Double DQN for Overestimation Reduction in Games Final Report

Akhil Avula
ID: 405429867
UCLA
aavula@g.ucla.edu

Calvin Chang
ID: 805430067
UCLA
calvinchang33@g.ucla.edu

Daniel Truong
ID: 505426463
UCLA
dktruong@g.ucla.edu

Abstract

Hasselt et al. [2] has shown why Q-learning methods, such as Deep Q-Learning (DQN), suffer from overestimation. This project demonstrates why Double-DQN (DDQN) is superior in alleviating overestimation and in performance for game environments like Atari. The Reinforcement Learning (RL) algorithms developed in this project demonstrate mastery over Hasselt's paper by re-implementing a tuned version of DQN and DDQN for new environments: Classic Control, Box2D and Atari. This paper further investigates how DQN and DDQN play different roles in overestimation. Results are reproduced that show DDQN empirically has more stability in training which can generally lead to convergence to a good policy faster than DQN.

1 Introduction

1.1 Background

Value-based reinforcement learning is a method used to find policies that optimize the cumulative future reward in a Markov Decision Process (Sutton and Barto [4]). To find the values for each possible state and action, it would be computationally hard since many interesting problems have an enormous amount of state action pairs. Mnih et al. [3] used a method called Deep Q Networks to leverage a neural network's universal function approximation to map states to their action values and achieved superhuman results on an Atari 2600 emulator. It uses an online network, which tries to minimize the loss between its estimation of the Q values against the estimation of the target network. The target network copies the online network at chosen intervals. However, Thrun and Schwartz [5] showed that the current Q learning algorithm [6] to create the target values overestimates the actual value of the state-action pair and shows how overestimation can lead to expected failure to obtain the optimal policy.

1.2 Overestimation Effects

Overestimation is a substantial flaw in the broad spectrum of Q-Learning methods. These overestimated results are from a positive bias that is introduced because Q-learning uses the maximum action value as an approximation for the maximum expected action value. The goal of this project is to adapt from "Deep Reinforcement Learning with Double Q-learning" from Hasselt et al. [2] which proposes a solution called "Double DQN" to prevent overestimating values. The methods from Hasselt et al. [2] report that Double DQN leads to better policies in video game environments than the DQN method. This project introduces variation in the algorithms presented in Hasselt et al. [2] to further investigate the issue of overestimation.

2 Methods

The first step in this project was setting up the environment. Using OpenAI gym and its wrapper classes, the Classic Control, Box2D and Atari environments were setup. Specifically for Atari environments, prepropssing was necessary and consisted of: stacking 4 frames together, re-scaling and gray-scaling the image, and other preprocessing techniques used in Mnih et al. [3]. This was followed by implementing a DQN and DDQN agent with Pytorch. The networks for both DQN and DDQN were implemented using class methods from Pytorch. Each environment/game has a separate file with a designated DQN and DDQN, where both were implemented using the same hyperparameters. Each agent was trained over a fixed amount of timesteps, varying for each environment. A link is given for a Google Colab folder that shows all of the Python notebooks used to implement the networks, agents and algorithms (Appendix C). Google Colab was used to implement the algorithms using a GPU accelerator for improving runtime.

2.1 Pseudo-code

Below is the pseudo-code for a vanilla Double DQN algorithm from Hasselt et al. [2] which is the basis of the project.

Algorithm 1 Double DQN

```
1:  input :$D-$ empty replay buffer; $\theta-$ online network params, $\theta^-$ - target network params
2:  input :$N_r-$ replay buffer maximum size; $N_b-$training batch size; $N^- -$ target network replacement frequency
3:  FOR $episode \in \{1, 2, ..., M\}$
4:      Initialize frame sequence $\text{X} \leftarrow ()$
5:      FOR $t \in \{1, 2, ...\}$
6:          Set state $s \leftarrow x$, sample action $a\ \pi_B$
7:          Sample next frame $x^t$ from environment $\xi$ given $(s, a)$ and receive reward, $r$, and append $x^t$ to X
8:          if $|\mathbf{X}| > N_f$ then delete oldest frame $x_{t_{min}}$ from X end
9:          Set $s' \leftarrow \text{X}$, and add transition tuple $(s, a, r, s')$ to $D$, replacing the oldest tuple if $|D| \geq N_r$
10:         Sample a minibatch of $N_b$ tubles $(s, a, r, s')$ Unif$(D)$
11:         Construct target values, one for each of the $N_b$ tuples:
12:         Define $a_{max}(s'; \theta) = \max_a Q(s', a'; \theta)$
13:         $y_j = \left\{ \begin{array}{ll} r, & \text{if } s' \text{ is terminal} \\ r + \gamma Q(s', a^{max}(s'; \theta); \theta^-), & \text{otherwise} \end{array} \right\}$
14:         Do a gradient descent step with loss $||y_j - Q(s, a; \theta)||^2$
15:         Replace target parameters $\theta^- \leftarrow \theta$ every $N^-$ steps
[1]
```

The only difference between DQN and DDQN is the calculation of the temporal difference target:

$$Y_t^{DQN} = R_t + Q(s', \theta^-)$$
$$Y_t^{DDQN} = R_t + Q(s', max_a Q(s'; \theta), \theta^-)$$

This incorporates Double Q Learning [1] into the DQN method to create DDQN.

2.2 Neural Network Architectures

For creating the algorithms, a neural network for each game needs to be made. Figure 1 shows the the type of layers that were used for each game.

The neural network architecture for Pong is the same as the one specified in Volodymyr Mnih's Deep Reinforcement learning paper [3]. Convolutional layers were used in Pong as the observation space of this game is an image. Convolutional layers thus are able to preprocess the inputs. C2D(H,W,F)

Game	Pong	Lunar Lander	Cartpole	Acrobot
Layer 1	C2D(8,8,32); stride 4	FC(100)	FC(20)	FC(128)
Layer 2	C2D (4,4,64); stride 2	FC(100)	FC(20)	FC(128)
Layer 3	C2D (3,3,64); stride 1	FC(4)	FC(2)	FC(3)
Layer 4	FC(512)			
Layer 5	FC(6)			

Figure 1: Layers for Each Game's Neural Network

means a 2D convolutional layer with height H, width W, and number of filters F. For the other games, FC layers were used for processing the inputs. FC(X) means a fully connected (dense) layer with output vector size X. These networks were used as the basis for the DQN and DDQN algorithms.

