University of Southern Mississippi

COS 703 Final Project

Comparison of Deep Reinforcement Q-Learning Algorithms in Partially Observable Environments

Daniel Lucas Thompson

May 8, 2017

1 Introduction

In this project, two different reinforcement learning algorithms will be compared on similar test bench problems. The Deep Q-Learning (DQN) algorithm [1] will be compared to the Deep Recurrent Q-Learning (DRQN) algorithm [2]. The problem comparison will be done using the Open AI Gym module [3]. The problem being tested will be two variations of the Cart Pole problem. One will be fully observable, and the other will be partially observable.

In the DQN and DRQN papers, the algorithms were used to learn to play Atari games by using the images as input with convolutional neural networks. This project consists of two main goals. The first being to modify the original algorithms to use individual feature values rather than images. The second goal of this project is to compare whether DRQN can learn to a level comparable to DQN when some information is removed from the problems making the problem partially observable.

In the reinforcement learning problem that will be used, several rounds of testing will be done on regular and modified versions of the problem. Both DQN and DRQN algorithms will be tested on the standard problems. Next, the problems will be modified to remove some information, making the problem partially observable, and the DQN and DRQN algorithms will be tested again. Under the partially observable problem, the DRQN algorithm should be able to infer the information that has been obscured while the DQN algorithm should perform poorly.

2 Tools Used

This project will make use of several libraries. The programming language used for the project will be Python. Several main python modules will be used, including the following. The OpenAI gym module will be used to handle the test bench problems. The Keras module will be used to handle the deep learning neural network code. Data analysis will be done using Jupyter Notebooks using IPython with Pandas and Matplotlib.

3 Reinforcement Learning

Reinforcement learning is a type of machine learning problem. The main idea of reinforcement learning is to allow an algorithm to interact in an environment and get immediate feedback on its decisions. These algorithms are sometimes referred to as agents or learners. These agents interact in an environment by performing actions, and performing these actions in the environment returns an observation and a reward value. In reinforcement learning, these observations and rewards represent the data set. Meaning, there is no data at the start of the learning. The data comes in a piece at a time as the agent is interacting with the environment. This type of data stream is generally referred to as online learning.

The data in reinforcement learning are the observations and reward value. The observation is the agents perception of the hidden state of the environment, and the reward value is produced by a programmer designed function to guide the agent in its choices. For every action the agent performs in the environment, it receives an observation and reward value. The agent uses these observations and reward values in its algorithm to calculate the best action to perform in the environment.

4 Q-Learning

The two algorithms discussed in this paper will make use of a common reinforcement learning algorithm called Q-Learning. Q-Learning can be used to find the optimal action policy for a Markov Decision Process. In Q-Learning, an agent learns to map state action pairs to Q-Values. A Q-Value represents the scores of performing an action in a given state and then following the optimal Q-Learning policy for future actions. The Q function is defined as follows.

$$Q(s,a) = Q(s,a) + \alpha(r + \lambda max_{a'} Q(s', a'))$$

Where α is a learning rate, r is the reward for performing the current action a in the current state s, and λ is the future discount factor. The equation $max_{a'} Q(s', a')$ selects the best action a' to perform in state s' where s' is the state you transition to after performing action a in state s. This is known as the Bellman equation, where $Q(s,a)$ is calculated as the current reward for performing an action in the current state plus the future rewards for performing actions in future states.

The Q-Learning algorithm initially starts with inaccurate values. As the agent receives more data samples, it uses these samples to improve its estimate of the Q-Values. When the agent is executing actions in the environment, it can select the best current action to perform by inputting the current state with all available actions and selecting the maximum value.

5 Deep Q-Learning

The Deep Q-Learning (DQN) algorithm approximates the Q-Learning method using a deep neural network. In the original DQN paper [1], a mixture of feed forward and convolutional layers were used to learn to play Atari games. In this project, the network structure is made up of an input layer with an input for each observation, multiple hidden feed forward layers, and an output layer with an output for each action. Since this does not use images, no convulutional layers are needed, so they are replaced with feed forward layers. To use the network, an observation is input, and all Q-Values for each action is output. To choose the action to perform, you select the maximum Q-Value.

In the original Deep Q-Learning paper [1], the authors makes use of several advanced techniques to train their network. In this project, the scale of data is much smaller, so all the techniques aren't needed. In this project, the techniques used include experience replay, target networks, and $\epsilon-greedy$ exploration.

Experience replay is a technique of saving past experiences from a reinforcement learning training episode. During the learning process, transaction tuples of the form (s, a, r, s') are saved in a list referred to as replay memory. These experiences are then used when training the network by selecting random batches from the replay memory which allows the network to be trained in a similar fashion to supervised learning rather than having to train as each single online data sample comes in.

Target networks are another technique used that helps make the training similar to supervised learning. Without target networks, the network is training to approximate the function $Q(s, a)$, but the target values that the network is attempting to train to, which include the function $Q(s', a')$, are also produced by this network. In order to make the training more stable, a separate copy of the network is made at certain intervals. Only updating this network occasionally helps the network training because the target values are fixed for several training iterations.

The final technique is an exploration-exploitation technique called $\epsilon{-}greedy$ exploration. In the Q-Learning algorithm, the maximum action is selected for the agent to perform. In the early stages of training, the Q-Values are very poorly estimated, so choosing the maximum value may not be the best choice. Using this exploration technique, the agent chooses a random action with a probability ϵ. Otherwise, it uses the maximum Q-Value action. This allows the agent to keep exploring to find better solutions as the Q-Values are converging.

