MARINE & MARITIME INTELLIGENT ROBOTICS

UNIVERSITY OF TOULON

May 15, 2024

Deep Reinforcement Learning Report

By:

Abubakar Aliyu BADAWI

N^o **étudiant: 022303696**

Submitted to:

Prof. J. Arjona-Medina

Contents

Introduction

This report contains some tasks relating to Imitation Learning, particularly focusing on Behavioral Cloning and refining outcomes through Dataset Aggregation to effectively train models by mimicking expert behaviors. The analysis extends into Deep Q Networks (DQN), a robust technique for solving complex control tasks by learning the optimal action-value function. Lastly, I use Policy Gradients with a specific emphasis on Proximal Policy Optimization (PPO), which refines the policy gradient approach to improve training stability and efficiency. Each section aims to implement these techniques and try to change some hyperparameters to see the effect of the results.

1 Imitation Learning

Imitation learning is a subset of machine learning where models are trained to mimic expert behavior in specific tasks, rather than learning from scratch through trial and error. This approach leverages expert-generated data to guide the learning process, aiming to quickly and efficiently achieve expert-level performance. In this section I use imitation learning to train policy networks—specifically, a Convolutional Neural Network (CNN) and a Multi-Layer Perceptron (MLP)—to accurately perform actions in a simulated driving environment. Lastly, I will analyze how different architectures and training parameters (such as batch size and number of epochs) affect the ability of neural networks to generalize from expert demonstrations to new, unseen scenarios.

1.1 Exploring Different Architectures

To test the impact of using different policy networks on the model performance, I used a CNN and an MLP policy network, in a simulated driving environment. Both models were trained using a Cross-Entropy Loss function, optimized by an Adam optimizer. Both models were trained with a batch size of 256 and over 20 epochs to assess general learning trends and stability, I also plot the training curves for both networks for visualization below.

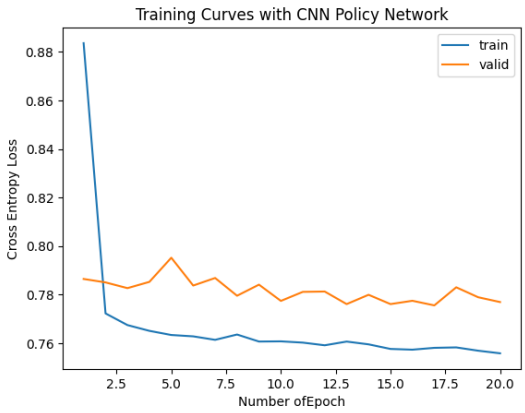

Figure 1: Training with CNN Policy Network

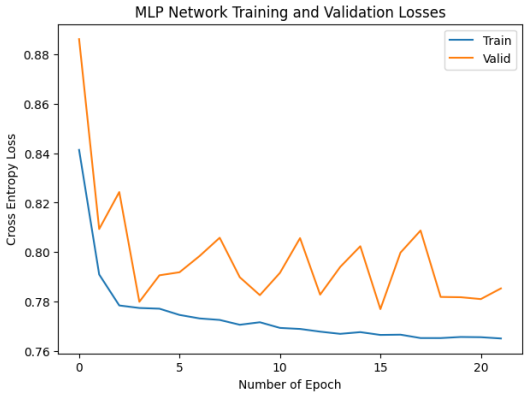

Figure 2: Training with MLP Policy Network

The two graphs above show the training curves for the CNN and MLP respectively. In the first graph (CNN), the training loss decreases more smoothly and stabilizes earlier around epoch 7, maintaining a consistently lower loss compared to the validation loss, which suggests that the CNN is effectively

learning and generalizing from the training data. The relatively smooth decrease in loss and lower variance between epochs indicate that the CNN architecture is well-suited to processing the spatial hierarchical features typical in image or video-based datasets like the driving environment.

In contrast, the second graph (MLP) shows a more volatile loss pattern, especially in the validation loss, which has sharp increases and decreases, peaking significantly around epoch 5 and again showing an upward trend after epoch 15. This volatility can be indicative of overfitting, where the MLP network, which lacks the spatial feature extraction capabilities of a CNN, might be memorizing the training data rather than learning to generalize. The higher variability in the MLP's performance across epochs also suggests that the MLP may struggle with the complexity or dimensionality of the input data compared to the CNN, potentially requiring adjustments in network architecture.