3 Results

3.1 Evaluation Scores

Graphs were plotted for the reward per episode, loss, and max Q values for both algorithms on the environments Pong, Lunar Lander, Cartpole, and Acrobot. Figure 2 the evaluation rewards achieved for each of our models, averaging over 50 episodes. The percentage improvements are calculated as the following:

$$score_{normalized} = \frac{score_{agent} - w}{score_{DQN} - w}$$

where w is the lowest possible score for an episode in the respective environment. In general, the games improved with evaluation with each upgrade on DQN (Figure 2).

Average Evaluation Scores after Training

Game	DQN	DDQN	DDQN Improvement	Tuned DDQN	Tuned DDQN Improvement
Pong	-11.84	-7.26	150%	4.94	283%
Lunar Lander	-88.54	-44.21	N/A	-13.92	N/A
Cartpole	145.8	215.18	148%	481.16	330%
Acrobot	-96.02	-189.92	77%	-107.24	97%

Figure 2: DQN, DDQN and tuned DDQN agents evaluated for fixed number of timesteps after training. Evaluation plots are provided in Appendix B.

The results for the reward, loss, and Q values can be seen in Appendix A. In general, the reward and Q value increased with each time episode, while the loss decreased. For Pong, the evaluation score improved drastically with tuned DDQN. Lunar Lander does not have improvement percentages because the lowest possible score is $-\infty$. Cartpole showed the greatest change in results with DDQN and tuned hyperparameters. This is likely due to it being in a simpler environment and to being an easier game to learn. However, a slightly more complex game like Acrobot did not have improved drastically evaluation scores. For Acrobat, DQN gave better performance in rewards than DDQN and tuned DDQN.

3.2 DQN and DDQN Hyperparameters

When testing the algorithms, the hyperparameters in Figure 3 were used for DQN and DDQN. The hyperparameters can be listed off as number of timesteps trained, learning rate, batch size, epsilon end, epsilon decay, epsilon decay type, and target network update period. The same hyperparameters were applied to DQN and DDQN to measure the performance difference. Each of these hyperparameters have their own special role in contributing to the overall performance. Number of timesteps trained influences how long the agent can learn for. Learning rate determines the networks amount of weight updated during training. Batch size determines the subset of data or mini batch. Epsilon end determines the end point starting from 1, while epsilon decay determines the number of decays over that length. Epsilon decay type assigns a function or polynomial type that determines the rate of change. Finally the target network update period determines when the target parameters are updated.

Hyperparameters	Pong	Lunar Lander	Cartpole	Acrobot
Number of Timesteps Trained	350000	100000	10000	50000
Learning Rate	0.0001	0.001	0.001	0.001
Batch Size	32	64	64	64
Epsilon end	0.01	0.1	0.05	0.05
Epsilon Decay	80000	10000	300	5000
Epsilon Decay Type	Linear	Exponential	Exponential	Exponential
Target Network Update Period	1000	1000	10	500

Figure 3: Hyperparameters of DQN and DDQN for each Game

3.3 Tuned DDQN Hyperparameters

After testing the normal DQN and DDQN algorithms with the default hyperparameters, tuning was done for DDQN to improve evaluation. A manual grid search was used, resulting in an improved performance for the DDQN to show its potential over its non-tuned counterparts, DDQN and DQN (Figure 4).

Hyperparameters	Pong	Lunar Lander	Cartpole	Acrobot
Number of Timesteps Trained	350000	100001	10001	50000
Learning Rate	0.00025	0.001	0.001	0.003
Batch Size	32	32	32	64
Epsilon end	0.1	0.05	0.05	0.05
Epsilon Decay Length	50000	10000	200	1000
Epsilon Decay Type	Linear	Exponential	Linear	Exponential
Target Network Update Freq	3000	2000	100	5000

Figure 4: Hyperparameters of Tuned DDQN for each Game

4 Discussion

Overall the results from training and testing DDQN showed improvement in action estimation through rewards and Q values. The DDQN has been able to get better rewards and find better policies. For Pong, Lunar Lander, Cartpole, the tuned DDQN algorithm gave better rewards for each time episode than DQN. Acrobat proved to be an exception, likely due to its action space. The differences in DQN and DDQN were also shown through stability in training, action space, and their max Q values.

4.1 Stability in Training

A reason proposed by Hasselt et al. [2] on why overestimation can also lead to instability is that DQN and DDQN use temporal difference for error. The bootstrapping method allows the error to propagate to future action value estimates. However, like a snowballing effect, this propagation will eventually cause instability and is hard to stop once the error gets too large. The training plots in Appendix A can empirically confirm this notion as the instability drop in the reward happens only after a certain number of timesteps. For example, in the Cartpole environment, Figure 11 shows the reward per training episode shoot up to 250 at around episode 80 but then crashes back down. The snowballing effect of the error grew larger and larger causing the estimator to wrongly estimate many state-action pairs, ultimately leading to training failure. Another possible example of the snowballing effect can be seen in the DQN loss of Lunar Lander in Figure 15.

For DDQN, the increased stability in training allows the agent to train faster than DQN. DDQN outperformed DQN in the 3 out of 4 games tested even when both of the agents were only given a fixed number of timesteps to train. DQN is expected to eventually achieve a policy that achieves high reward, but the plots show that it will generally need a longer training phase to get there.

4.2 Size of Action Space

In some of games, DDQN may not always be more effective than DQN. Thrun and Schwartz [5] have proved that the expected overestimation error is upper bounded by

$$\gamma\epsilon\frac{m-1}{m+1}$$

where m is the number of actions in the action space. This can explain why DQN Acrobot in Figure 26a fared better than the DQN and DDQN. The size of the action space for the Acrobot environment is 3 and since it is a simple environment, it would not be surprising if the DQN had equal or better performance than the tuned DDQN. For the rest of the games, tuned DDQN was better suited because of their larger action space.