Algorithm 1 Deep Q-Learning

```
 1: Initialize the Replay Memory D
 2: Initialize the Q network with random weights
 3: $s \leftarrow$ the initial problem state
 4: repeat
 5:     $a \leftarrow$ random with probability $\epsilon$, or $argmax_a Q(s, a)$
 6:     $s' \leftarrow$ perform action $a$ in the environment to get $s'$
 7:     $r \leftarrow$ reward from performing $a$ in $s$
 8:     $D \leftarrow$ append $(s, a, r, s')$
 9:     $B \leftarrow$ sample a batch from replay memory $D$
10:     for each transaction in batch B do
11:         if $s$ is the terminal state then
12:             $target \leftarrow r$
13:         else
14:             $target \leftarrow r + \lambda max_{a'} Q(s', a')$
15:         end if
16:     end for
17:     Train the network using $(targets - Q(s, a))^2$ as loss
18:     $s \leftarrow s'$
19: until average reward reached
```

The Deep Q-Learning (DQN) algorithm is performed in an on-line training fashion. Meaning, the agent is receiving observations from the environment and performing actions in the environment while training is taking place. It is performed in a sequential fashion. The agent receives a starting state and performs its initial action. The environment then returns a reward and the next observation. This agent-environment interaction continues until the agent solves the problem or a pre-determined time limit is reached. This is referred to as an episode. The reward at each step is summed to give a total reward for the episode. The agent is trained until an average reward value over several episodes has been reached.

The above Algorithm 1 explains this in more detail. In lines 1 and 2 initialize the replay memory to empty and give the neural network a random initialization. In line 3, the initial starting state, also referred to as an observation, is received from the environment and it is assigned to our starting state s. In line 4, we enter a loop that is repeated for the entire training process. In lines 5 through 9, we build and sample the replay memory. The values for a, s', and r are received from interacting with the environment. In lines 10 through 16, we iterate over each individual sample in the batch and compute its target value. If it is a terminal sample, meaning this sample was the final transaction before winning or losing an episode, it gets just the current reward value. If it is not a terminal sample, it gets the current reward value plus the future discounted reward value. In line 17, the neural network is trained using the small batch of samples that were calculated in the previous lines. This training continues in an on-line fashion until the agent is able to achieve a pre-determined average reward across a number of episodes.

6 Deep Recurrent Q-Learning

Deep Recurrent Q-Learning (DRQN) makes use of the previous DQN algorithm with two minor changes. The first change is to replace the last feed forward layer in the network with a recurrent Long Short Term Memory (LSTM) layer. The second change is a side effect of the LSTM layer which requires a change to the replay memory in order to store transaction sequences. The DRQN algorithm is identical to the DQN algorithm in all other aspects.

A LSTM layer has a different architecture from a normal feed forward network. In the LSTM layer, there are multiple sets of weights that govern an internal cell state. These weights are referred to as gates, and there are three kinds. The forget gate allows the LSTM cell to forget or erase a part of the cell state. The next gate, referred to as an input gate, decides which values of the cell state will be updated. The output gate determines the extent to which the internal state is used in the final output. This allows the layer to keep an internal state that represents past values that let it better predict future values.

To train the LSTM layer, the data must be in a sequential series format. Meaning, that the training samples must come from consecutive steps in an episode. Because of this, the replay memory must be modified. The replay memory is changed to keep a sequence of tuples of the form (s_0, a_0, r_0, s'_0), (s_1, a_1, r_1, s'_1), ... ,(s_n, a_n, r_n, s'_n) where n is the length of the episode. When training, a batch of episodes are sampled. From these episodes, a small sub-segment of these sequential tuples are sampled. The length of this sub-segment is referred to as the trace length and they are sampled starting from a random point in the episode. The network is then trained on the batches of these sequences. With these changes made, the algorithm is trained in the same way as Algorithm 1 in the previous section.

Adding this LSTM layer allows the network to keep an internal memory associated with the past observations that were passed through the network. This LSTM layer should be able to infer obscured information from the test problems, as long as the hidden and visible values of the problem have some corelation.

7 Test Bench Problems

This section will describe and detail the test bench problems that will be used in this report. The Cart Pole problem will be used to make comparisons, but two versions of the problem will be used. The first test will consist of the fully observable problem that provides all the observations to the algorithm. The second problem will be a partially observable implementation where some of the observations are withheld from the algorithm.

In the partially observable problems, the data that is removed must still have some correlation to the data that remains. The data that will be removed from the problems will be the velocity values. The data that remains in the problem will be the position values. Given positions at consecutive time steps, velocity values can be inferred.

The theory behind this selection is the comparison between the DQN and DRQN algorithms. In the partially observable problem, the DQN algorithm has no memory state, so it should do a poor job with the problem. However, the DRQN algorithm should be able to use its LSTM layer to "remember" positions and infer the velocity to perform much better than the DQN algorithm.

7.1 Cart Pole

In the Cart Pole problem, a pole is attached to a small cart. The goal of the problem is to balance the pole by issuing actions to the cart to move left or right. The fully observable test will use four observations to make a decision on which action to choose. Those observations are the cart position, cart velocity, pole angle, and pole velocity at the tip of the pole. The partial observable test will only use two observations. Those observations are cart position and pole angle to make a decision on which action to choose.