1.2 Hyperparameter Tuning

To explore the effects of more stochastic gradient estimates, I reduced the batch size to 64, and the training duration was shortened to 15 epoch. I made this adjustment to evaluate the impact of increased gradient noise and reduced training on model performance, the training curve for both networks is shown below:

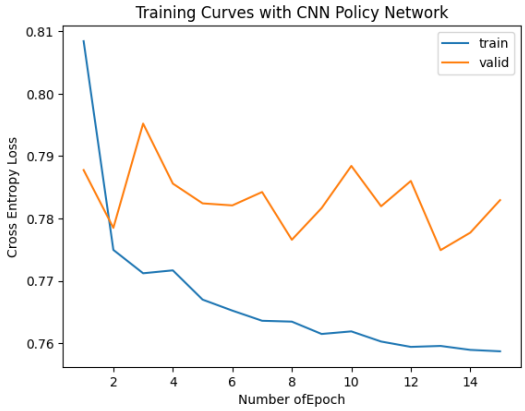

Figure 3: CNN Policy Network Batch Size 64

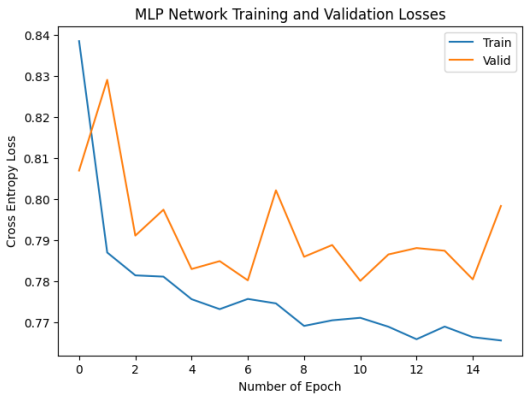

Figure 4: MLP Policy Network Batch Size 64

Reducing the batch size to 64 and the number of epochs to 15 has an impact on the training dynamics of both the CNN and MLP networks. The CNN Policy network shows a dramatic initial decrease in both training and validation loss, followed by significant spikes, particularly in the validation loss. This behavior suggests that the CNN is sensitive to batch size reduction. On the other hand, in the MLP policy network, the smaller batch size appears to have introduced more variability in the validation loss, which might be due to the increased noise in the gradient estimates because smaller batches often lead to less stable, but potentially more explorative updates. Reducing the epoch generally results in less training overall, which might not be optimal for networks requiring more extensive training to generalize well from complex or high-dimensional input spaces.

1.3 Dataset Aggregation (DAgger)

Dataset Aggregation, or DAgger, is a powerful technique in imitation learning that addresses the limitations of traditional supervised learning methods by iteratively refining the training dataset.

In the above example, our model was trained only on the expert-generated data, which can lead to significant discrepancies when the model encounters scenarios that were not represented in the training data. This results in poor generalization.

DAgger enhances this process by allowing the model to interact with the environment and generate its own data, which is then augmented with the expert's corrections. This iterative process involves the following steps:

1. **Initial Training**: Train the model on the initial dataset of expert demonstrations.
2. **Policy Execution**: Let the trained model interact with the environment to collect new states.
3. **Expert Correction**: At each new state encountered by the model, query the expert to provide the correct action, thereby creating a new dataset.
4. **Dataset Aggregation**: Combine the newly collected data with the existing dataset.
5. **Retraining**: Retrain the model on the aggregated dataset.

By iteratively applying these steps, DAgger ensures that the model is exposed to a broader range of scenarios, including those that are challenging or uncommon. This leads to improving the generalization as well as reducing the covariate shift which lead to increase performance.

I applied the DAgger algorithm to the CNN policy network, comparing their performance with and without DAgger. Below, I detail the impact of DAgger on the training dynamics and performance of these models.

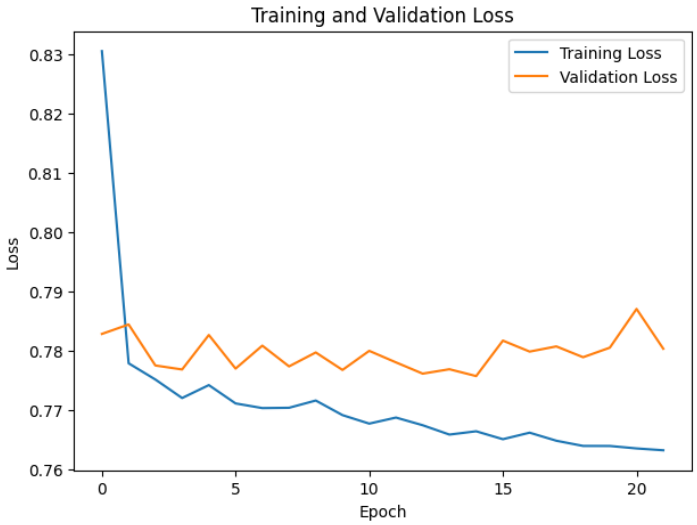

Figure 5: Training Curve using Dagger

The graphs above show the training curves for the CNN network when trained with DAgger. After the training, I compared the scores achieved in both cases.

1.4 Performance Evaluation

The training loss for the CNN decreases more consistently and stabilizes at a lower value compared to training without DAgger. The validation loss also shows a slight reduction in volatility, indicating

better generalization to unseen scenarios. The CNN, benefiting from DAgger, effectively captures and learns from a diverse set of driving situations.

