 3.6
Mean Rewards under each of 10 Envs for 2 models

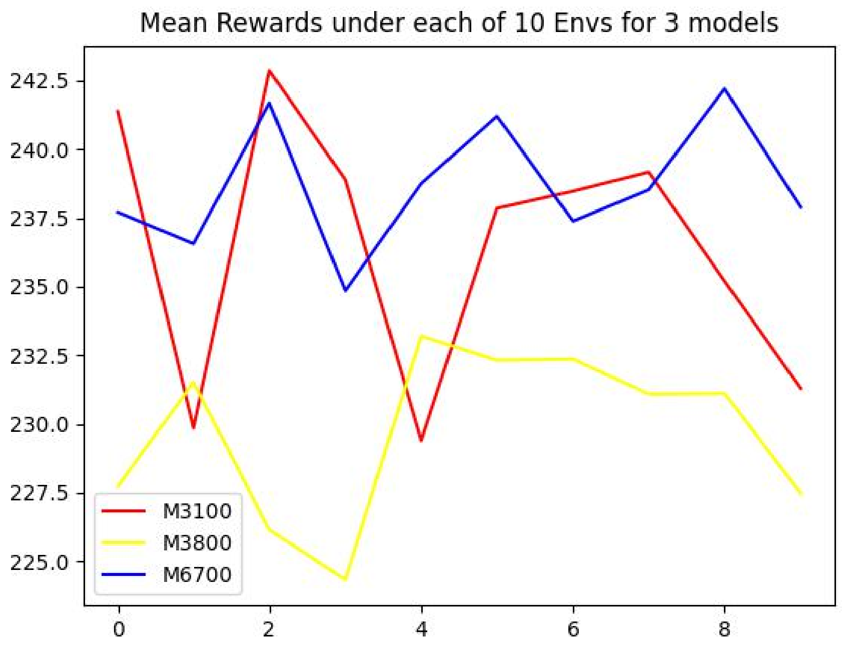


FIGURE 3.7
Mean Rewards under each of 10 Envs for 3 models

CHAPTER 4

DISCUSSION

We consider the agent possibly sticking to sub-optimal policies after training and repressing its tendency to explore. Therefore, although the outcome may turn out smooth, the model is, in fact, unsuitable for real-world, more complex, and stochastic environments. Furthered in the question, the sparse reward problem and the sub-optimal probability are currently being addressed by the Go-Explore algorithm by Uber AI Lab(Ecoffet et al. 2021), which presents a different approach to the problem. Instead of utilizing standard deviation in the reward function, Go-Explore proposes keeping a record of the trajectory the agent attained and exploring from the endpoint of these trajectories.

Chapter 5

Conclusion

5.1 Result Analysis

This project proposes two methods to address the problem of sparse rewards and fluctuation. The first method of statistical optimization with 40 extra reward points has the second least fluctuation and the best generalization ability. The second method of severely repressing the exploration tendency achieved the least fluctuation and somehow the second-best generalization ability.

5.2 Limitations

As discussed in Chapter 4, the methods we proposed retain a probability and tendency of incurring sub-optimal policies. Therefore, further research focusing on reducing the probability of sticking to sub-optimal policies is required.

Bibliography

Brockman, Greg et al. (2016). *OpenAI Gym*. arXiv: `1606.01540 [cs.LG]`.

Farebrother, Jared, Marlos C Machado and Michael Bowling (2018). 'Generalization and Regularization in DQN'. In: *arXiv preprint arXiv:1810.00123*.

Wang, Zhikang T. and Masahito Ueda (2021). 'Convergent and Efficient Deep Q Network Algorithm'. In: *arXiv preprint arXiv:2106.15419*. `https://doi.org/10.48550/arXiv.2106.15419`.

Wang, Ziyu, Tom Schaul, Matteo Hessel, Hado van Hasselt et al. (2016). 'Dueling Network Architectures for Deep Reinforcement Learning'. In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning-Volume 48*. JMLR.org, pp. 1995–2003.

– (2016). 'Dueling Network Architectures for Deep Reinforcement Learning'. In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. JMLR.org, pp. 1995–2003.

Zhang, Amy, Nicolas Ballas and Joelle Pineau (2018). 'A Study on Overfitting in Deep Reinforcement Learning'. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 1078–1089.

Ecoffet, Adrien et al. (2021). 'First return, then explore'. In: *Nature*. Vol. 590. 7847. Nature Publishing Group, pp. 580–586.