ECE5242 - INTELLIGENT AUTONOMOUS SYSTEMS

PROJECT 4 - REINFORCEMENT LEARNING

MAY 4, 2020

EVA ESTEBAN
CORNELL TECH

ABSTRACT

The aim of this report is to provide an in-depth analysis of the algorithms developed to implement Reinforcement Learning (RL) in different environments with both discrete and continous state spaces, and analyzing the performance of each algorithm for different hyperparameters. The algorithms include Policy Iteration, Q-Learning and REINFORCE with Baseline.

TABLE OF CONTENTS

1 INTRODUCTION

In the field of Machine Learning (ML), Reinforcement Learning (RL) [1] is the area that studies how to determine the actions that a software agent must take in different environments to maximize the cumulative rewards it obtains. It explores how to combine and balance the exploration of new areas and the use of the information the agent has about explored areas to achieve the best outcome.

In this project, algorithms were developed to implement several RL tasks in three different environments. These environments include a domain **Maze** with a discrete state space, and two environments with a continuous state space from the RL toolkit OpenAI Gym [2]: **Acrobot-v1** and **MountainCar-v0**. The algorithms Policy Iteration [3], Q-Learning [4] and REINFORCE with Baseline [5] were developed and thoroughly evaluated for different conditions and hyperparameters.

The project was divided into three main parts. The first part consisted of implementing the Policy Iteration algorithm to find both the optimal policy and the optimal Q values in the **Maze** domain. Using these results, the path describing the actions the robot will take at each timestamp assuming no slippage was obtained. The second part of the project consisted of implementing the Q-learning algorithm and applying it to the **Maze** domain. Several experiments and tests to determine the optimal learning rate and epsilon values, as well as to analyze the overall performance of the algorithm were conducted. The final part of the project includes the implementation of the REINFORCE function as well as the application of Q-learning in the above-mentioned OpenAI Gym environments. The performance of both algorithms was analyzed and compared by using learning curves.

2 PROBLEM FORMULATION

The problem tackled in this project is the application of RL algorithms to agents within different environments. The objective is to optimize the algorithms to enable the agent to obtain the maximum score. The data available to complete this challenge consists of a description of the specific conditions and goals to reach in each environment, and the base code for a function to evaluate the agent's performance.

3 TECHNICAL APPROACH

3.1 PART 1: POLICY ITERATION / VALUE ITERATION

The first step to complete this section of the project consisted of initializing an instance of the **Maze** environment. Using the conditions in this enviroment, the states of the agent were initialized as a 1D array of length 112, the positions were initialized as a dictionary with 14 keys each of which corresponds to an XY-coordinate value indicating the location of that position in the Maze, the flag states were initialized as a 1-D array of length 8 where each element corresponds to an index provided to the **num2flag()** function to find the status of each of the three flags (1 for collected and 0 for not collected), and the actions were initialized as a 1D array of length 4 containing the values 0 for UP, 1 for DOWN, 2 for LEFT and 3 for RIGHT. Moreover, the policy was initialized as a 112x4 array where each element was 0.25.

The next step consisted of implementing the Policy Iteration algorithm in order to find the Optimal Policy. An outline of the algorithm can be found on Figure 1. The functions in the code that contain these calculations are **evaluate_policy()** and **policy_iteration()**. Additionally, the optimal Q values were obtained and saved. They can be found in the project folder as **q_values.npy**. The path of the robot and other results obtained in this part of the project can be found in the **Results** section of this document.

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
      $\Delta \leftarrow 0$
      For each $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\big[r + \gamma V(s')\big]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      old-action $\leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
      If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Figure 1: Policy Iteration Algorithm.

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Initialize $Q(s, a)$, for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
   Initialize S
   Repeat (for each step of episode):
      Choose A from S using policy derived from Q (e.g., ϵ-greedy)
      Take action A, observe R, S'
      $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
      $S \leftarrow S'$
   until S is terminal

Figure 2: Q-Learning Algorithm.

3.2 Part 2: Q-Learning

The second part of this project consisted of developing the Q-Learning algorithm and applying it to the agent in the **Maze** environment for 5,000 steps. The Q-Learning algorithm is shown on Figure 2. The first task of this part of the project consisted of experimenting with different learning rates. The learning rates in the range of 0.0 to 0.9 were tested in steps of 0.1. The second task consisted of implementing ϵ-greedy for the action selection rule, which can be found in the function **get_action_egreedy()** of the code. The values of epsilon in the rage 0.0 to 1.0 were tested in steps of 0.1. For the third task, the Root Mean Square Error (RMSE) was calculated between the optimal Q-values (Q*) obtained in Part 1 of this project and the Q-values obtained in Part 2 in order to determine which hyperparameters converge to Q* faster. During these experiments, the performance of the algorithm was evaluated by stopping the learning every 50 steps and plotting the average number of steps taken to complete an interval and the average reward over the last 50 steps. This was done by using the function **evaluation()**, in which the epsilon value was kept at 0.05. The resources consulted are found on references [6] and [7]. The results obtained for each experiment described above together with the optimal hyperparameters obtained and a discussion on the meaning of the results is available in the section **Results**.

3.3 PART 3: CONTINUOUS STATE SPACE PROBLEMS

The first step was to install the OpenAI Gym and understand the environments **Acrobot-v1** and **MountainCar-v0**, which have continuous state spaces and discrete action spaces. Then, a parameterized policy was chosen by using a softmax function and the REINFORCE with Baseline algorithm was implemented in the funcion **reinforce_with_baseline()** of the code. The REINFORCE with Baseline algorithm steps are outlined on Figure 3. The baseline used was the mean of the total cummulative rewards over time. Moreover, to encourange exploration, a new action was only chosen every 5 steps, and the maximum number of steps for the **MountainCar-v0** environment was increased to 1,000. Different values for the hyperparameters were evaluated by using the functions **evaluation_mountain_reinforce()** and **evaluation_acrobot_reinforce()** for both environments, and the optimal values were found. The epsilon value during evaluation was kept at 0.05. The resources consulted for this section are found on reference [8]. The resulting learning curves from each evaluation, together with the final hyperparameters chosen are in the section **Results**.

REINFORCE with Baseline (episodic)

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$

Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$
Repeat forever:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  For each step of the episode $t = 0, \ldots, T-1$:
    $G_t \leftarrow$ return from step t
    $\delta \leftarrow G_t - \hat{v}(S_t,\mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \gamma^t \delta \nabla_{\mathbf{w}} \hat{v}(S_t,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla_{\boldsymbol{\theta}} \ln \pi(A_t|S_t,\boldsymbol{\theta})$

Figure 3: Q-Learning Algorithm.

Next, the Q-learning algorithm was applied to the **MountainCar-v0** and **Acrobot-v1** environments. For this, linear function approximation with Radial Basis Function (RBF) kernels that account for different variances was used. Initially, state discretization was chosen for this section over function approximation and part of the code was written, but linear function approximation was selected as the final approach for the submission because it was found to have a shorter run-time and involve less memory storage. The function approximation was implemented using **sklearn** [9]. The functions **create_scaler()**, **prediction()**, **create_models()**, and **q_learning_func_approx()** in the code implement this Q-learning functionality. The performance was evaluated by extracting learning curves for the average number of steps and reward every 50 episodes by using the functions **evaluation_mountain()** and **evaluation_acrobot()** while keeping the epsilon value at 0.05.

Finally, the best hyperparameters for each of the algorithms were found by plotting the learning curves for all discount factors in the range 0.1 to 1.0 and epsilon values in the range 0.1 to 1.0. The results obtained in this section of the document together with the plots of the different learning curves can be found in the **Results** part of this report.

4 RESULTS

4.1 PART 1: POLICY ITERATION / VALUE ITERATION

The optimal policy can be found as a row array with 112 elements in the **epe26_project4.ipynb** code file as well as saved as **final_policy_row.npy** where each element corresponds to one of the 112 states and the value is an integer between 0 and 4 indicating the best action to take at each stage. The value 0 is UP, 1 is DOWN, 2 is LEFT, and 3 is RIGHT. The expanded 112x4 policy array where each 4-element row contains 3 zeros and a 1 indicating which action to take at each state can also be found in the .zip folder saved as **final_policy.npy**.

Moreover, the path that the agent takes in a run following the optimal policy and assuming no slippage is as follows:

- t = 0: Position 0 (START) at (0,0) with 0 flags which is state 0 -> take action DOWN
- t = 1: Position 1 at (1,0) with 0 flags which is state 8 -> take action RIGHT
- t = 2: Position 3 at (1,1) with 0 flags which is state 24 -> take action RIGHT
- t = 3: Position 7 at (1,2) with 0 flags which is state 56 -> take action UP
- t = 4: Position 6 at (0,2) with 1 flag (1,0,0) which is state 49 -> take action DOWN
- t = 5: Position 7 at (1,2) with 1 flag (1,0,0) which is state 57 -> take action RIGHT
- t = 6: Position 9 at (1,3) with 1 flag (1,0,0) which is state 73 -> take action DOWN
- t = 7: Position 10 at (2,3) with 1 flag (1,0,0) which is state 81 -> take action DOWN
- t = 8: Position 11 at (3,3) with 2 flags (1,0,1) which is state 93 -> take action UP
- t = 9: Position 10 at (2,3) with 2 flags (1,0,1) which is state 85 -> take action UP
- t = 10: Position 9 at (1,3) with 2 flags (1,0,1) which is state 77 -> take action RIGHT
- t = 11: Position 13 at (1,4) with 2 flags (1,0,1) which is state 109 -> take action UP
- t = 12: Position 12 (GOAL) at (0,4) with 2 flags (1,0,1) which is state 101 -> GAME OVER and GET REWARD

Once the game is over, the agent collects the reward and does not move anywhere else.