4.3 Max Q Values

In the evaluation stage, the max Q values suggest that a lower value approximation will lead to more reward. Although this is not concretely apparent in the evaluation plots shown in the Appendix, DDQN and tuned DDQN converge to value estimates roughly equal to or lower than DQN's estimates in all environments. Also, it is interesting to point out that in the control environments, the max Q values oscillate between two values because of the nature of the system it is interacting with.

5 Conclusion

From creating the networks, the results overall improved for each game with each version of DQN. Pytorch provided many of the tools needed to create the neural network, DQN, and DDQN. Other machine learning libraries such as TensorFlow and TensorForce could be considered for the future. Testing in different environments, like Cartpole was also a success. The Atari games also showed good evaluation scores and significant difference between DQN and DDQN, showing that DDQN finds better policies. After tuning DDQN's hyperparameters, there was an improvement in training stability for each game. However, depending on the size of the action space, the DDQN algorithm may not always yield better rewards.

Many ideas could be implemented to take the results and implementation of these algorithms further. In this project, a major limitation was hardware. Although Google Colab's cloud services and GPU proved successful, the DQN and DDQN would likely be able to perform better with a higher-end GPU. More complex games from different environments could also be used for testing. Each agent per game could be trained for a longer period of time, and the hyperparameters could be further tuned for better training stability. Overall this project provided a good implementation of Reinforcement Learning algorithms and how they can be used to reduce overestimations in gaming environments.

6 Contributions

Akhil - I implemented the DDQN algorithm and trained it with several games in the Atari environment. I created the plots for the rewards, loss and Q values for each game and analyzed the results. I also helped with tuning the hyperparameters, debugging, and retraining DDQN to yield better results.

Calvin - I did background research on the Atari environment and determined how to utilize OpenaAI Gym. I implemented the DQN algorithm and created the reward, loss, and Q value plots for each game. I helped with debugging the neural network and creating the different layers.

Daniel - I created the base neural network on Pytorch that was used for both DQN and DDQN. I also determined how to tune the hyperparameters for DDQN each game. I also trained and tested the

DQN and DDQN algorithms for the Cartpole environment.

References

[1] Hado V. Hasselt. Double q-learning. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2613–2621. Curran Associates, Inc., 2010. URL `http://papers.nips.cc/paper/3964-double-q-learning.pdf`.

[2] Hado V. Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015. URL `http://arxiv.org/abs/1509.06461`.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015. ISSN 1476-4687. doi: 10.1038/nature14236. URL `https://doi.org/10.1038/nature14236`.

[4] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

[5] Sebastian Thrun and Anton Schwartz. Issues in using function approximation for reinforcement learning. In Michael Mozer, Paul Smolensky, David Touretzky, Jeffrey Elman, and Andreas Weigend, editors, *Proceedings of the 1993 Connectionist Models Summer School*, pages 255–263. Lawrence Erlbaum, 1993. URL `http://www.ri.cmu.edu/pub_files/pub1/thrun_sebastian_1993_1/thrun_sebastian_1993_1.pdf`.

[6] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992698. URL `https://doi.org/10.1007/BF00992698`.

Appendix

A Training Plots

Each of the agents are trained with the same number of timesteps.