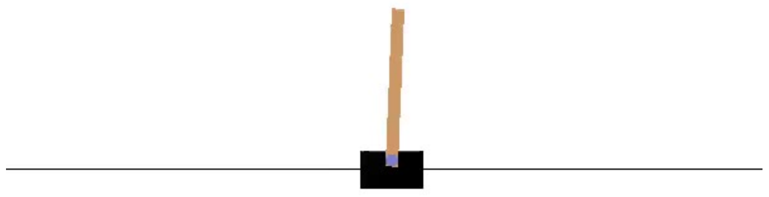

Figure 1: An Illustration of the Cart Pole Problem

8 Results

The results section is broken up into two main sections. In the first section, the results for the DQN algorithm will be shown for the fully observable and partially observable problems. In the second section, the results for DRQN will be shown for the fully observable and partially observable problems.

For each algorithm and problem type, a total of 10 training runs were used. After training is completed, a testing run of 100 episodes for each model is used to evaluate the performance. In the graphs, there are three plots. The green line shows the goal reward that is needed to consider the problem solved. The blue line shows the final reward for each individual episode. The red line shows the average reward over the last 100 episodes. For the training, once the average reward reaches the goal reward, training is stopped and testing is done. If the average reward never reaches the goal reward, training is stopped at 5000 episodes, and testing is done. In the testing graphs, the green line is shown for comparison. The testing does not stop if the reward is greater.

8.1 DQN Algorithm

In this section, the results for Deep Q-Learning algorithm will be displayed.

8.1.1 DQN Fully Observable

In figure 2, the results for the fully observable unmodified Cart Pole problem are shown for the DQN algorithm. In all ten training runs, the average reward made it to the goal reward. For the test results in figure 3, the agent was able to score the maximum value of 1000 on 9 out of the 10 runs, and obtained a value above the goal of 400 on the remaining.