4.2 PART 2: Q-LEARNING

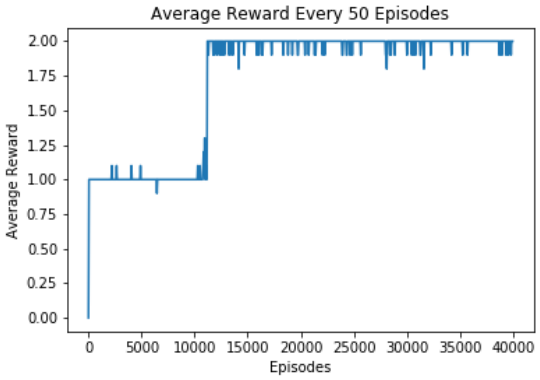
The learning curve was calculated for each of the hyperparameters described in the section **Technical Approach** of this document. First, each fixed value for the learning rate α in the range 0.1 to 0.9 and epsilon ϵ in the range 0.1 to 1.0 was explored in terms of average number of steps per episode, average reward per episode, and RMSE between Q-values. The other parameters set were a discount factor γ of 0.9, and an episode number of 40,000. The resulting plots can be found on the zip file **epe26_project4.zip** and in the code file **epe26_project4.ipynb**. For the purpose of not overloading this document, a smaller number of the figures can be found below. The reward plots can be found on Figure 4, the steps plot are on Figure 5, and Figure 6 contains the RMSE graphs.

These graphs show the algorithm converging to a reward of 2. As the learning rate decreases, the convergence gets more accurate, and the number of steps per episode decreases. Moreover, as epsilon increases, the algorithm converges faster and with less noise. This can be seen from both the steps

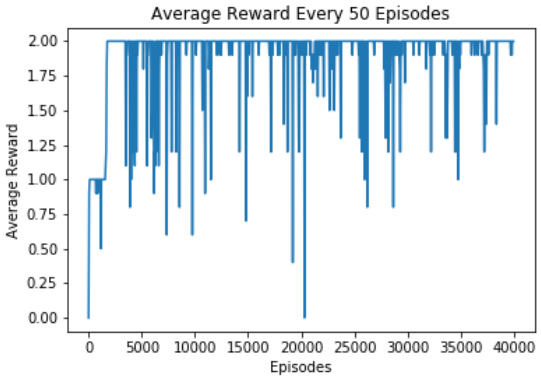
and the reward graphs. Additionally, changes in the learning rate slightly affect the RMSE, but it is the changes in the epsilon value that impact the curve most, with larger epsilon values leading to a lower RMSE between the Q-values obtained and the optimal Q-values faster. Keeping epsilon at a value of 1.0 during training allows the agent to continuously explore. Therefore, it was concluded that a learning rate α of 0.2 and an epsilon value ϵ of 1.0 were the optimal values for this task.

To further improve the Q-learning algorithm, an adaptative epsilon-greedy algorithm was tested, in which the epsilon value was decreased over time according to the formula $\epsilon = \frac{no}{no+visits(s_t)}$, where no is a constant hyperparameter to set the rate of decay and visits(s_t) represents the number of times a state was visited in that episode. Different values for no in the range of 50 to 10,000 were tested. Some of the resulting graphs can be found on Figure 7 below, and the rest can be found in the project folder. As can be seen from the plots, the decay over time of epsilon slightly improved the fixed value results and it was concluded that no = 100 gave the best performance.

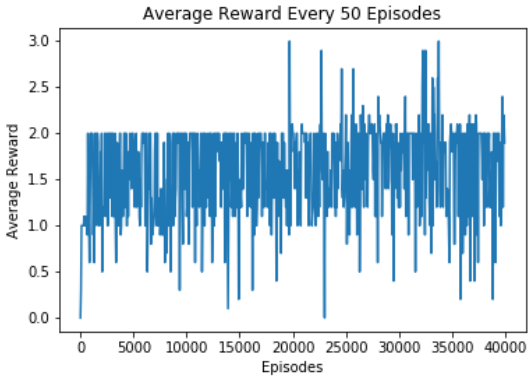
Therefore, we can conclude that the hyperparameters for which the algorithm converges faster during training are a learning rate α of 0.2, an epsilon ϵ value of 1.0, and an no value for epsilon decay of 100. All these learning plots were calculated by using the function **evaluation()** in the code.



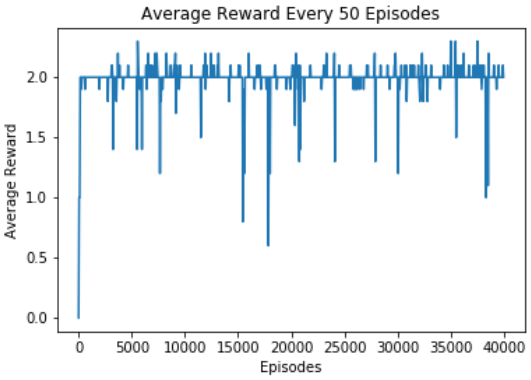
Average Reward for $\alpha = 0.1$, $\epsilon = 0.1$.



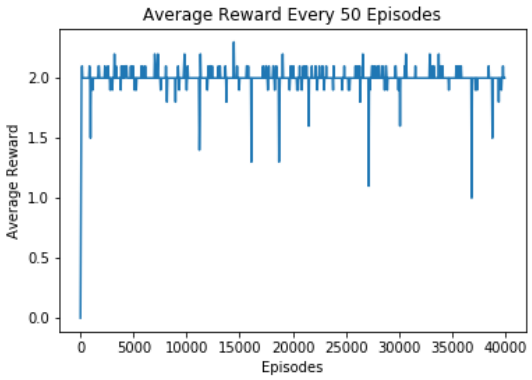
Average Reward for $\alpha = 0.5$, $\epsilon = 0.1$.



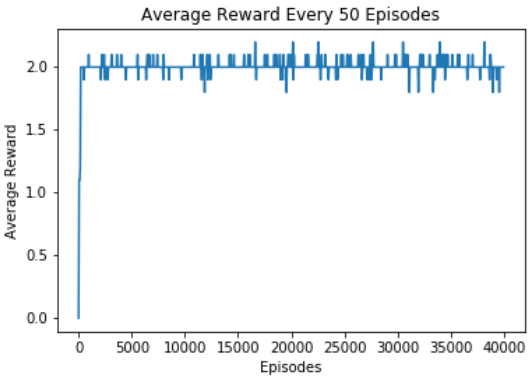
Average Reward for $\alpha = 0.9$, $\epsilon = 0.1$.



Average Reward for $\alpha = 0.5$, $\epsilon = 0.5$.



Average Reward for $\alpha = 0.5$, $\epsilon = 1.0$.



Average Steps for $\alpha = 0.2$, $\epsilon = 1.0$ - Optimal.

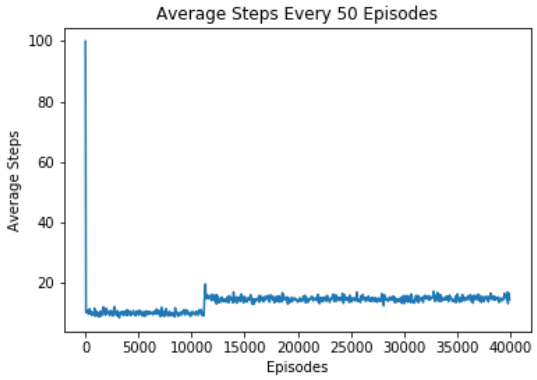
Figure 4: Several Average Reward Plots for the Maze Environment.

4.3 PART 3: CONTINUOUS STATE SPACE PROBLEMS

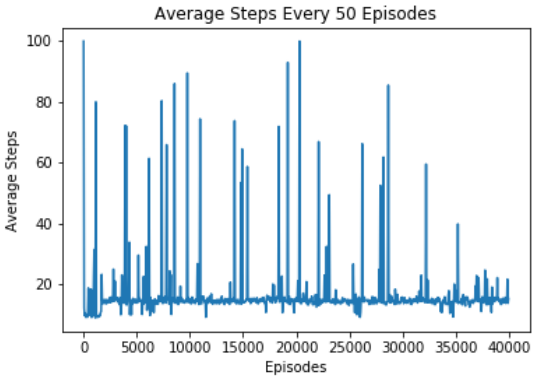
4.3.1 REINFORCE WITH BASELINE

For this section the learning curve was calculated for each of the hyperparameters described in the section **Technical Approach**. Regarding the REINFORCE with Baseline algorithm, the environments were tested for the discount factors 0.8, 0.99 and 1.0, and with the learning rates of 0.0001 and 0.001. The Steps and Rewards results for **MountainCar-v0** are shown in Figures 8 and 9, and the results for **Acrobot-v1** can be found on Figures 10 and 11.