A.1 Pong

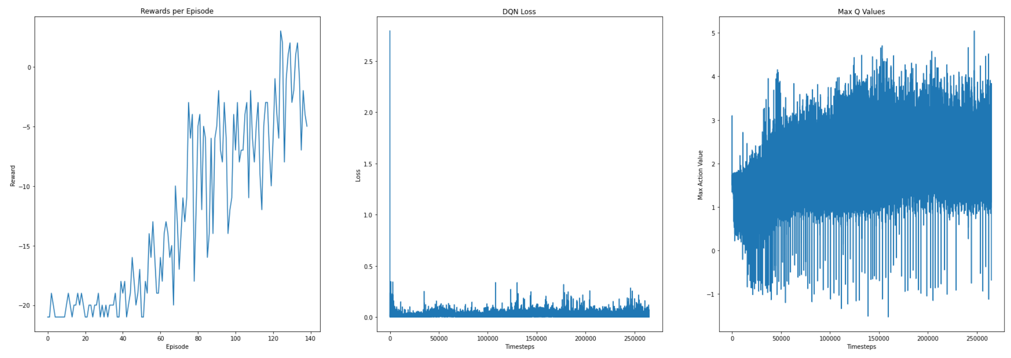


Figure 5: DQN on Pong Training phase

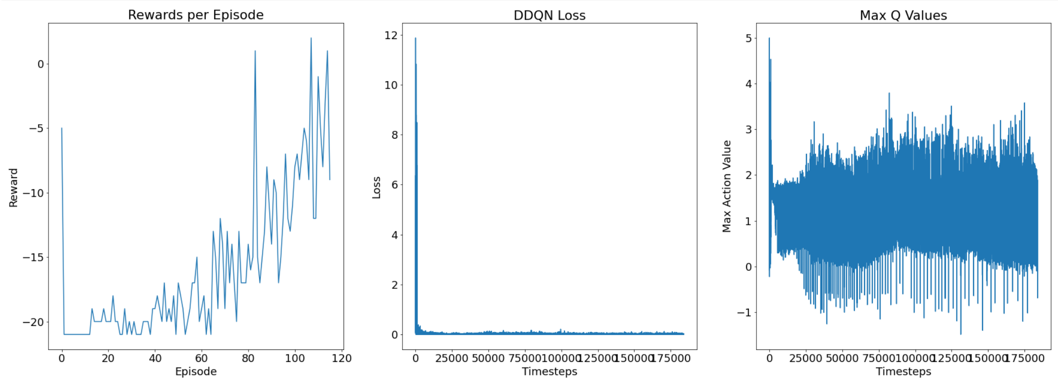


Figure 6: DDQN on Pong Training phase

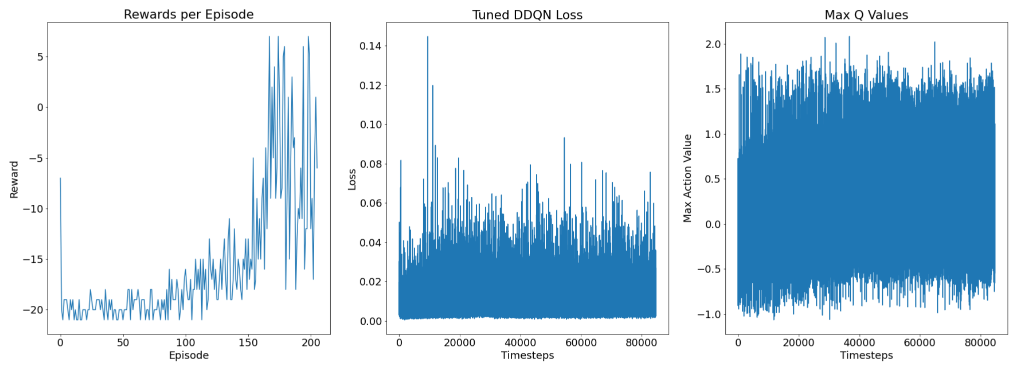


Figure 7: Tuned DDQN on Pong Training phase

A.2 Lunar Lander

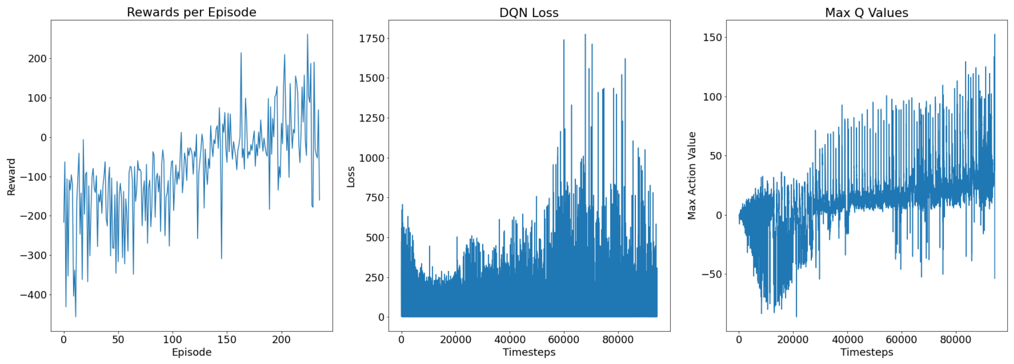


Figure 8: DQN on Lunar Lander Training phase

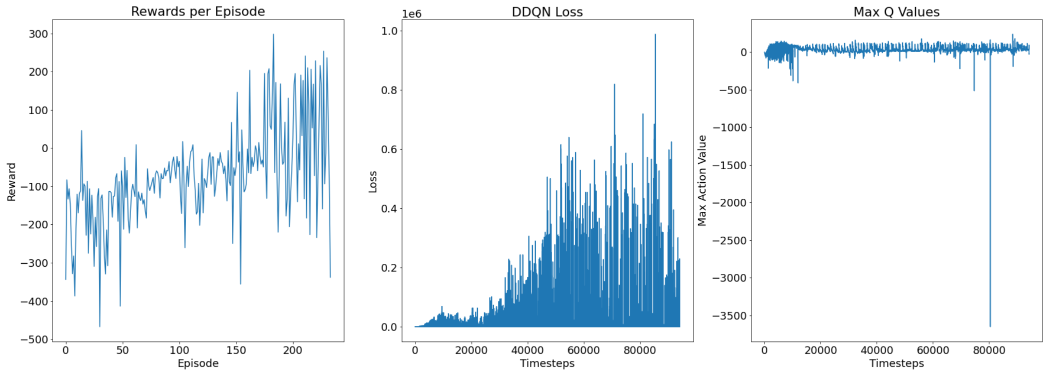


Figure 9: DDQN on Lunar Lander Training phase

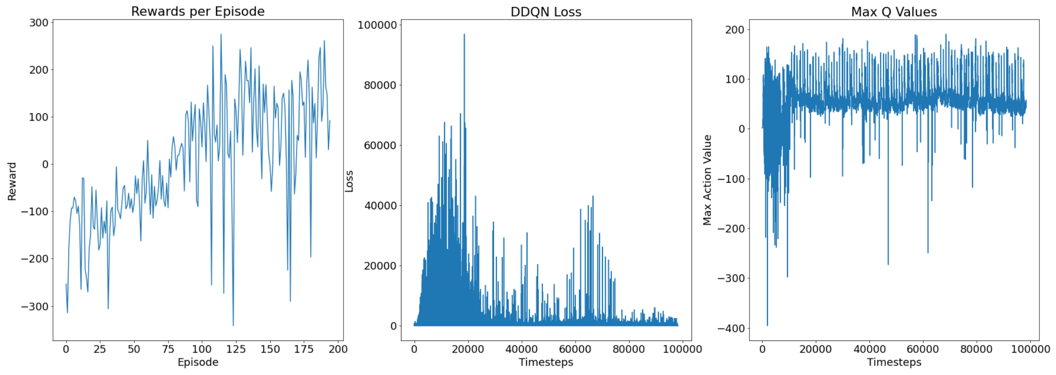
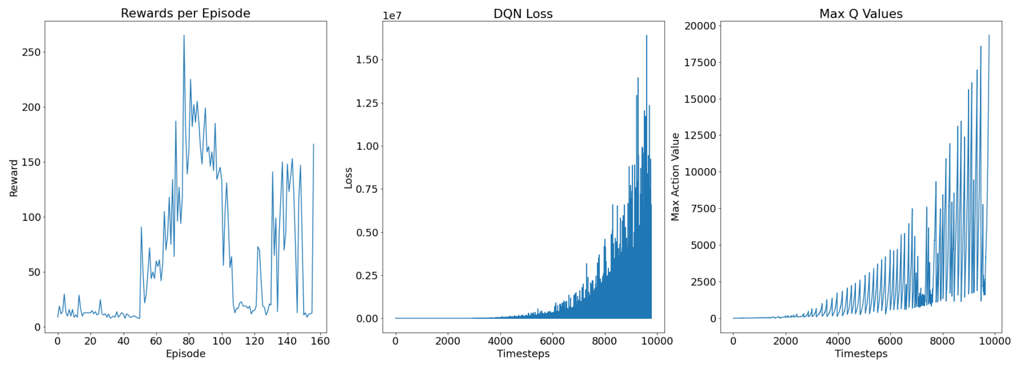