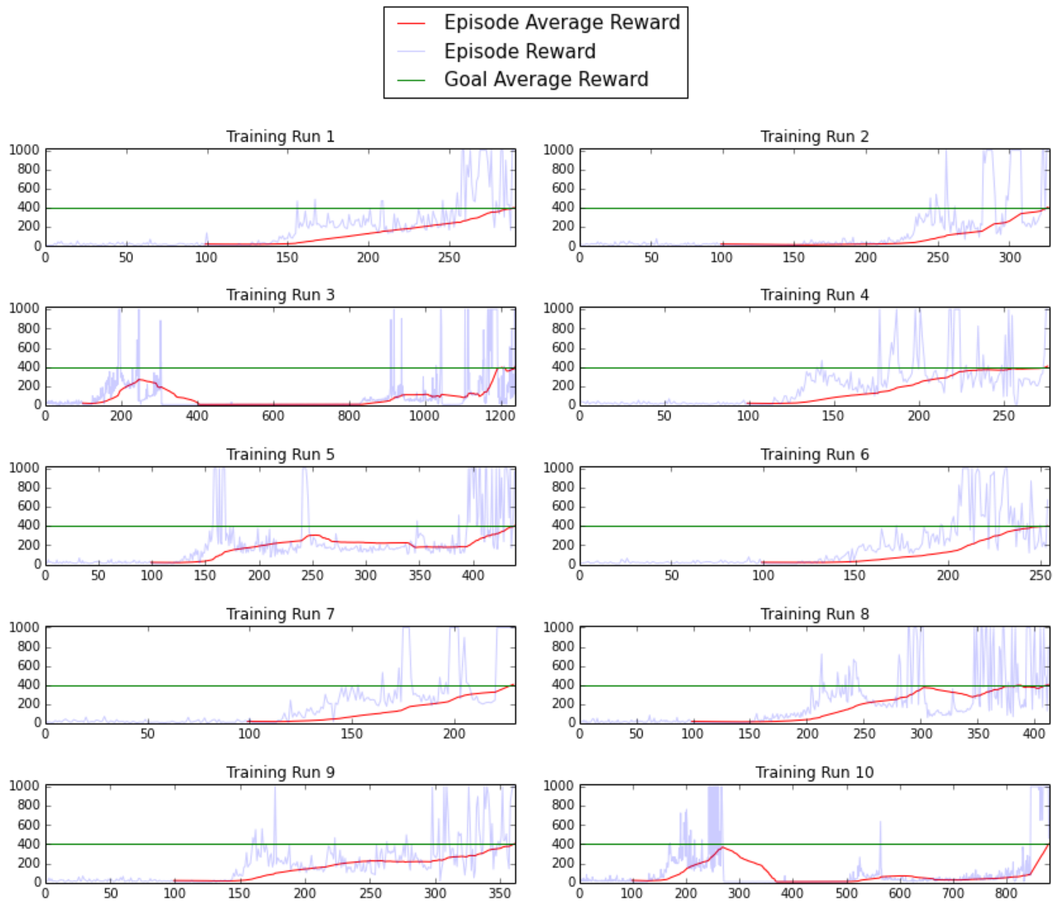

Figure 2: Results for DQN on Fully Observable Cart Pole

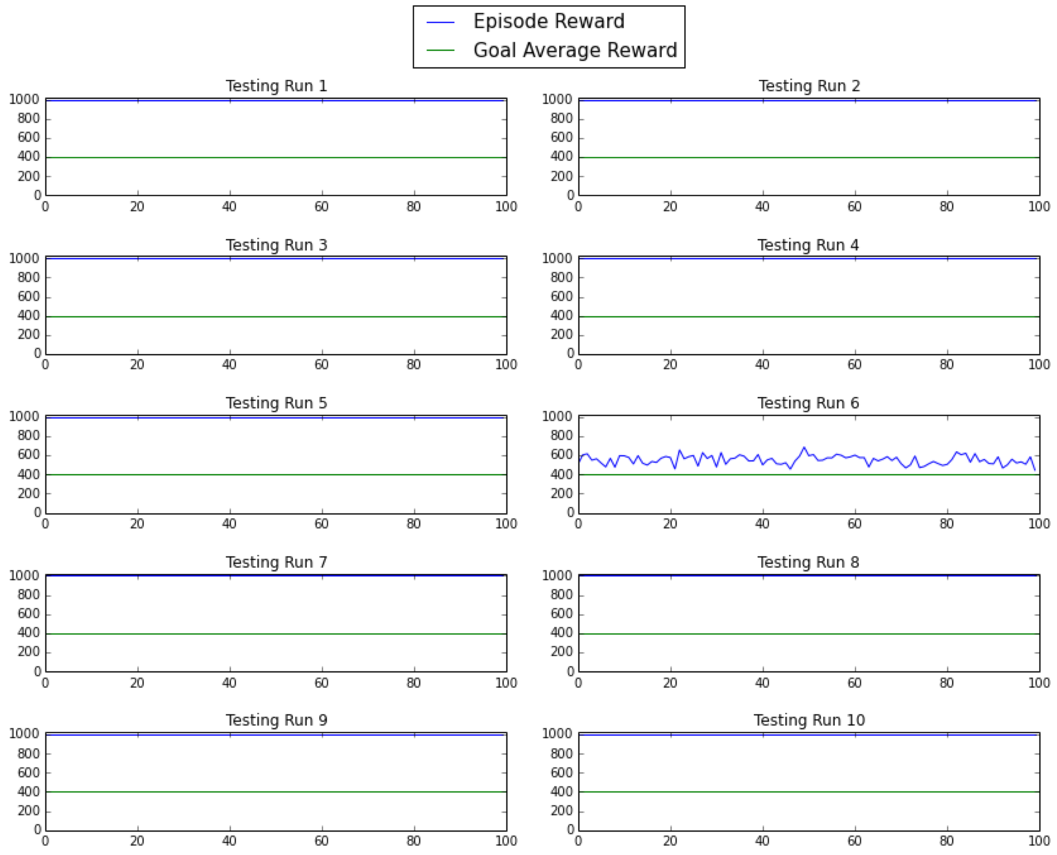

Figure 3: Test Runs for DQN Fully Observable Cart Pole

Average Test Scores				
Run 1	Run 2	Run 3	Run 4	Run 5
1000.0	1000.0	1000.0	1000.0	1000.0
Run 6	Run 7	Run 8	Run 9	Run 10
551.9	1000.0	1000.0	1000.0	1000.0

8.1.2 DQN Partially Observable

In figure 4, the results for the partially observable modified Cart Pole problem are shown for the DQN algorithm. All ten training runs fail to achieve the goal average. In the testing episodes of figure 5, all ten runs fail to achieve the goal average.