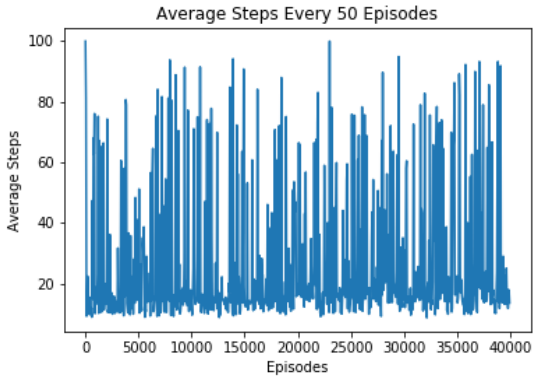
As can be seen from the plots, for the **MountainCar-v0** environment, the performance improves as the discount factor increases. The performance also improves as the learning rate decreases within a certain limit, as it causes the algorithm to learn faster and converge earlier. Therefore, the optimal hyperparameters for **MountainCar-v0** were found to be $\gamma = 1.0$ and $\alpha = 0.001$. In regards to **Acrobot-v1**, it can be seen that the performance increases as the discount factor increases, similarly to **MountainCar-v0**, and that a learning rate of 0.001 also suits the environment. The optimal hyperparameters for **Acrobot-v1** were found to be $\gamma = 0.99$ and $\alpha = 0.001$. Nevertheless, due to the random nature of the algorithm, it is not clear whether a discount factor of 1.0 could work better than 0.99 if the algorithm was executed a few more times.



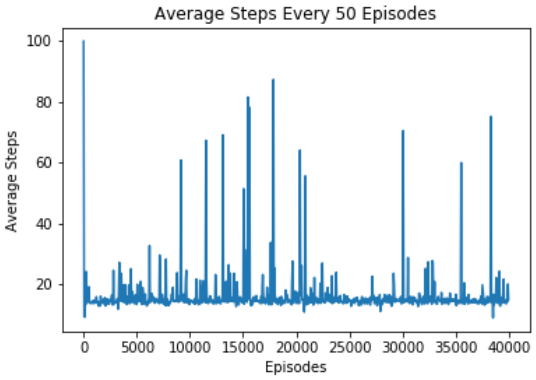
Average Steps for $\alpha = 0.1$, $\epsilon = 0.1$.



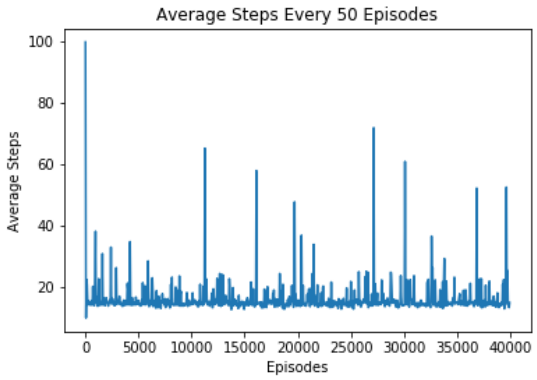
Average Steps for $\alpha = 0.5$, $\epsilon = 0.1$.



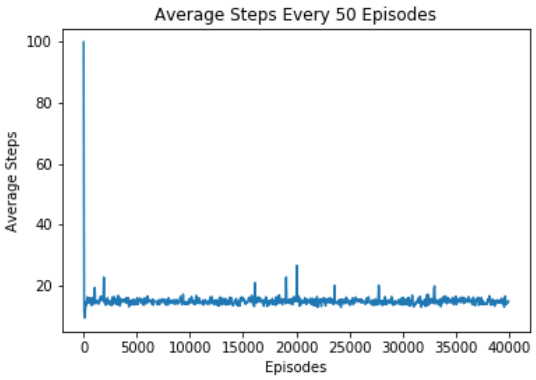
Average Steps for $\alpha = 0.9$, $\epsilon = 0.1$.



Average Steps for $\alpha = 0.5$, $\epsilon = 0.5$.

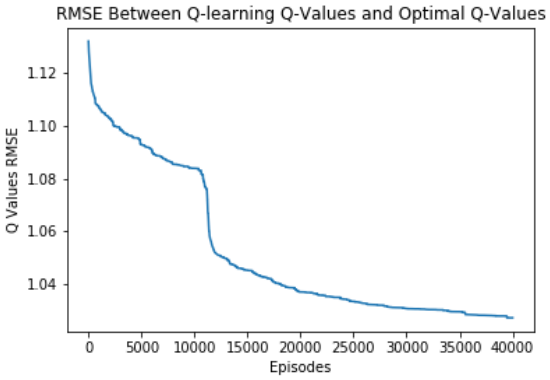


Average Steps for $\alpha = 0.5$, $\epsilon = 1.0$.

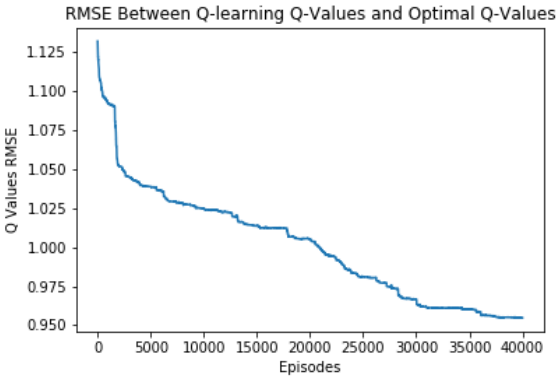


Average Steps for $\alpha = 0.2$, $\epsilon = 1.0$ - Optimal.

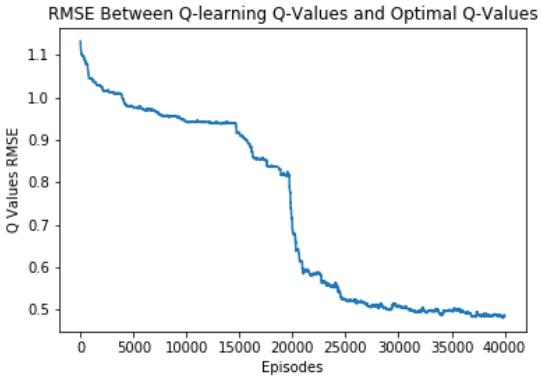
Figure 5: Several Average Steps Plots for the Maze Environment.



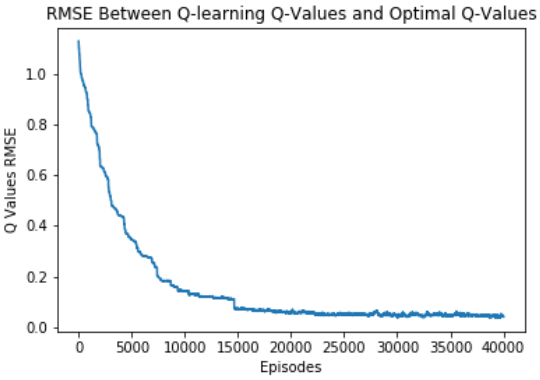
RMSE for $\alpha = 0.1, \epsilon = 0.1$.



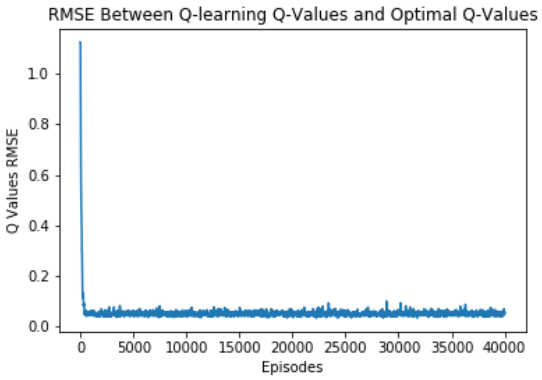
RMSE for $\alpha = 0.5, \epsilon = 0.1$.



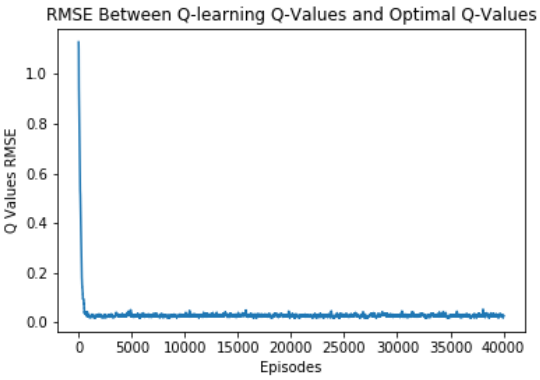
RMSE for $\alpha = 0.9, \epsilon = 0.1$.



RMSE for $\alpha = 0.5, \epsilon = 0.5$.