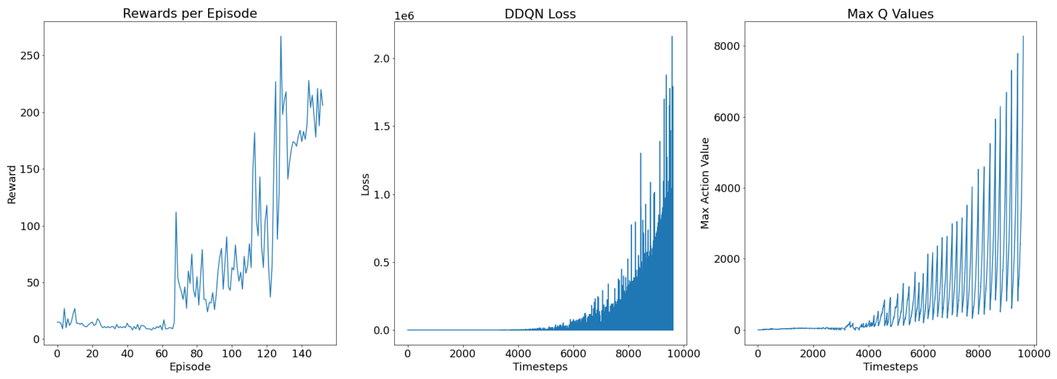
Figure 10: Tuned DDQN on Lunar Lander Training phase

A.3 Cartpole



```
Training reward mean: 63.59235668789809
```

Figure 11: DQN on Cartpole Training Phase



```
Training reward mean: 64.18300653594771
```