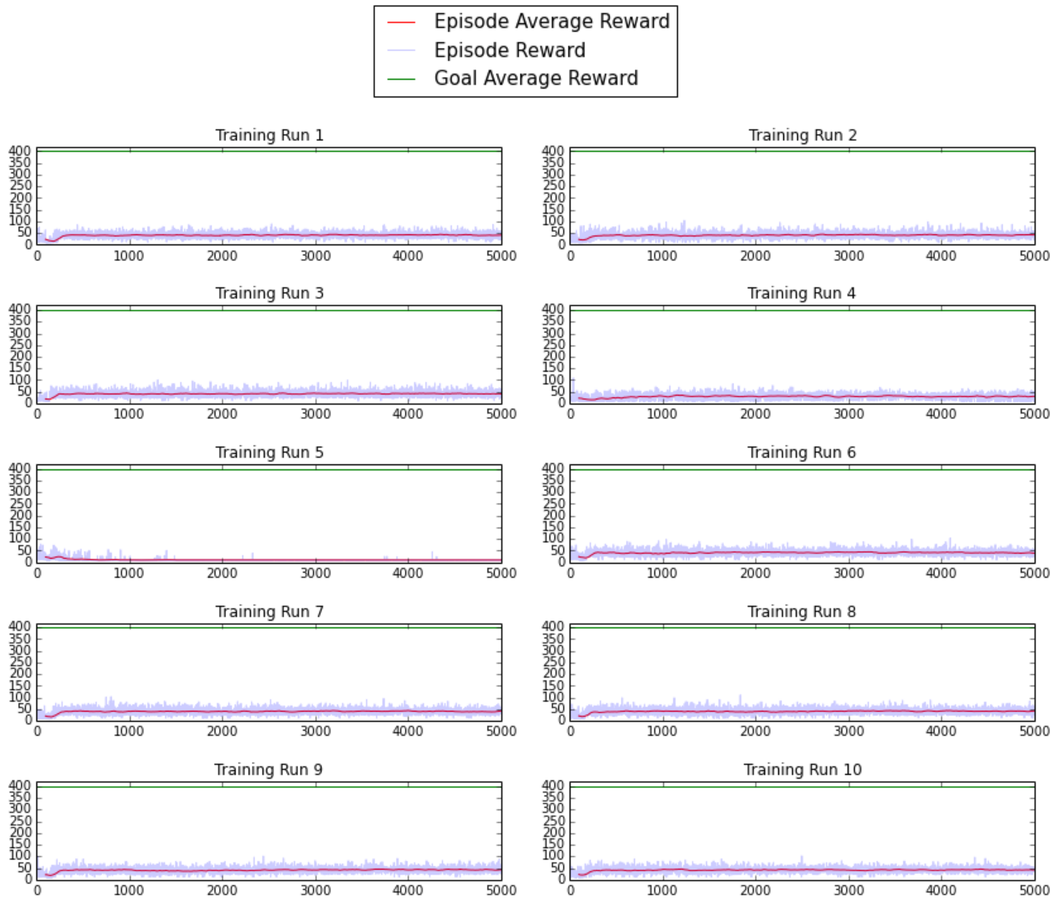

Figure 4: Results for DQN on Partially Observable Cart Pole

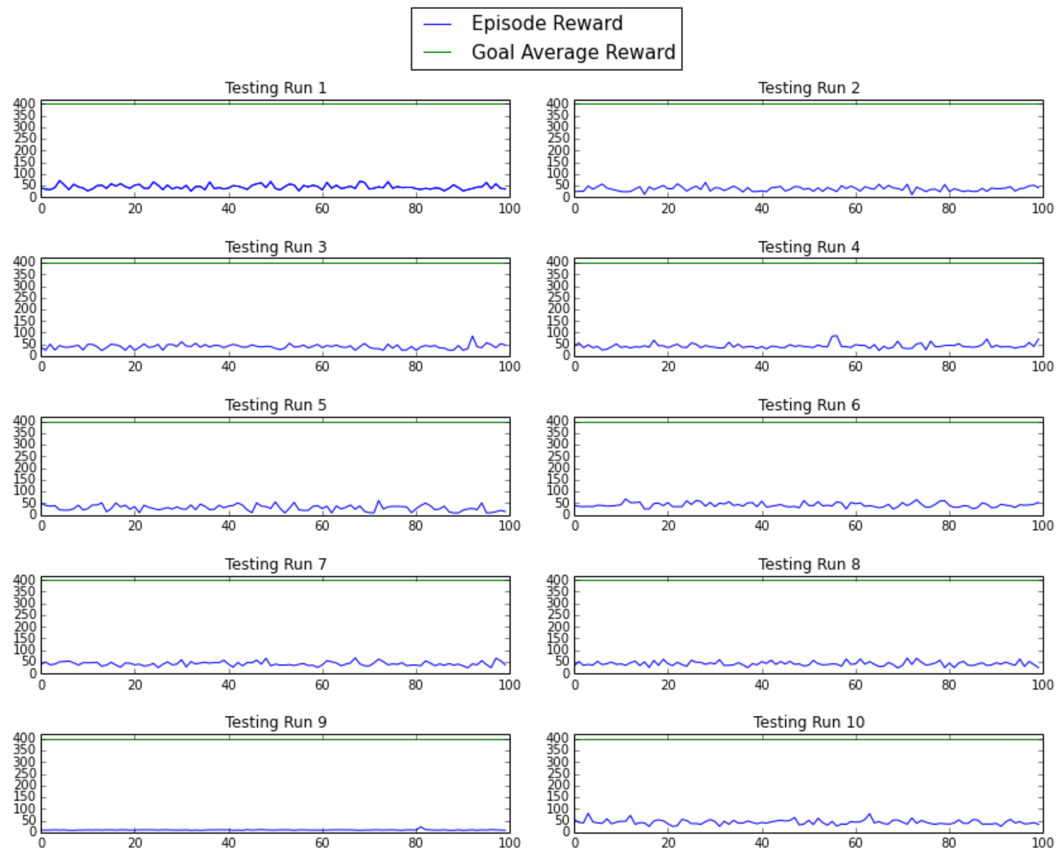

Figure 5: Test Runs for DQN Partially Observable Cart Pole

Average Test Scores				
Run 1	Run 2	Run 3	Run 4	Run 5
44.8	42.3	43.3	36.8	9.5
Run 6	Run 7	Run 8	Run 9	Run 10
42.1	40.6	41.7	29.6	43.0

8.2 DQRN Algorithm

In this section, results for the Deep Recurrent Q-Learning algorithm will be displayed.

8.2.1 DRQN Fully Observable

In figure 6, the results for the fully observable unmodified Cart Pole problem are shown for the DRQN algorithm. In all ten training runs, the average reward made it to the goal reward. For the test results in figure 7, the agent was able to score the maximum value of 1000 on 4 out of the 10 results. However, some of the remaining were very close to the maximum. For the other 6 out of 10 results, the average agent score was well above the goal of 400.