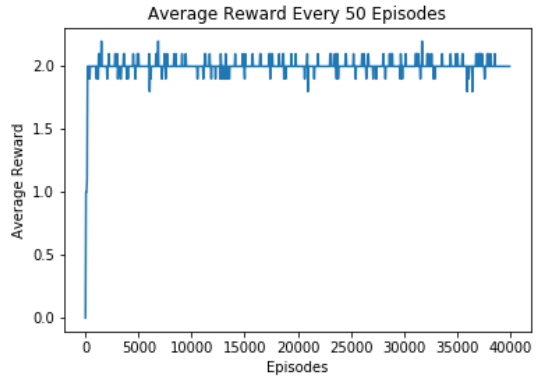


RMSE for $\alpha = 0.5, \epsilon = 1.0$.

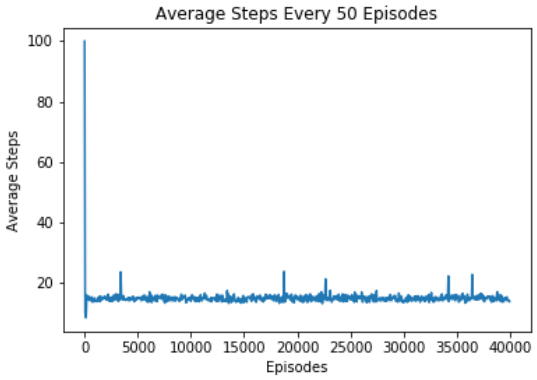


RMSE for $\alpha = 0.2, \epsilon = 1.0$ - Optimal.

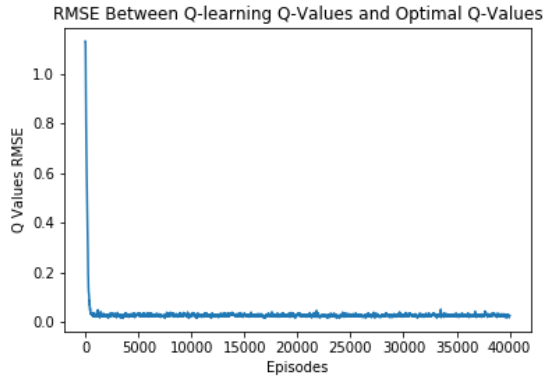
Figure 6: Several RMSE Plots for the Maze Environment.



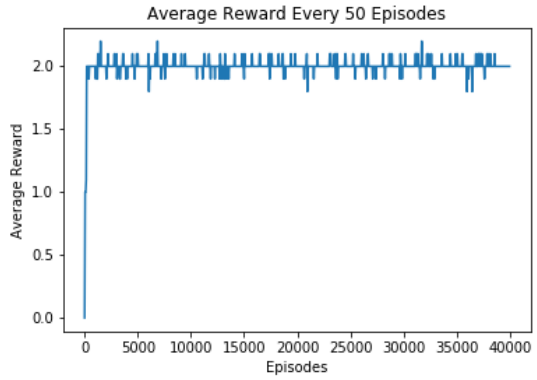
Average Reward for $\alpha = 0.2$, $\epsilon = 1.0$, and $no = 1000$.



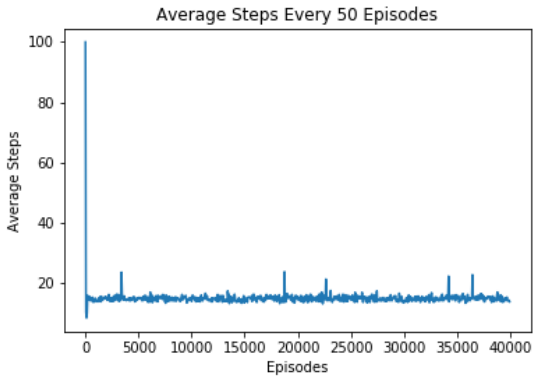
Average Steps for $\alpha = 0.2$, $\epsilon = 1.0$, and $no = 1000$.



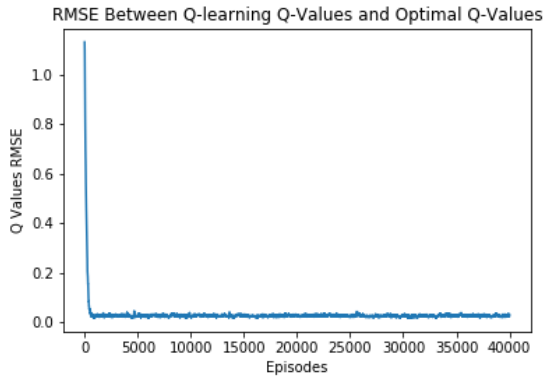
RMSE for $\alpha = 0.2$, $\epsilon = 1.0$, and $no = 1000$.



Average Reward for $\alpha = 0.2$, $\epsilon = 1.0$, and $no = 100$.

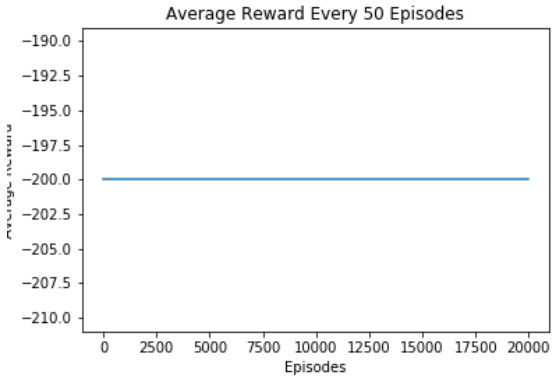


Average Steps for $\alpha = 0.2$, $\epsilon = 1.0$, and $no = 100$.

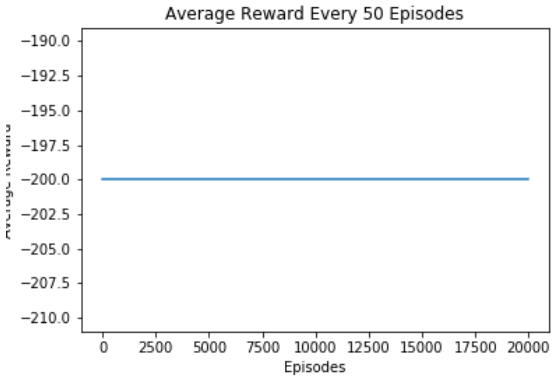


RMSE for $\alpha = 0.2$, $\epsilon = 1.0$, and $no = 100$ - Optimal.

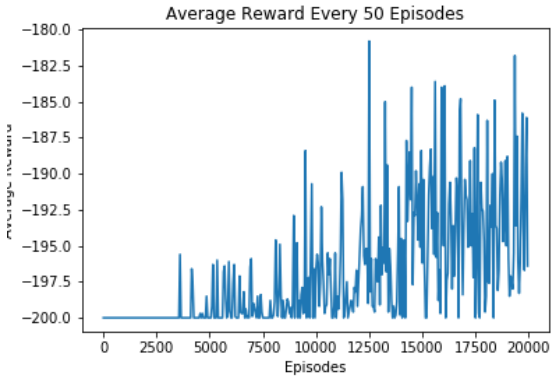
Figure 7: Plots using Epsilon Decay in the Optimal Case for the Maze Environment.



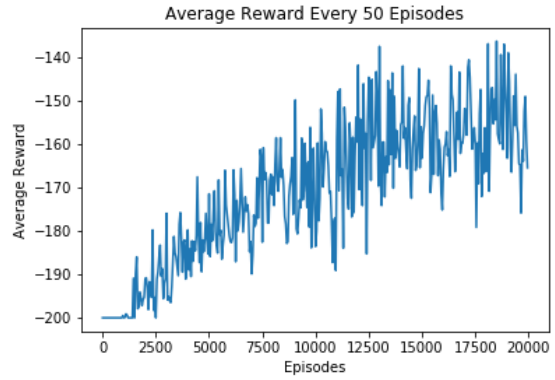
Average Reward for $\gamma = 0.8$ and $\alpha = 0.0001$.



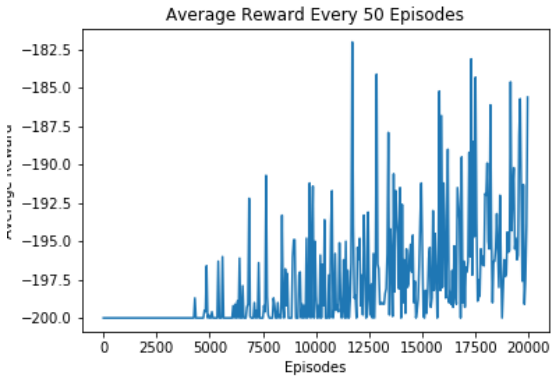
Average Reward for $\gamma = 0.8$ and $\alpha = 0.001$.



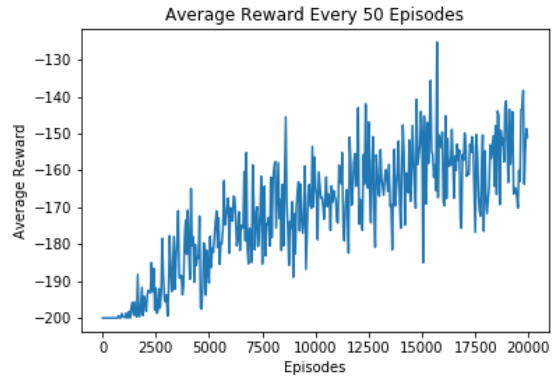
Average Reward for $\gamma = 1.0$ and $\alpha = 0.0001$.