Figure 12: DDQN on Cartpole Training Phase

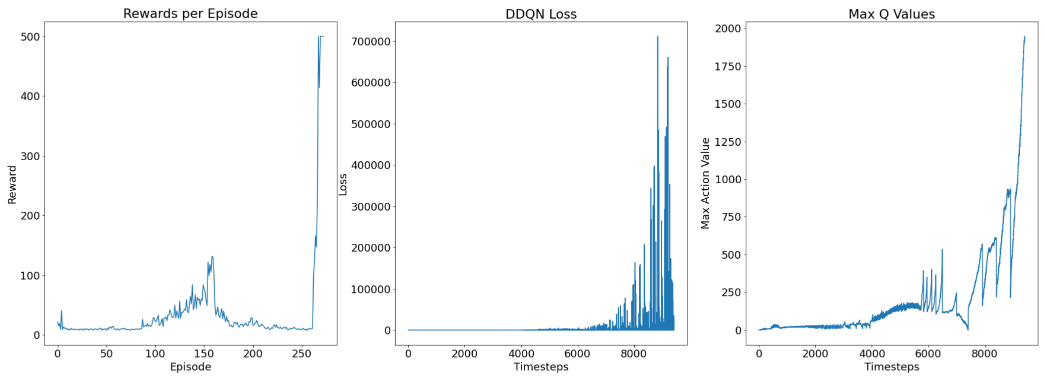


Figure 13: Tuned DDQN on Cartpole Training Phase

A.4 Acrobot

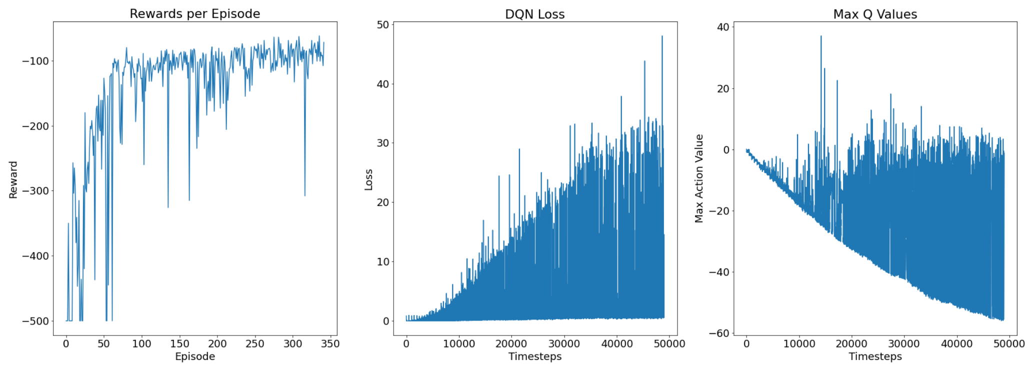


Figure 14: DQN on Acrobot Training Phase

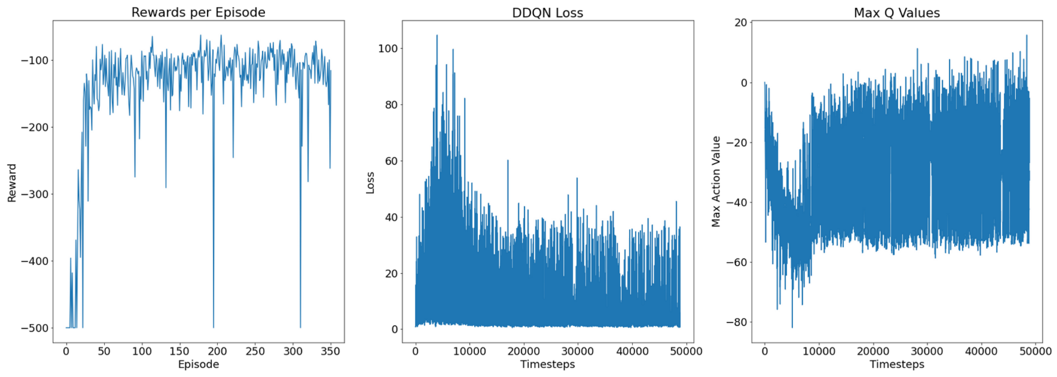


Figure 15: DDQN on Acrobot Training Phase

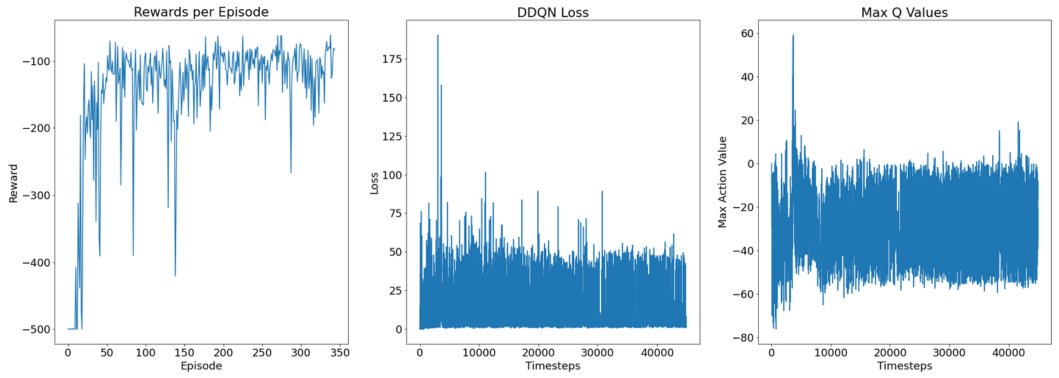
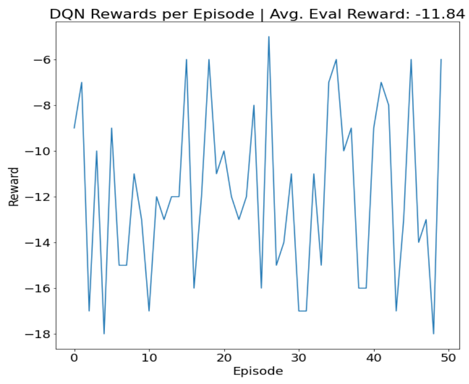


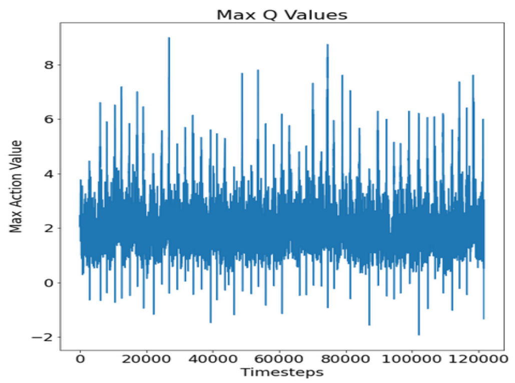
Figure 16: Tuned DDQN on Acrobot Training Phase