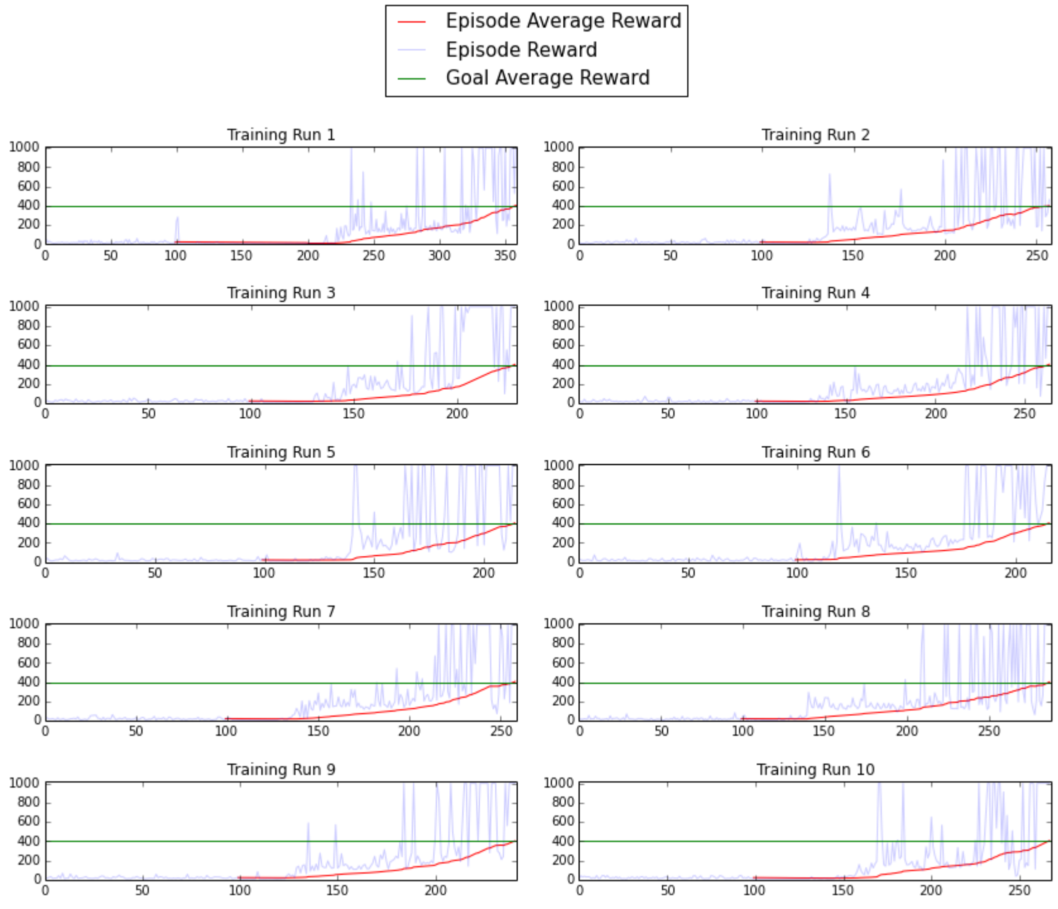

Figure 6: Results for DRQN on Fully Observable Cart Pole

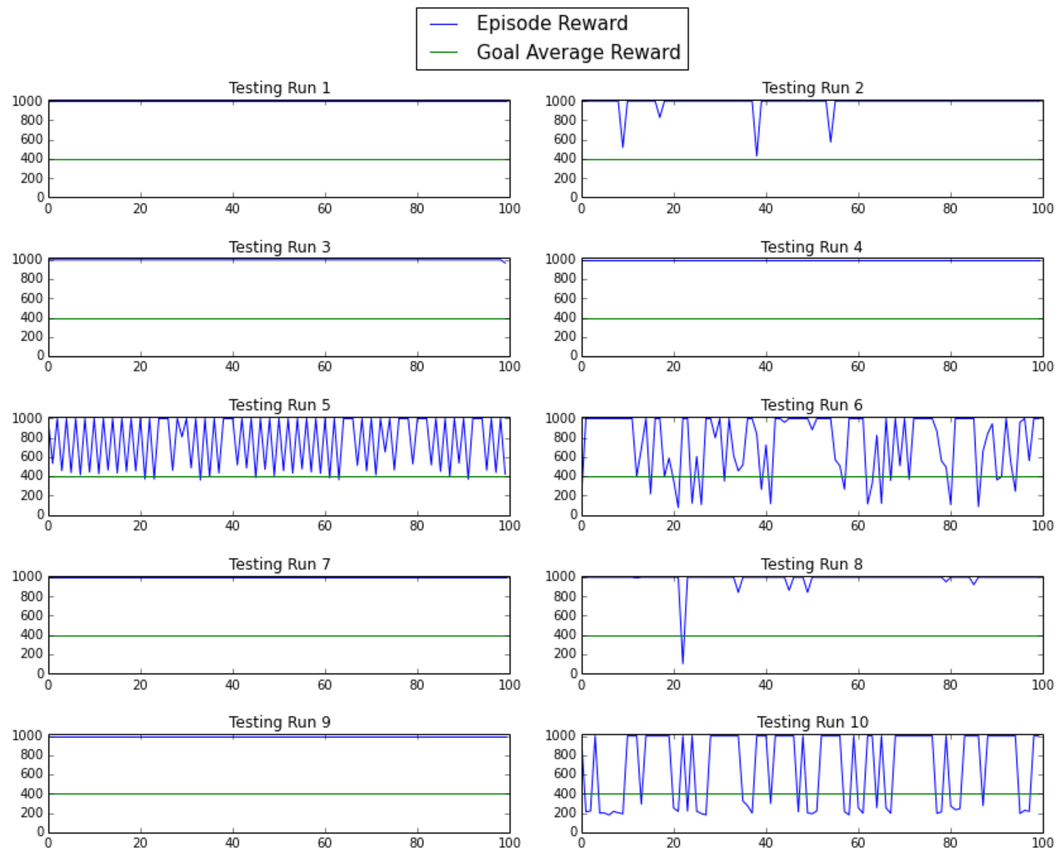

Figure 7: Test Runs for DRQN Fully Observable Cart Pole

Average Test Scores				
Run 1	Run 2	Run 3	Run 4	Run 5
1000.0	758.2	1000.0	983.5	1000.0
Run 6	Run 7	Run 8	Run 9	Run 10
985.1	999.7	1000.0	761.1	697.3

8.2.2 DRQN Partially Observable

In figure 8, the results for the partially observable modified Cart Pole problem are shown for the DRQN algorithm. In all ten training runs, the average reward made it to the goal reward. For the test results in figure 9, the agent was able to score above the average goal value on 6 out of 10 of the results.