To evaluate the performance of the CNN policy network with and without DAgger, I ran the network for 10 episodes in the simulated driving environment. The results are as follows:

The results indicate that using DAgger improves the mean score of the CNN policy network from 632.46 to 674.72. This improvement demonstrates that the model trained with DAgger is better at generalizing and performing in the simulated driving environment. The increase in the standard deviation from 177.95 to 264.07 suggests that while the model performs better on average, there is more variability in its performance. This variability might be due to the increased complexity and diversity of the training data collected during the DAgger process, which includes more challenging scenarios.

1.5 Summary of Results

In the training of both CNN and MLP networks, altering the batch size and epoch count led to notable differences in performance and learning stability. Initially, with a larger batch size of 256 and more extensive training over 20 epochs, the CNN demonstrated a smoother and more stable decline in training loss with less overfitting, while the MLP displayed greater fluctuation and potential overfitting, suggesting challenges in generalization. When the batch size was reduced to 64 and epochs limited to 15, both models experienced increased variability in their validation losses, with the MLP showing particularly severe spikes and instability in performance. This reduction in batch size and training duration introduced greater noise into the learning process, affecting the stability and effectiveness of the MLP more markedly than the CNN, potentially due to its simpler, non-convolutional structure's lower capacity for handling the complexity and variability of the input data. The application of DAgger further improved the performance of the CNN policy network. With DAgger, the mean score increased from 632.46 to 674.72, indicating better generalization and performance in the simulated driving environment.

2 Deep Q-Network

Deep Q-Network (DQN) is an advanced reinforcement learning algorithm that combines Q-learning with deep neural networks to enable agents to learn optimal policies directly from high-dimensional inputs. In this section, I make use of a DQN with two distinct policy network architectures, the MLP and CNN in a mini-grid environment to understand how different network designs impact model performance in the environment. I later adjusted some hyperparameters such as epochs and learning rates, to see how they also affect the model performance.

In the other section of the DQN, I tried a different environment (pong environment) and tried to tweak some hyperparameters to see how the average reward and loss behave.

2.1 Exploring Different Architectures

1 - DQN in a MiniGrid Environment with MLP & CNN Policy Network

In the training procedure for the Deep Q-Network (DQN), I used different policy networks (MLP and CNN), to evaluate their performance in the minigrid environment. I set both networks to train with a learning rate of 0.0001 across 1000 epochs, the result is shown below:

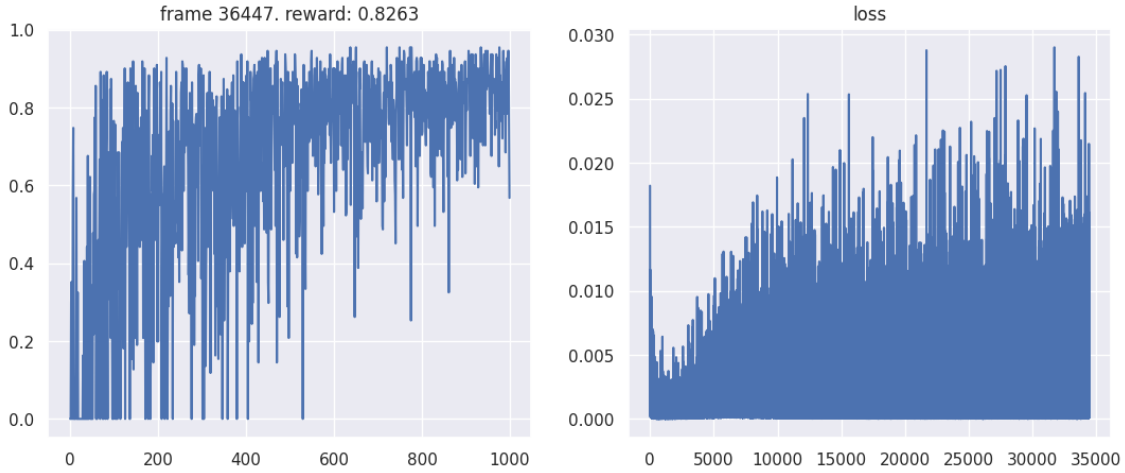

Figure 6: MLP Network (minigrid) - lr = 0.0001, episode = 1000

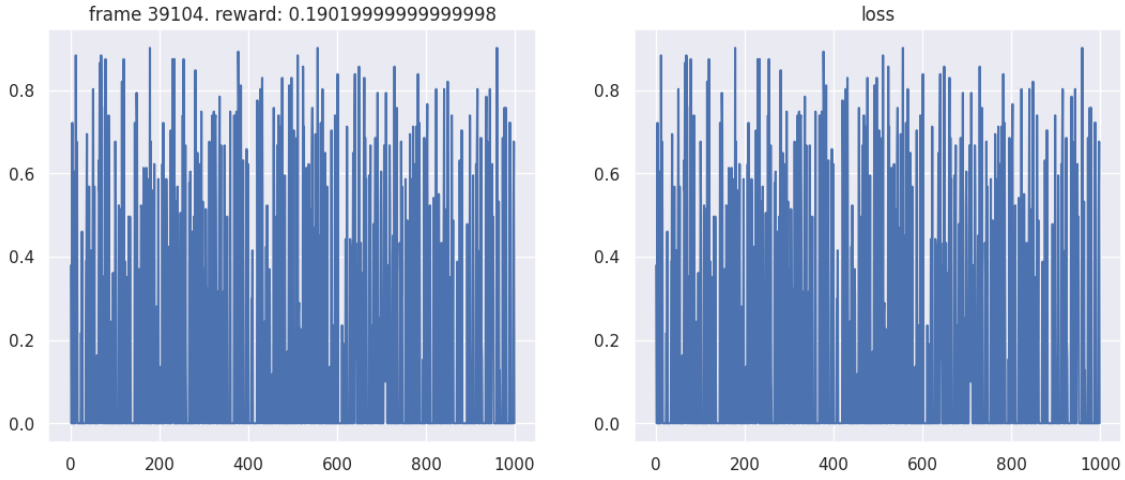

Figure 7: CNN Network (minigrid) - lr = 0.0001, episode = 1000

The results above indicate that the MLP policy network outperforms the CNN in a Deep Q-Network setting. Specifically, the MLP in the Minigrid environment shows a higher overall reward, averaging around 0.82, with relatively low loss peaking at approximately 0.029. This suggests that the MLP, despite its simpler architecture, effectively captures the dynamics of the Minigrid environment, leading to stable and high performance.

On the other hand, the CNN policy has significantly lower reward levels with an average of about 0.19 and higher loss values up to 0.9 when trained with the same hyperparameters as the MLP policy. This indicates challenges in CNN's ability to generalize or optimize effectively in this particular setting. The higher loss and lower rewards suggest that the CNN may not be extracting or leveraging spatial

and temporal features as effectively as needed for this environment.

2.2 Hyperparameter Tuning - MiniGrid Environment

In the training procedure for the Deep Q-Network (DQN), I try to change some hyperparameters to see how the average reward and loss changes, I reduce the number of episodes from 1000 to 500, the result is shown below:

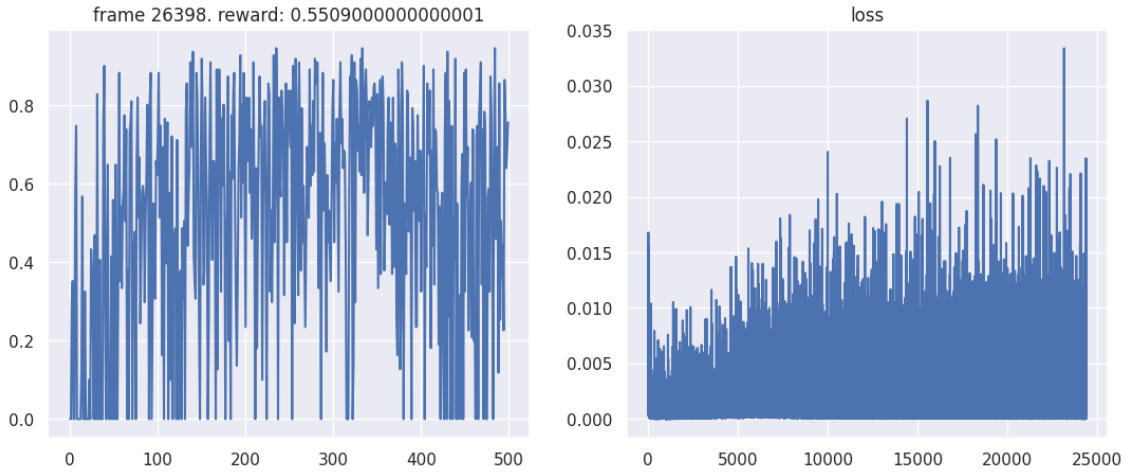

Figure 8: MLP Network (minigrid) - lr = 0.0001, episode = 500

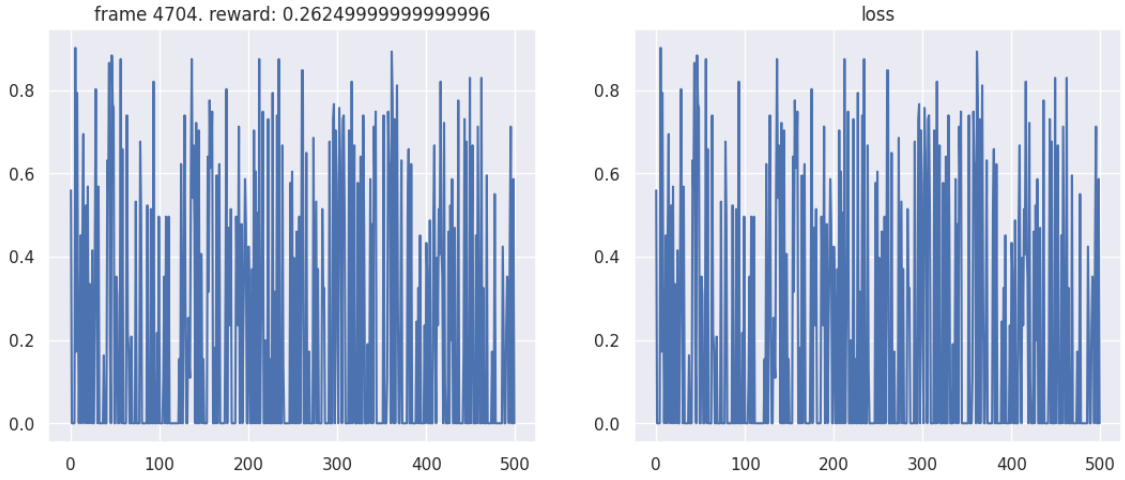

Figure 9: CNN Network (minigrid) - lr = 0.0001, episode = 500

From the above results shows that reducing the number of epochs to 500 while maintaining a learning rate of 0.0001 yielded interesting outcomes. The MLP network, as depicted in figure 7 outperforms the CNN network shown in figure 8 with an average reward of approximately 0.55 and a loss peaking at 0.035, the MLP network demonstrates superior performance compared to the CNN's average reward of 0.262 and a significant higher loss of 0.95. This superiority can be attributed to the MLP's ability to effectively model the decision-making process in less complex or more structured environments like MiniGrid where the data does not heavily rely on spatial hierarchies that benefit from convolutional layers. The CNN, while typically advantageous for spatial data processing, may not align as well with

the characteristics or scale of the MiniGrid environment, leading to less effective learning and higher error rates. Thus, the simpler, more direct connections of the MLP seem to capture the essential patterns more effectively in this scenario, leading to better performance metrics.

2.3 Hyperparameter Tuning - Pong Environment

For the DQN in a pong environment, due to GPU limitation, I could not train the model for many episodes, With that said I tried two different hyperparameters to see how it affects the reward and the loss. In the first run, I set the number of episodes to 200 and in the second run I increased it to 400, the results are shown below:

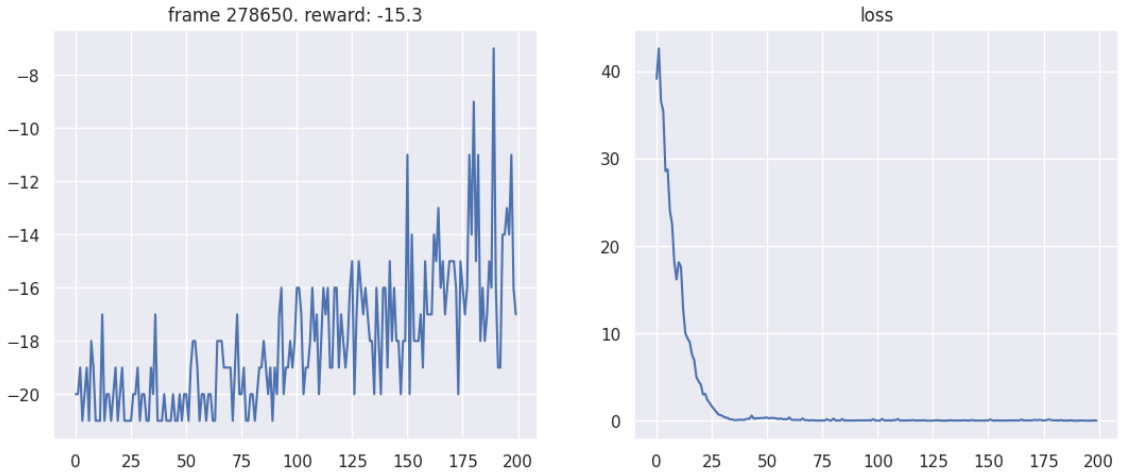

Figure 10: DQN in Pong Environment - episode = 200

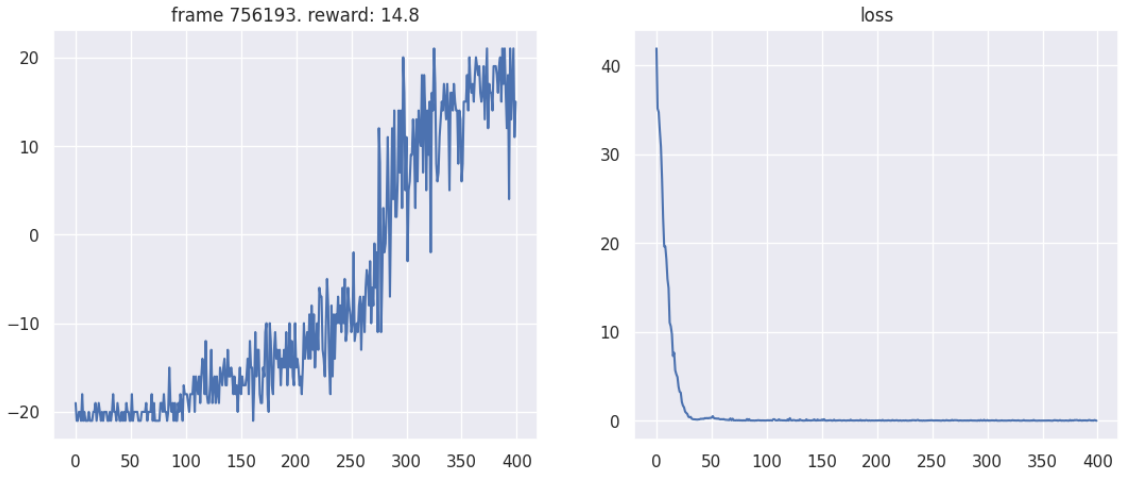

Figure 11: DQN in Pong Environment - episode = 400

The two sets of results reveal an improvement in performance as the number of episodes increases. Initially, over 200 episodes, the model yielded an average return of -15.3, which indicates a learning phase where the agent is still adapting and refining its policy based on the received rewards and penalties. With an extended training period to 400 episodes, the average return improved to 14.8 This progression underscores a gradual but consistent learning curve where the agent, through increased

interaction with the environment, optimizes its decision-making strategies to achieve higher returns. The reduction in loss over time further corroborates the effectiveness of the training, reflecting a more stable and accurate approximation of the action-value function as the agent learns to predict and enact actions that lead to less penalized outcomes in the game.

2.4 Summary of Results

The result from the minigrid environment shows that the higher the number of episode the better the average reward the agent gets and the lower the loss. In Pong Deep Q-Network experiments, the agent's performance improved over time, from an average reward of -15.3 after 200 episodes to 14.8 after 400 episodes, indicating adaptive learning and enhanced decision-making. However, the results also highlight the need for additional training and possible algorithmic or architectural adjustments to achieve more stable and consistently higher rewards.

3 Policy Gradients - PPO

Policy Gradient methods, particularly Proximal Policy Optimization (PPO), optimizes a policy directly and is recognized for its stability and reliability in training by optimizing the policy's parameters in a way that maximizes expected returns. This section focuses on utilizing PPO to study the impact of various hyperparameters such as episode count and weight decay on the learning performance of an agent within simulated environments. By examining changes in episode rewards, lengths, returns, and entropy under different configurations.

3.1 Hyperparameter Tuning - PPO

In my training procedure, I vary the number of episodes and weight decay parameters from their default values to observe their impact on the agent's learning dynamics. The training involved running the agent through a set number of episodes, during which we collected data on episode rewards, lengths, returns, and entropy to evaluate performance and the effectiveness of exploration strategies. I experimented with different settings—initially 1000 episodes with a weight decay of 0.001, then modifying it to 0.01 for 500 episodes, and finally reverting to the default decay rate for 50 episodes—to determine the optimal configuration for stable learning and improved returns.

3.1.1 PPO in BipedalWalker-v3 - episode = 1000 - weight decay = 0.001

The first run was on the default hyperparameter values which are, episode number = 1000 and weight decay = 0.001, the results of the simulation is show in the figure below:

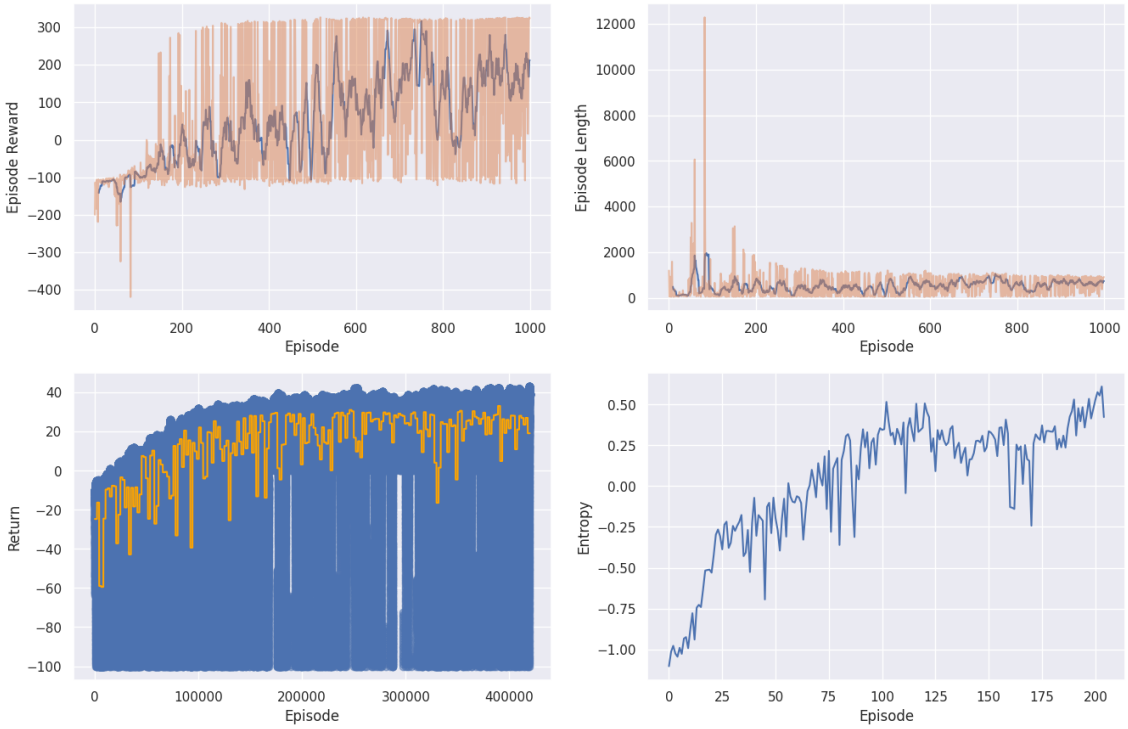

Figure 12: PPO in BipedalWalker-v3 - episode = 1000 - weight decay = 0.001

The results of the Proximal Policy Optimization (PPO) algorithm with an actor-critic setup illustrate several key aspects of the model's performance over 1000 episodes. The graphs show episode reward, episode length, returns, and entropy, each providing insight into different dimensions of the learning process. Initially, both the reward and episode length exhibit significant fluctuations, suggesting initial exploration and instability in policy performance. As training progresses, rewards begin to stabilize, indicating that the agent is learning to maximize returns more consistently within the environment. This is supported by the generally increasing trend in the returns graph, which signifies accumulating rewards over time. The entropy graph shows a gradual increase, reflecting the agent's continuous exploration of the action space to optimize its policy further. This increase in entropy, together with stable or improving rewards (max 324.5666), suggests a balance between exploration and exploitation.

3.1.2 PPO in BipedalWalker-v3 - episode = 500 - weight decay = 0.01

On the second run, I changed the hyperparameter values from the default ones, reduced the number of episodes from 1000 to 500, and also reduced the weight decay from 0.001 to 0.01, the results of the simulation is shown in the figure below:

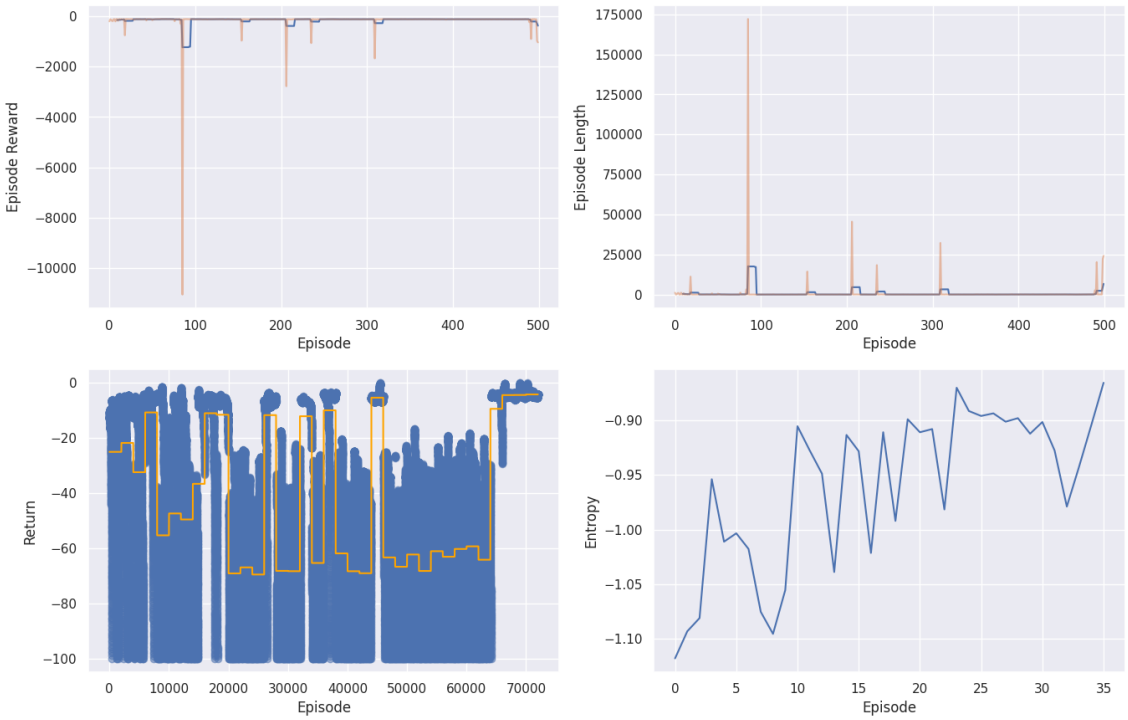

Figure 13: PPO in BipedalWalker-v3 - episode = 500 - weight decay = 0.01

The modified hyperparameters (increasing the weight decay from 0.001 to 0.01 and reducing the number of episodes to 500) have led to a noticeable impact on the model's performance. Compared to the previous results, the most noticeable differences are observed in the episode rewards and lengths. The new setup resulted in substantially lower episode rewards, with values reaching as low as -10,000, alongside a spike in episode length, occasionally surpassing 150,000 steps. These changes suggest that the increased weight decay might be penalizing the weights of the neural network, leading to a conservative policy that fails to take sufficient risks for higher rewards. This conservative behavior likely causes the agent to sustain episodes without achieving successful termination criteria, reflected in the extended episode lengths. Furthermore, the entropy shows an increasing trend over the episodes, indicating a higher level of exploration, which could be the system's response to the lack of effective policy consolidation due to high weight decay, preventing the agent from settling on a stable strategy.