Average Reward for $\gamma = 1.0$ and $\alpha = 0.001$ - Optimal.



Average Reward for $\gamma = 0.99$ and $\alpha = 0.0001$.



Average Reward for $\gamma = 0.99$ and $\alpha = 0.001$ - Optimal.

Figure 8: Several Average Reward Plots using REINFORCE with Baseline for the MountainCar-v0 Environment.

4.3.2 Q-LEARNING

First, each fixed value for the discount factor γ in the range 0.1 to 1.0 and epsilon ϵ in the range 0.1 to 1.0 was explored in terms of average number of steps per episode, and average reward per episode. The other parameters set were a maximum number of steps per episode of 1,000 for **MountainCar-v0** and 500 for **Acrobot-v1**. Even though the default value of steps per episode for **MountaiCar-v0** is 200, this number was increased during training to encourage exploration, and set back to 200 for evaluation. The resulting figures can be found on the zip file **epe26_project4.zip** and in the code file **epe26_project4.ipynb**. For the purpose of not overloading this document, a smaller number of the figures can be found below. The reward plots for **MountainCar-v0** can be found on Figure 12, and the steps plot are on Figure 13. Figure 14 contains the reward plots for **Acrobot-v1**, and Figure 15 contains its steps per episode plots.

The **MountainCar-v0** graphs show the algorithm converging to a reward of approximately -120. As the discount factor increases up to a factor of 1.0, the convergence improves and the number of steps per episode decreases. Moreover, as epsilon increases, the algorithm converges faster and with less noise. Therefore, it was concluded that a discount factor γ of 1.0 and an epsilon value ϵ of 0.9 were the optimal values for Q-learning in this environment.

The **Acrobot-v1** graphs show the algorithm converging to a reward of around -150. As the discount factor increases up to 0.9, the performance of the algorithm in regards to both number of steps to complete each episode and reward obtained for each episode improves. Moreover, as epsilon decreases, the algorithm improves its accuracy and convergence to a reasonable value. Therefore, it was concluded that a discount factor γ of 0.9 and an epsilon value ϵ of 0.1 are the optimal values for Q-learning in **Acrobot-v1**.

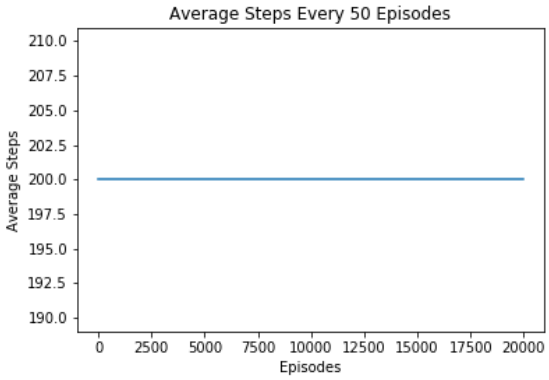
5 DISCUSSION

Regarding the first part of the project, the path chosen by the robot gives the highest reward as compared to the rest of the paths. The manual calculation gives an expected score of around 0.56. It can be seen from the code that the state values equal the maximum Q value for every state, and that the policy lets the agent know what action to take at each state, which corresponds to the action with maximum Q value at that state.

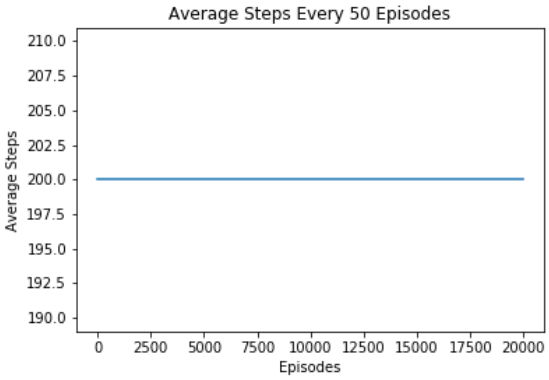
In regards to the second part of the project, Q-learning successfully achieves convergence after the first few episodes, with the optimal hyperparameters being $\alpha = 0.2$, $\epsilon = 1.0$, and $no = 100$ for the reasons explained in **Results**. The final RMSE between the Q-values for the **Maze** environment obtained in Part 1 and Part 2 is negligible, so the Q-learning algorithm can be considered to be working correctly.

For the third part of the project, the REINFORCE with Baseline algorithm reaches convergence for both the **MountainCar-v0** and the **Acrobot-v1** environments. The optimal hyperparameters for MountainCar-v0 are a learning rate γ of 0.99 and a discount factor α of 0.001. Moreover, the optimal hyperparameters for Acrobot-v1 are a learning rate γ of 0.99 and a discount factor α of 0.001.

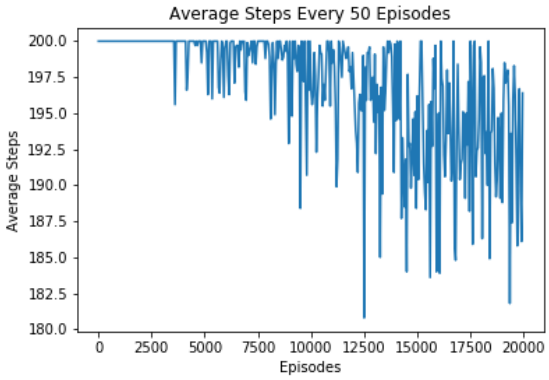
Additionally, the Q-learning algorithm on both of the continuous state spaces **MountainCar-v0** and **Acrobot-v1** successfully converges to a reward of -120 and -150, respectively. The optimal hyperparameters to reach this convergence in **MountainCar-v0** are a discount factor of 1.0 and an epsilon value of 0.9, and in **Acrobot-v1** they are a discount factor of 0.9 and an epsilon value of 0.1. The reasoning for this can be found in the **Results** part above.



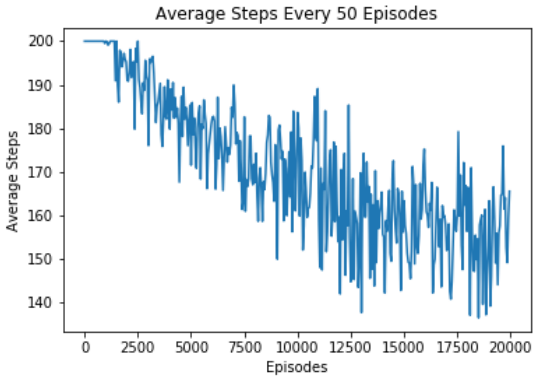
Average Steps for $\gamma = 0.8$ and $\alpha = 0.0001$.



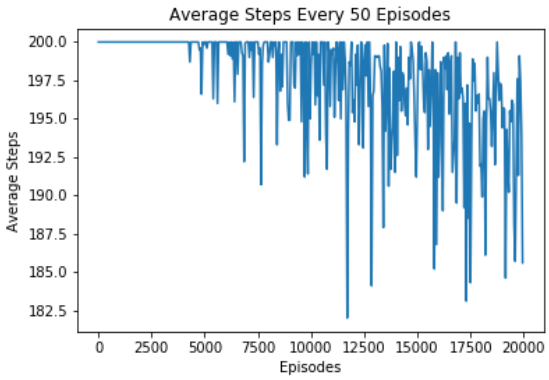
Average Steps for $\gamma = 0.8$ and $\alpha = 0.001$.



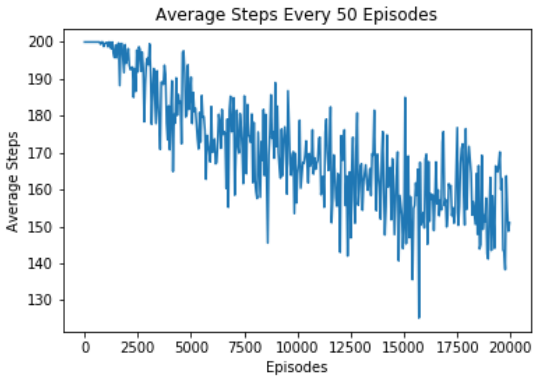
Average Steps for $\gamma = 1.0$ and $\alpha = 0.0001$.



Average Steps for $\gamma = 1.0$ and $\alpha = 0.001$ - Optimal.

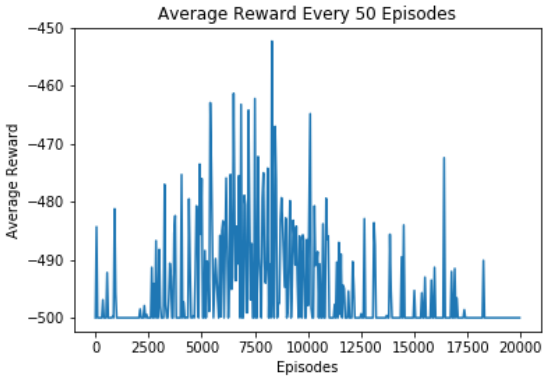


Average Steps for $\gamma = 0.99$ and $\alpha = 0.0001$.