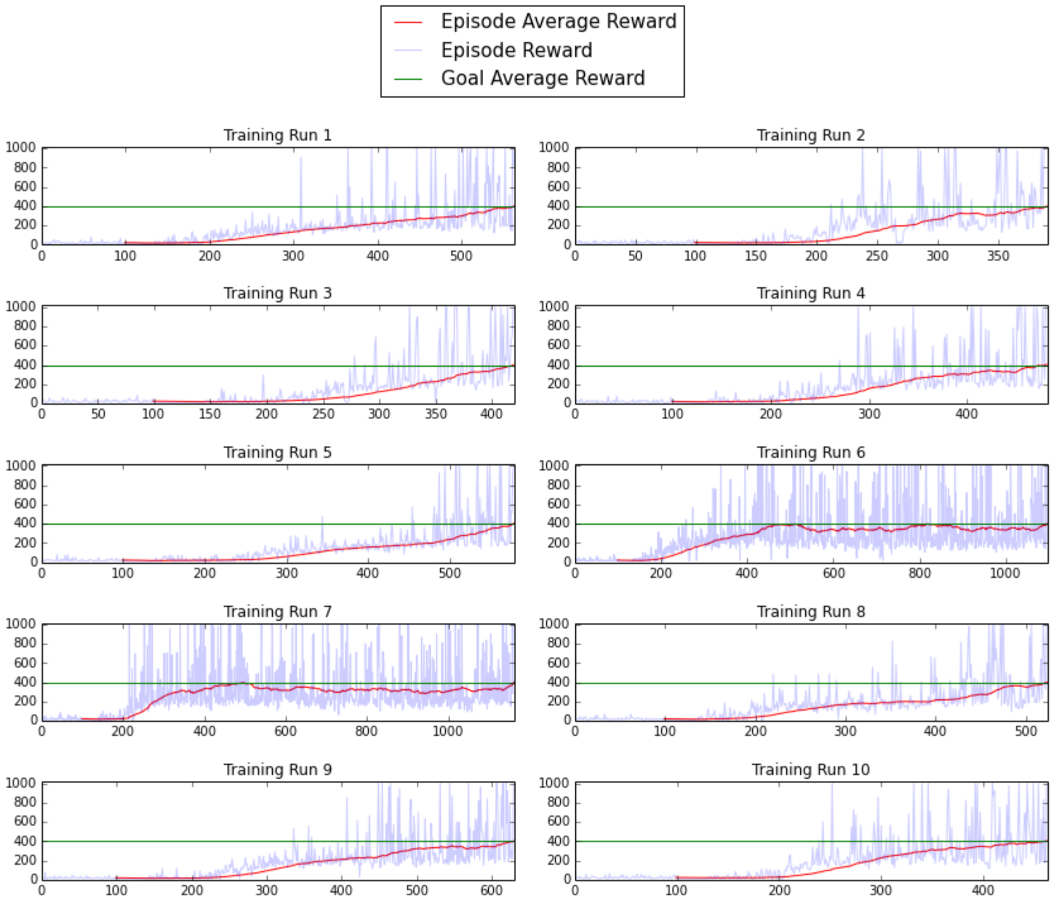

Figure 8: Results for DRQN on Partially Observable Cart Pole

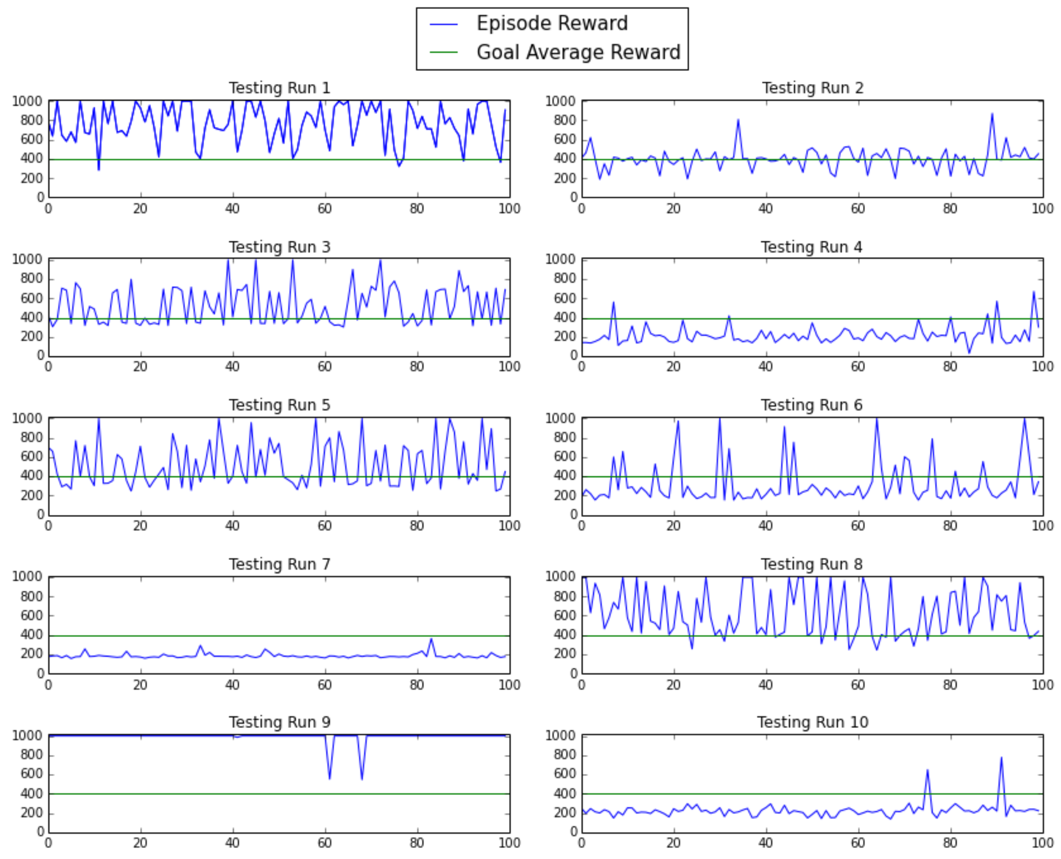

Figure 9: Test Runs for DRQN Partially Observable Cart Pole

Average Test Scores				
Run 1	Run 2	Run 3	Run 4	Run 5
767.1	317.1	217.0	402.1	990.8
Run 6	Run 7	Run 8	Run 9	Run 10
622.7	525.4	183.7	518.1	227.8

9 Analysis

In this section, the results are analyzed. One of the main goals of this project was to show the the DRQN algorithm could solve a partially observable problem where the DQN algorithm would fail. In this section, that result will be shown.

In this section, the best run of each algorithm will be used for comparison. The best run consists of the run with the highest test average with ties broken randomly. The training and test values will be used for comparison purposes.