B Evaluation Plots

B.1 Pong

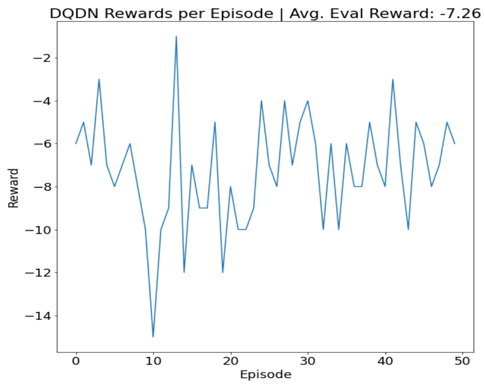


(a) DQN Evaluation Reward

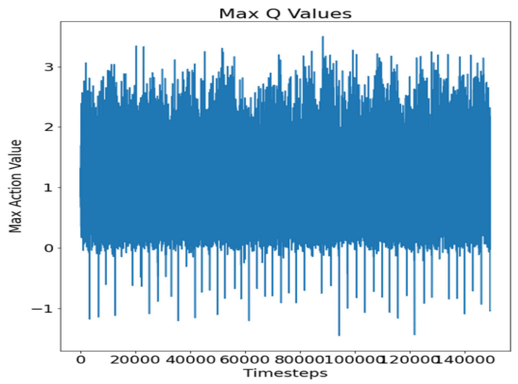


(b) DQN Max Q Values

Figure 17: DQN Evaluation Phase on Pong

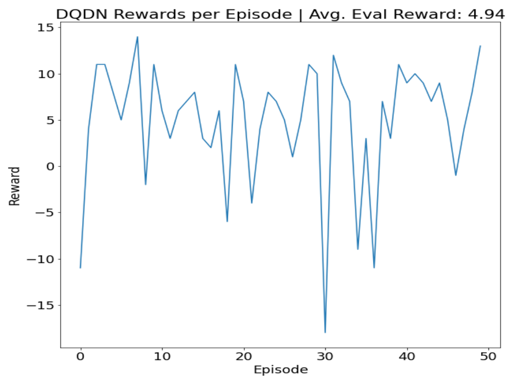


(a) DDQN Evaluation Reward

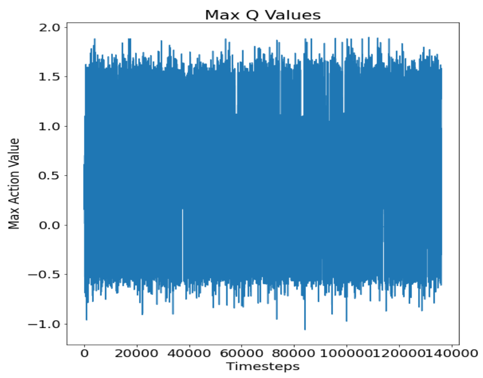


(b) DQN Max Q Values

Figure 18: DDQN Evaluation Phase on Pong



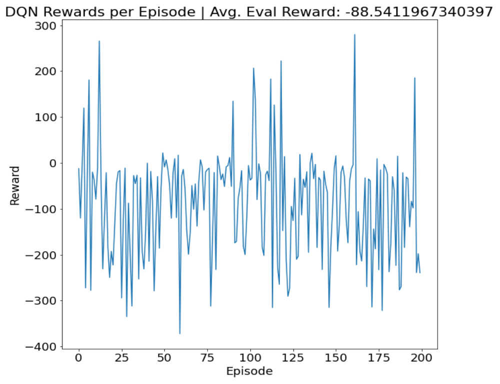
(a) Tuned DDQN Evaluation Reward



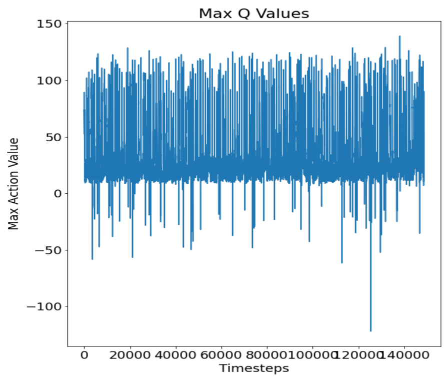
(b) Tuned DDQN Max Q Values

Figure 19: Tuned DDQN Evaluation Phase on Pong

B.2 Lunar Lander

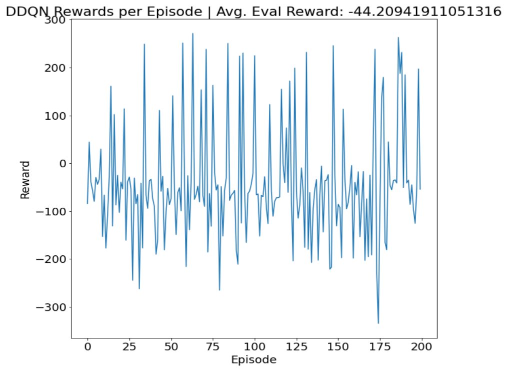


(a) DQN Evaluation Reward

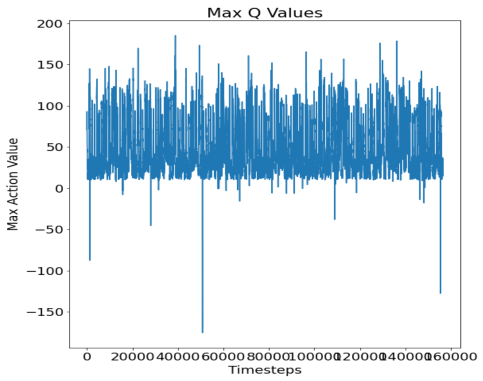


(b) DQN Max Q Values

Figure 20: DQN Evaluation Phase on Lunar Lander

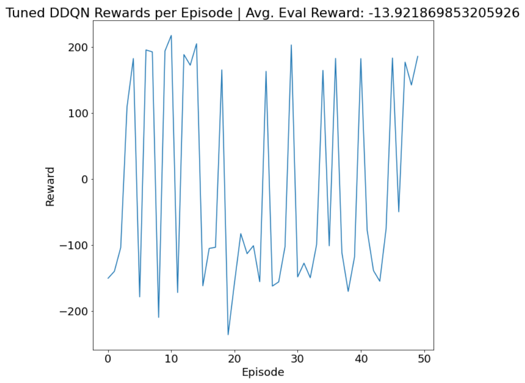


(a) DDQN Evaluation Reward

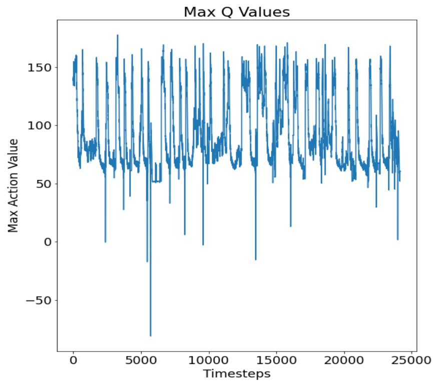


(b) DDQN Max Q Values

Figure 21: DDQN Evaluation Phase on Lunar Lander



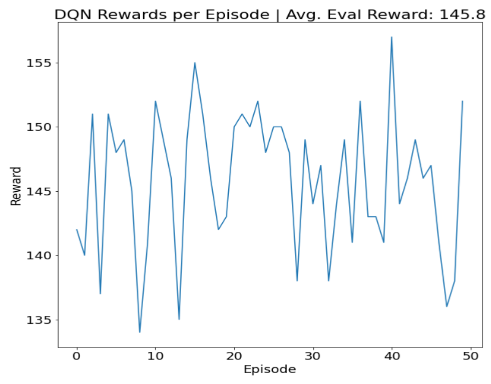
(a) Tuned DDQN Evaluation Reward



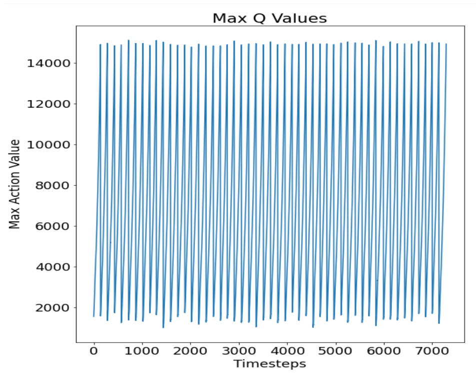
(b) Tuned DDQN Max Q Values

Figure 22: Tuned DDQN Evaluation Phase on Lunar Lander

B.3 Cartpole

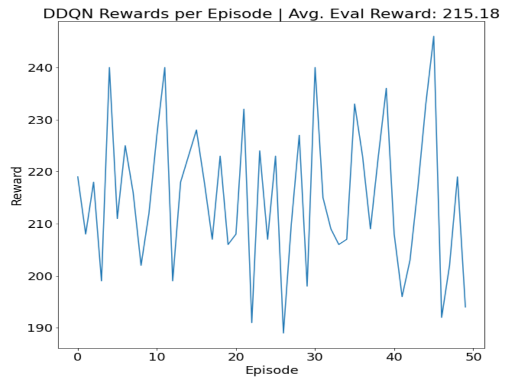


(a) DQN Evaluation Reward

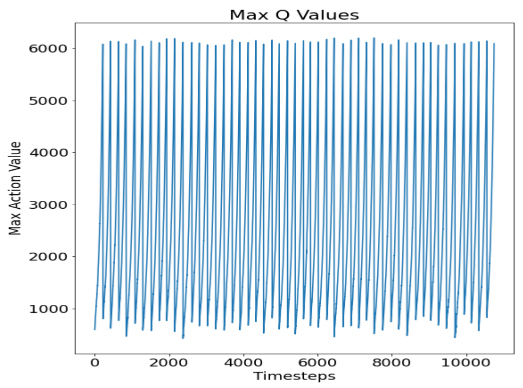