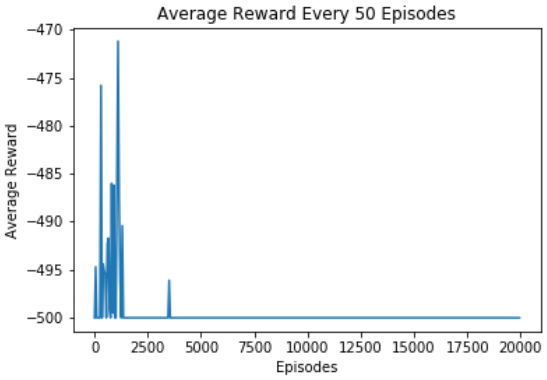


Average Steps for $\gamma = 0.99$ and $\alpha = 0.001$ - Optimal.

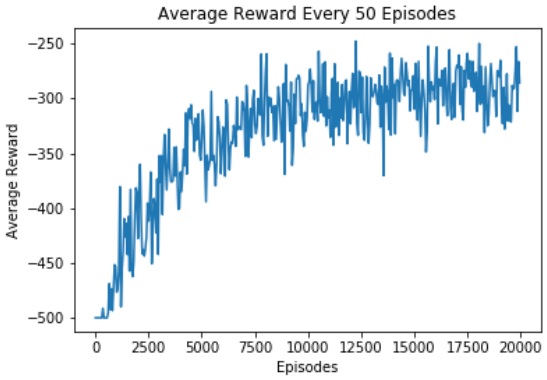
Figure 9: Several Average Steps Plots using REINFORCE with Baseline for the MountainCar-v0 Environment.



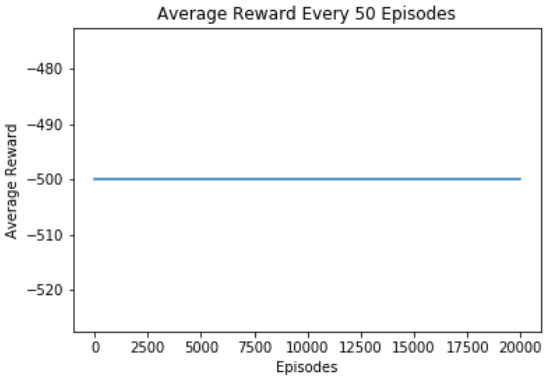
Average Reward for $\gamma = 0.8$ and $\alpha = 0.0001$.



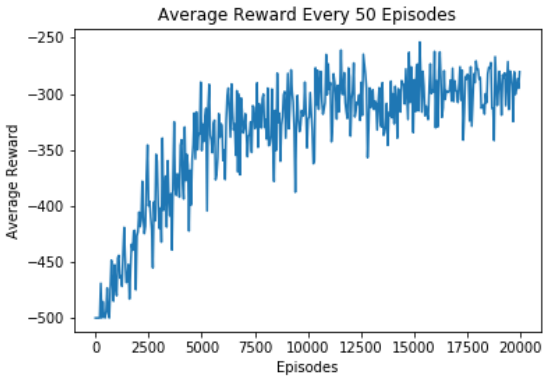
Average Reward for $\gamma = 0.8$ and $\alpha = 0.001$.



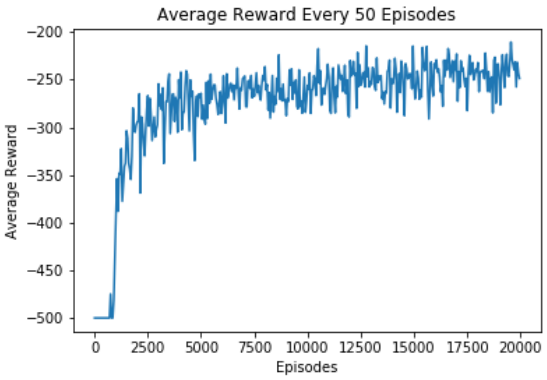
Average Reward for $\gamma = 1.0$ and $\alpha = 0.0001$.



Average Reward for $\gamma = 1.0$ and $\alpha = 0.001$.

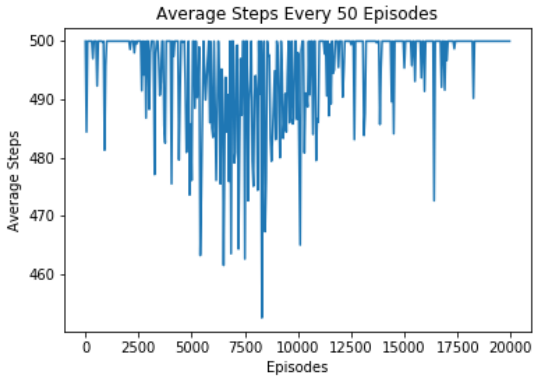


Average Reward for $\gamma = 0.99$ and $\alpha = 0.0001$.

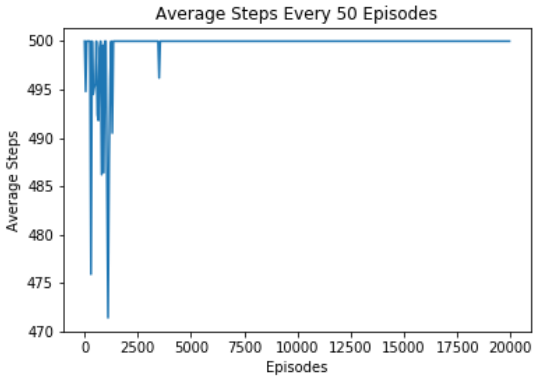


Average Reward for $\gamma = 0.99$ and $\alpha = 0.001$ - Optimal

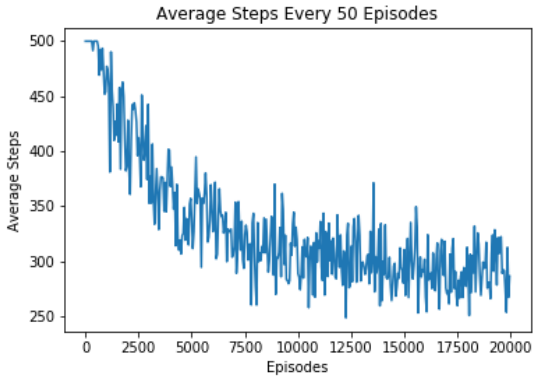
Figure 10: Several Average Reward Plots using REINFORCE with Baseline for the Acrobot-v1 Environment.



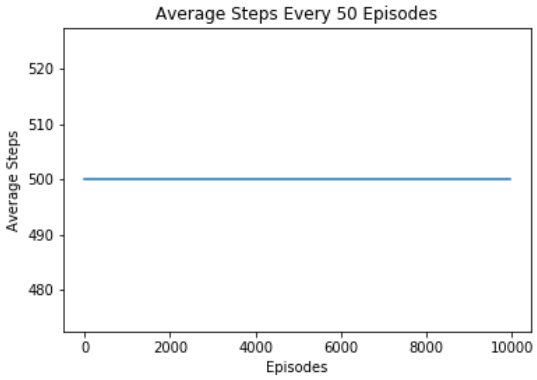
Average Steps for $\gamma = 0.8$ and $\alpha = 0.0001$.



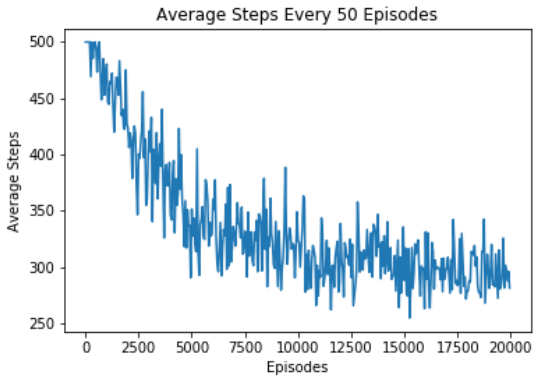
Average Steps for $\gamma = 0.8$ and $\alpha = 0.001$.



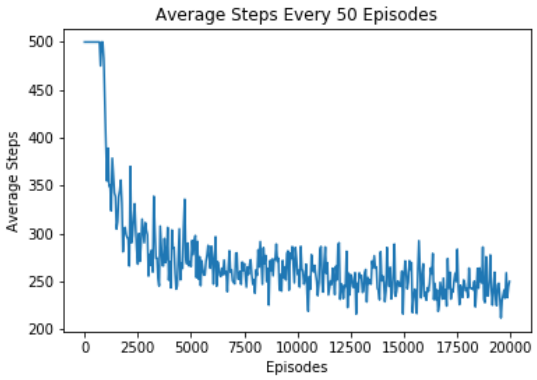
Average Steps for $\gamma = 1.0$ and $\alpha = 0.0001$.



Average Steps for $\gamma = 1.0$ and $\alpha = 0.001$.