3.1.3 PPO in BipedalWalker-v3 - episode = 50 - weight decay = 0.001

On the third run, having interested in how the episode length affects the results, I set the weight decay back to 0.001 and reduced the episode to a much smaller number which is 50, the results of the simulation are shown in the figure below:

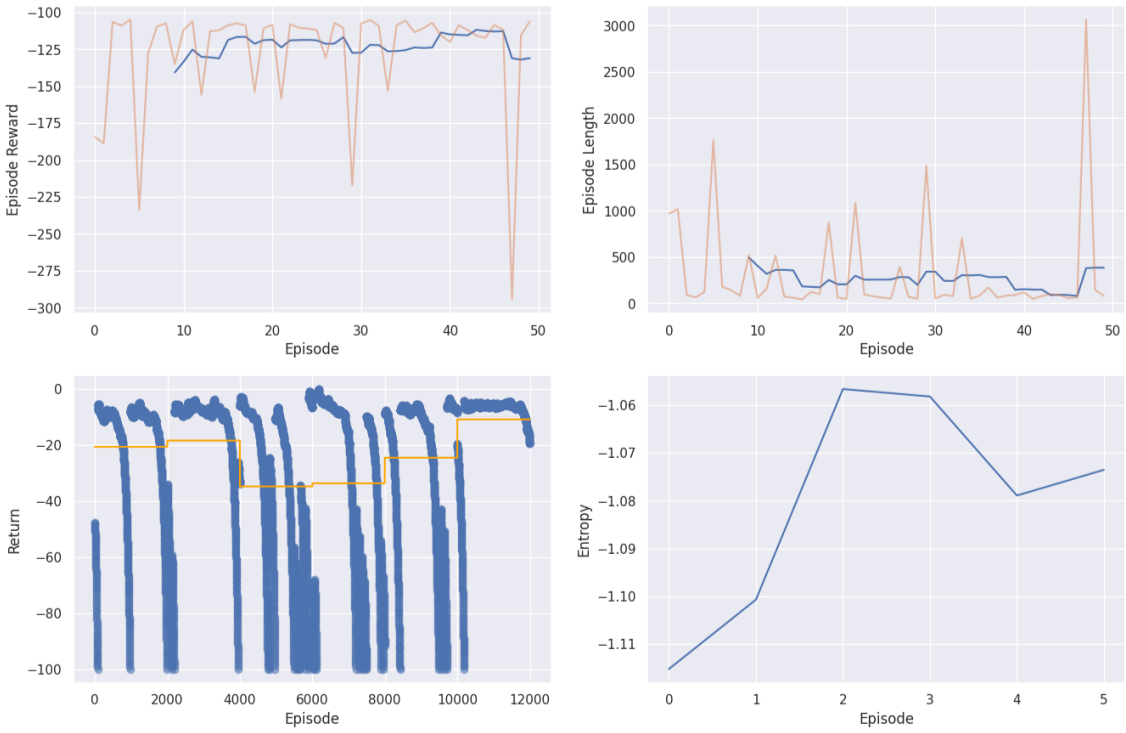

Figure 14: PPO in BipedalWalker-v3 - episode = 50 - weight decay = 0.001

Returning to the default weight decay setting of 0.001 for a shorter 50-episode run shows a distinct improvement in both the stability and performance of the reinforcement learning model compared to the previous runs with higher weight decay. The episode rewards in this configuration remain more consistent and less negative than with a higher weight decay. The episode lengths are generally shorter and exhibit less variance, indicating that the agent is likely finding efficient solutions faster and with less deviation in strategy across episodes. The return plot stabilizes around -20, unlike the severe losses seen previously, suggesting that the agent is maintaining a relatively consistent performance throughout the training. Similarly, the entropy plot shows a sharp rise initially followed by a plateau, which suggests early exploration followed by strategy refinement. This behavior shows the effectiveness of a lower weight decay in allowing enough flexibility for the agent to adjust and optimize its policy without prematurely converging to suboptimal actions, a problem exacerbated by higher weight decay rates.

3.2 Summary of Results

By testing the Proximal Policy Optimization (PPO) in various configurations, I observed significant differences in performance based on the selected hyperparameters. The analysis revealed that a lower weight decay of 0.001 and a larger number of episodes consistently produced better outcomes, characterized by more stable and higher episode rewards and longer episode lengths. Notably, increasing the weight decay to 0.01 resulted in lower rewards and longer episode lengths, indicating suboptimal policy performance and efficiency. Furthermore, the entropy trends highlighted the agent's exploration behavior under different settings, showing more controlled exploration with lower weight decay.