(b) DQN Max Q Values

Figure 23: DQN Evaluation Phase on Cartpole

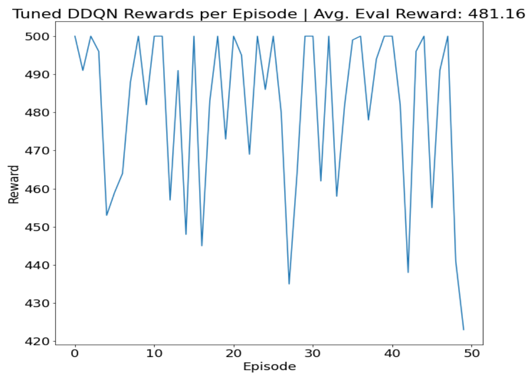


(a) DDQN Evaluation Reward

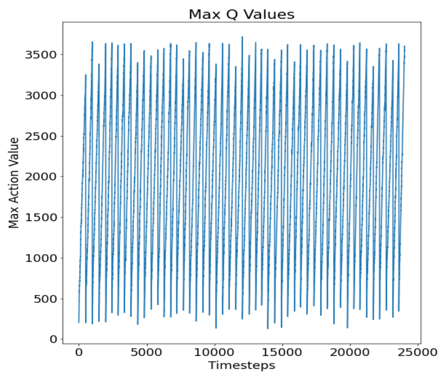


(b) DQN Max Q Values

Figure 24: DDQN Evaluation Phase on Cartpole



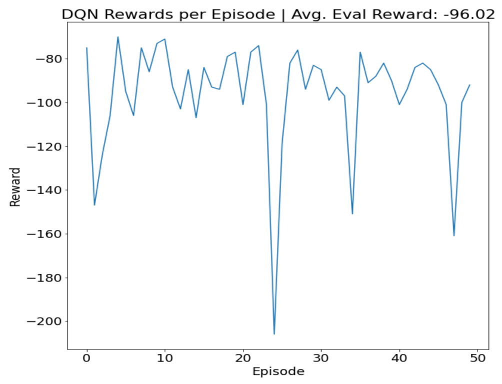
(a) Tuned DDQN Evaluation Reward



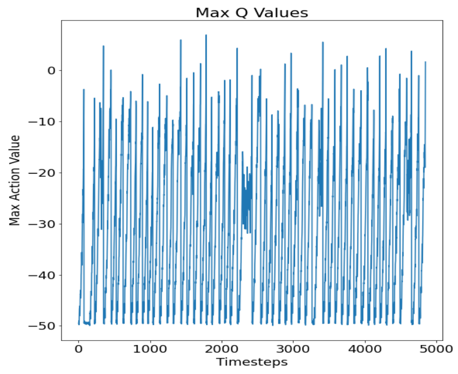
(b) Tuned DDQN Max Q Values

Figure 25: Tuned DDQN Evaluation Phase on Cartpole

B.4 Acrobot

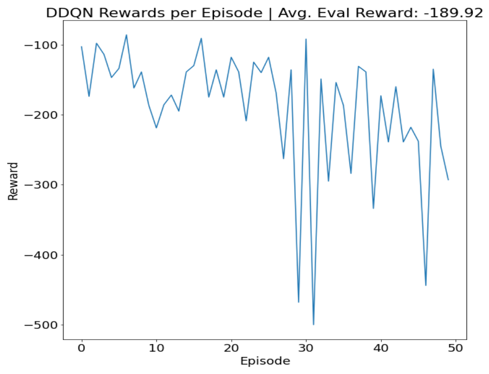


(a) DQN Evaluation Reward

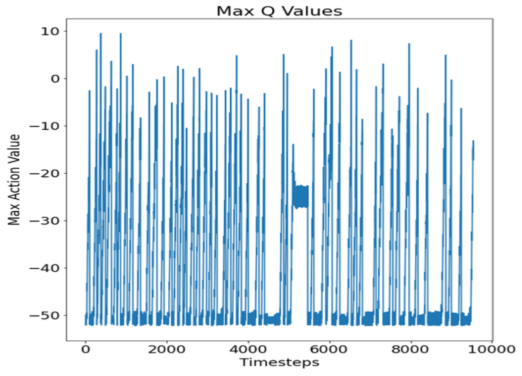


(b) DQN Max Q Values

Figure 26: DQN Evaluation Phase on Acrobot

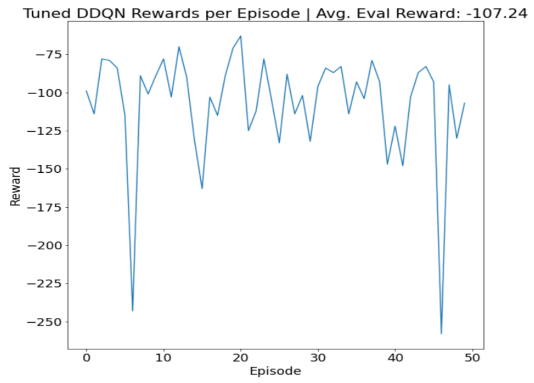


(a) DDQN Evaluation Reward

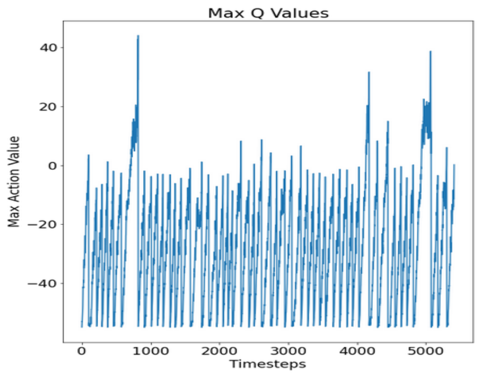


(b) DQN Max Q Values

Figure 27: DDQN Evaluation Phase on Acrobot



(a) Tuned DDQN Evaluation Reward



(b) Tuned DDQN Max Q Values

Figure 28: Tuned DDQN Evaluation Phase on Acrobot

C Python Code

The code is being done in Google Colab to take advantage of their free computational resources, namely GPU accelerator. Other games were also implemented, but not used for the results due to runtime issues.The following url is a link to the google drive: `https://tinyurl.com/y74jna9o`

The notebooks for the 4 games are also included as part of submission.