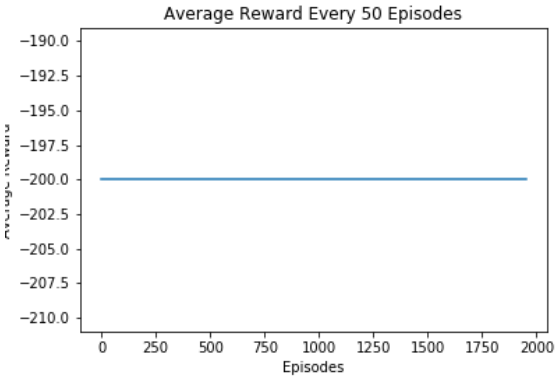


Average Steps for $\gamma = 0.99$ and $\alpha = 0.0001$.



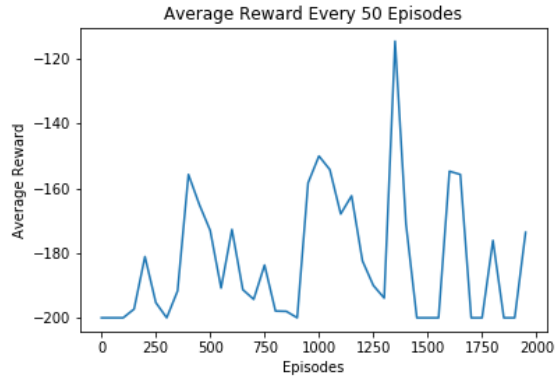
Average Steps for $\gamma = 0.99$ and $\alpha = 0.001$ - Optimal.

Figure 11: Several Average Steps Plots using REINFORCE with Baseline for the Acrobot-v1 Environment.



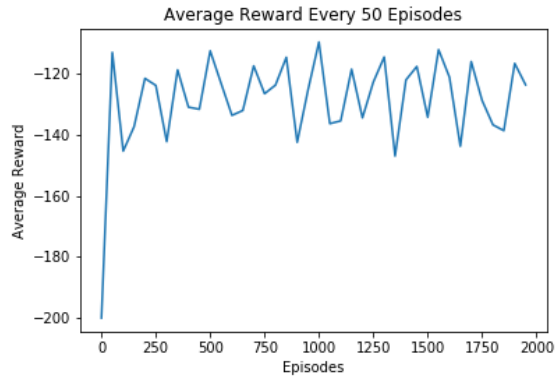
Average Reward for $\gamma = 0.5$ and $\epsilon = 0.5$.

Average Reward for $\gamma = 0.5$ and $\epsilon = 1.0$.

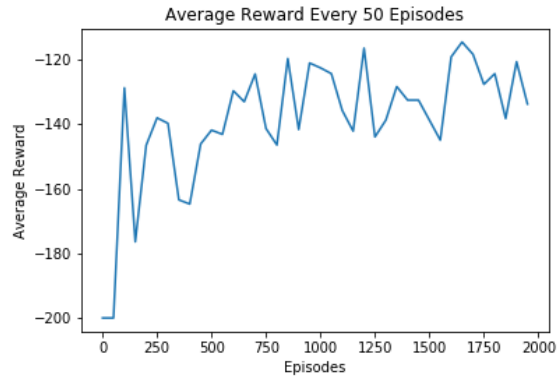


Average Reward for $\gamma = 0.9$ and $\epsilon = 0.1$.

Average Reward for $\gamma = 0.9$ and $\epsilon = 1.0$.

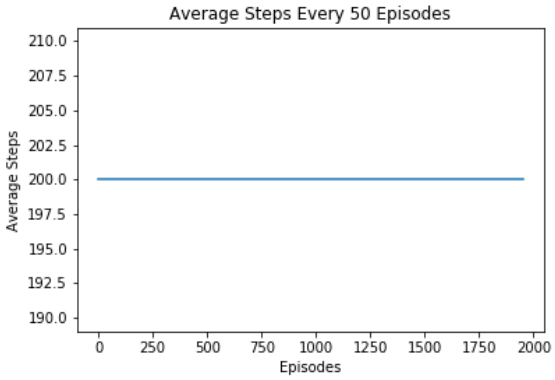


Average Reward for $\gamma = 1.0$ and $\epsilon = 0.9$ - Optimal.

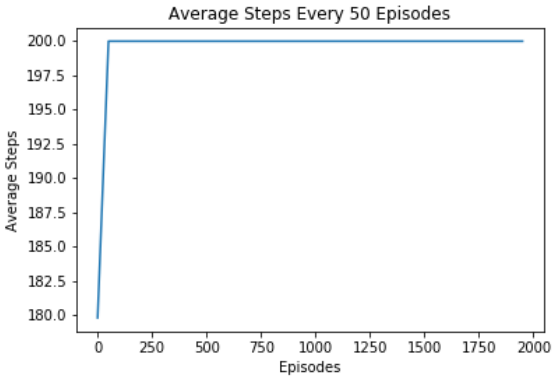


Average Reward for $\gamma = 1.0$ and $\epsilon = 0.1$.

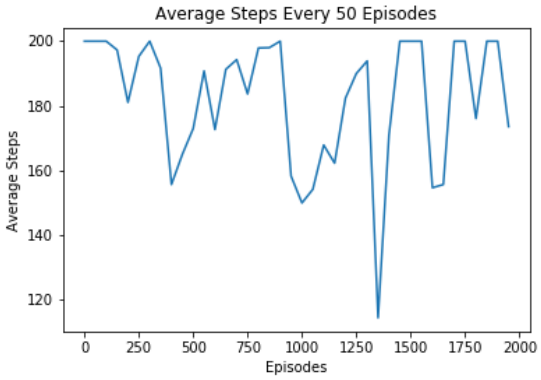
Figure 12: Several Average Reward Plots using Q-Learning for the MountainCar-v0 Environment.



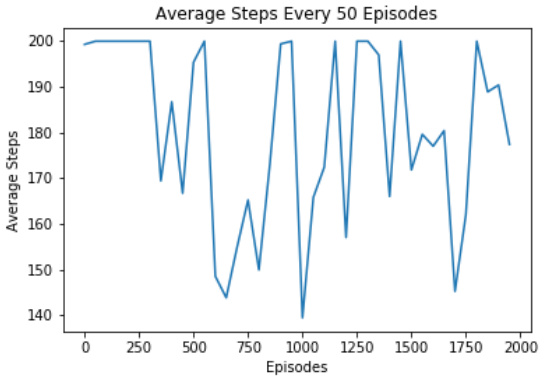
Average Steps for $\gamma = 0.5$ and $\epsilon = 0.5$.



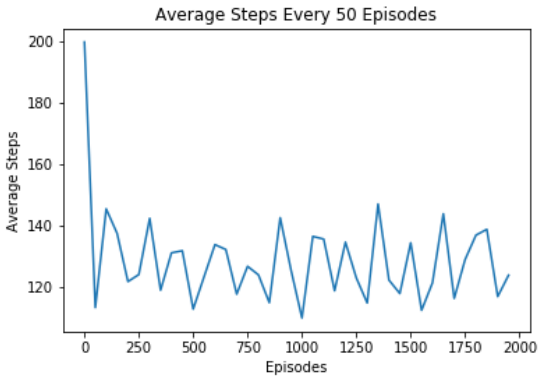
Average Steps for $\gamma = 0.5$ and $\epsilon = 1.0$.



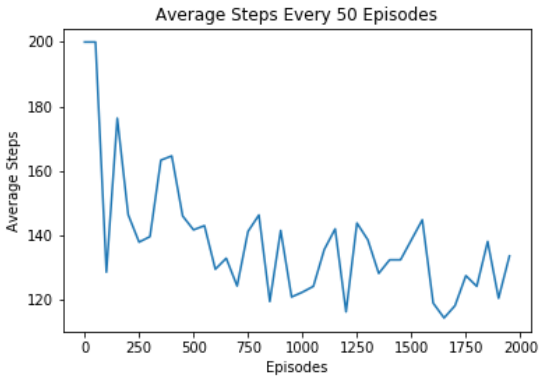
Average Steps for $\gamma = 0.9$ and $\epsilon = 0.1$.



Average Steps for $\gamma = 0.9$ and $\epsilon = 1.0$.

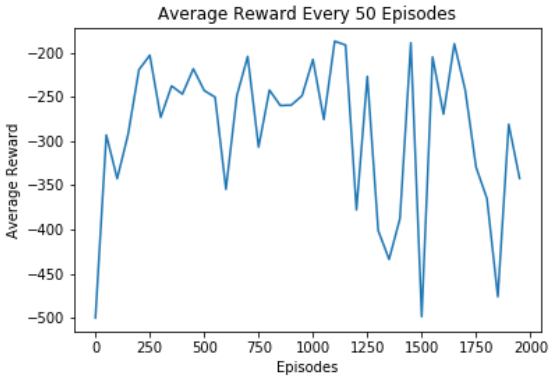


Average Steps for $\gamma = 1.0$ and $\epsilon = 0.9$ - Optimal.

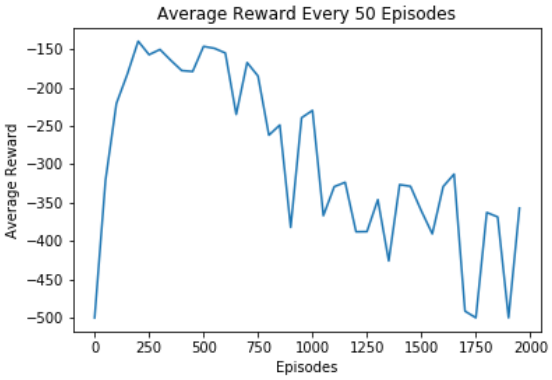


Average Steps for $\gamma = 1.0$ and $\epsilon = 0.1$.

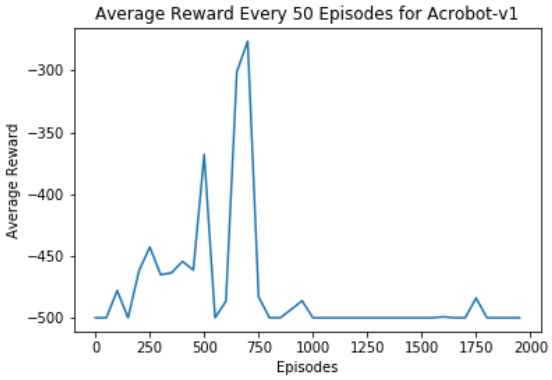
Figure 13: Several Average Steps Plots using Q-Learning for the MountainCar-v0 Environment.



Average Reward for $\gamma = 0.5$ and $\epsilon = 0.5$.



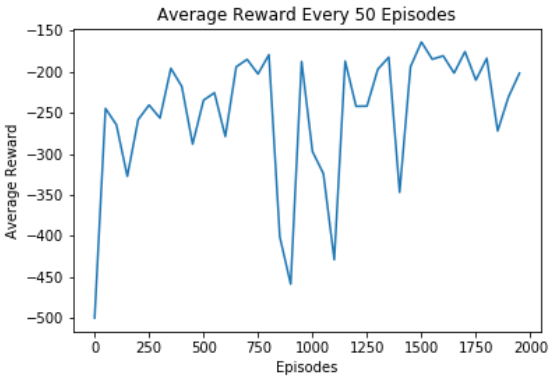
Average Reward for $\gamma = 0.5$ and $\epsilon = 1.0$.



Average Reward for $\gamma = 0.5$ and $\epsilon = 0.1$.

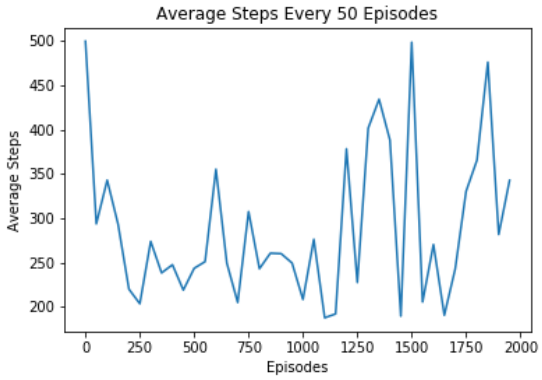
Average Reward for $\gamma = 0.9$ and $\epsilon = 0.1$ - Optimal.

Average Reward for $\gamma = 0.9$ and $\epsilon = 0.9$.

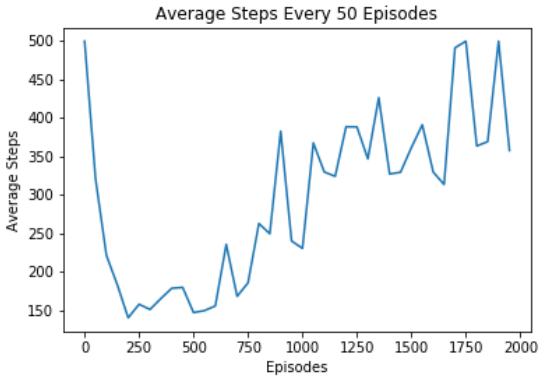


Average Reward for $\gamma = 1.0$ and $\epsilon = 0.1$.

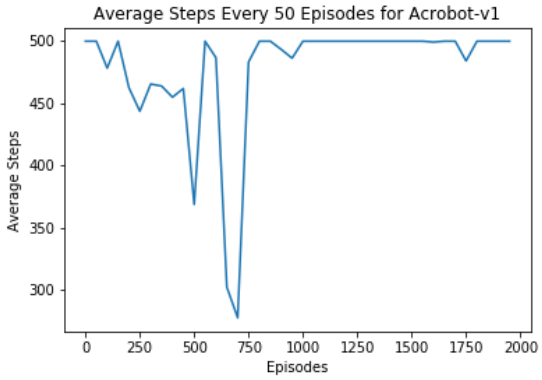
Figure 14: Several Average Reward Plots using Q-Learning for the Acrobot-v1 Environment.



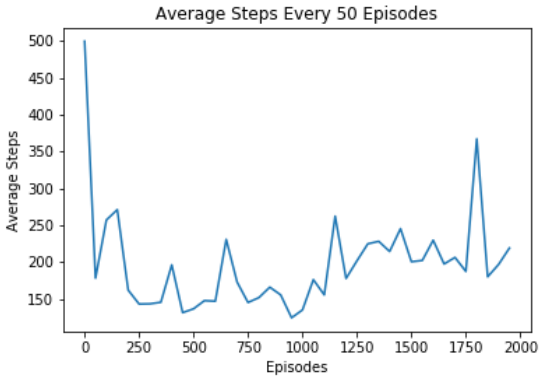
Average Steps for $\gamma = 0.5$ and $\epsilon = 0.5$.



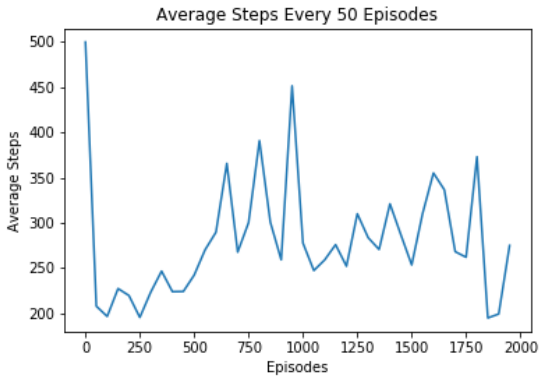
Average Steps for $\gamma = 0.5$ and $\epsilon = 1.0$.



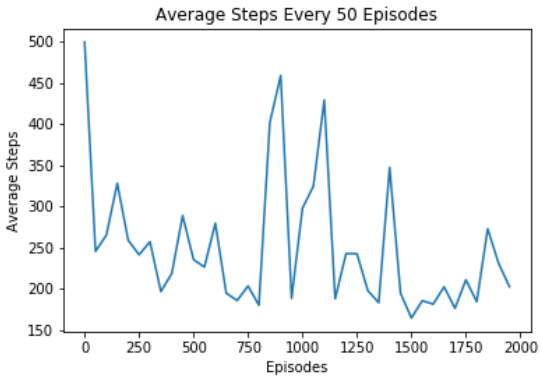
Average Steps for $\gamma = 0.5$ and $\epsilon = 0.1$.



Average Steps for $\gamma = 0.9$ and $\epsilon = 0.1$ - Optimal.



Average Steps for $\gamma = 0.9$ and $\epsilon = 0.9$.



Average Steps for $\gamma = 1.0$ and $\epsilon = 0.1$.

Figure 15: Several Average Steps Plots using Q-Learning for the Acrobot-v1 Environment.

6 FUTURE WORK

Future work includes fine-tuning the hyperparameters and optimizing the code to achieve a better convergence and a shorter computation time of the Q-learning algorithm in both the **Maze** and the **Gym** environments. Additionally, both the hyperparameters and the code could be optimized to improve the performance of the REINFORCE with Baseline algorithm presented above. Finally, different baseline options could be tested, such as implementing a value estimator based on Kernel Approximation to improve the converge of REINFORCE.

REFERENCES

[1] Wikipedia. (2020) Reinforcement Learning. "https://en.wikipedia.org/wiki/Reinforcement_learning"[Accessed 04-16-2020].

[2] O. Gym. (2020) Gym. "https://gym.openai.com/"[Accessed 04-16-2020].

[3] Wikipedia. (2020) Policy Iteration. "https://en.wikipedia.org/wiki/Markov_decision_process#Policy_iteration"[Accessed 04-16-2020].

[4] ——. (2020) Policy Iteration. "https://en.wikipedia.org/wiki/Q-learning"[Accessed 04-16-2020].

[5] Scholarpedia. (2020) REINFORCE. "http://www.scholarpedia.org/article/Policy_gradient_methods#Likelihood_Ratio_Methods_and_REINFORCE"[Accessed 04-16-2020].

[6] D. Silver. (2019) Reinforcement Learning. "https://github.com/dennybritz/reinforcement-learning"[Accessed 04-16-2020].

[7] ——. (2018) Reinforcement Learning. "https://github.com/dalmia/David-Silver-Reinforcement-learning"[Accessed 04-16-2020].

[8] S. Kirkiles. (2018) REINFORCE Policy Gradients From Scratch In Numpy. "https://medium.com/samkirkiles/reinforce-policy-gradients-from-scratch-in-numpy-6a09ae0dfe12"[Accessed 04-16-2020].

[9] A. G. e. a. Fabian Pedregosa, Gaël Varoquaux, “Scikit-learn: Machine learning in python,” *Journal of Machine Learning Research*, 2011.