In figure 10, the training values for the best runs are shown. For the fully observable Cart Pole problem, both algorithms achieved the goal. From figure 11, both algorithms achieved the maximum test value for the problem.

For the partially observable problem, the difference is greater. In the DQN version in figure 10, the algorithms was never able to reach the training goal. Also, the best testing value was a small value of 44.8. To summarize the data, the DQN algorithm completely fails on the partially observable Cart Pole problem. Referring back to figure 5, it can be seen that the algorithm failed on all 10 out of 10 runs. However, the DRQN algorithm reached the training goal and had a maximum test value of 990.8. As can be seen from the DRQN partially observable test runs in figure 9, the DRQN algorithms tested successfully in 6 out of 10 runs.

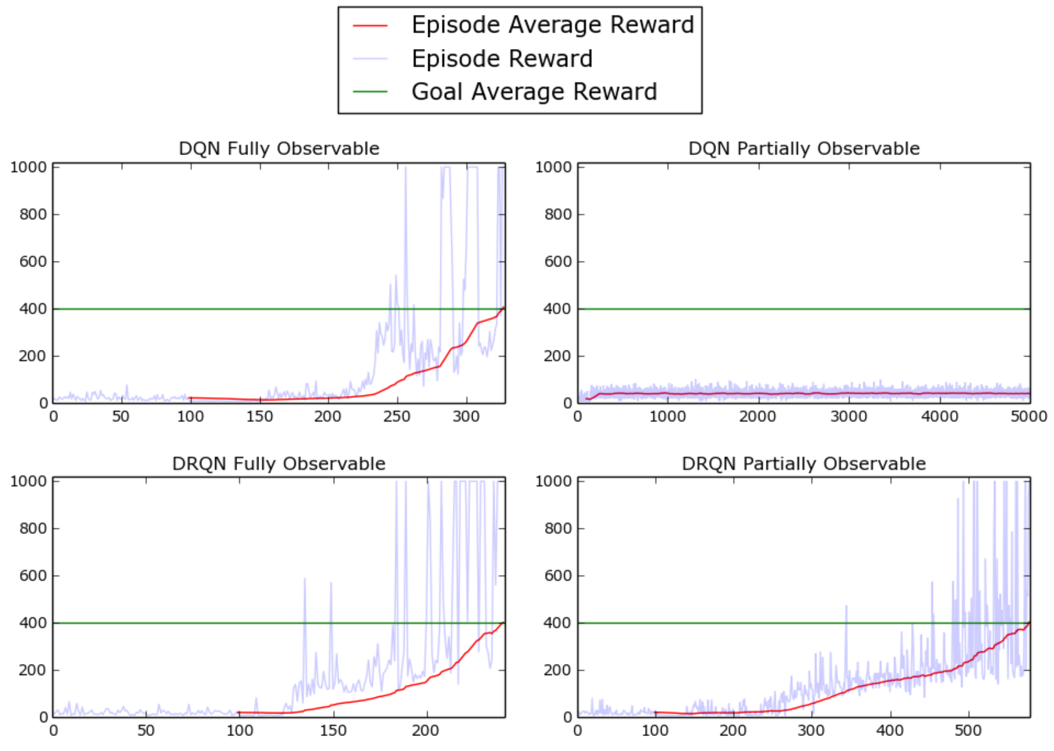

Figure 10: Analysis Training Comparison

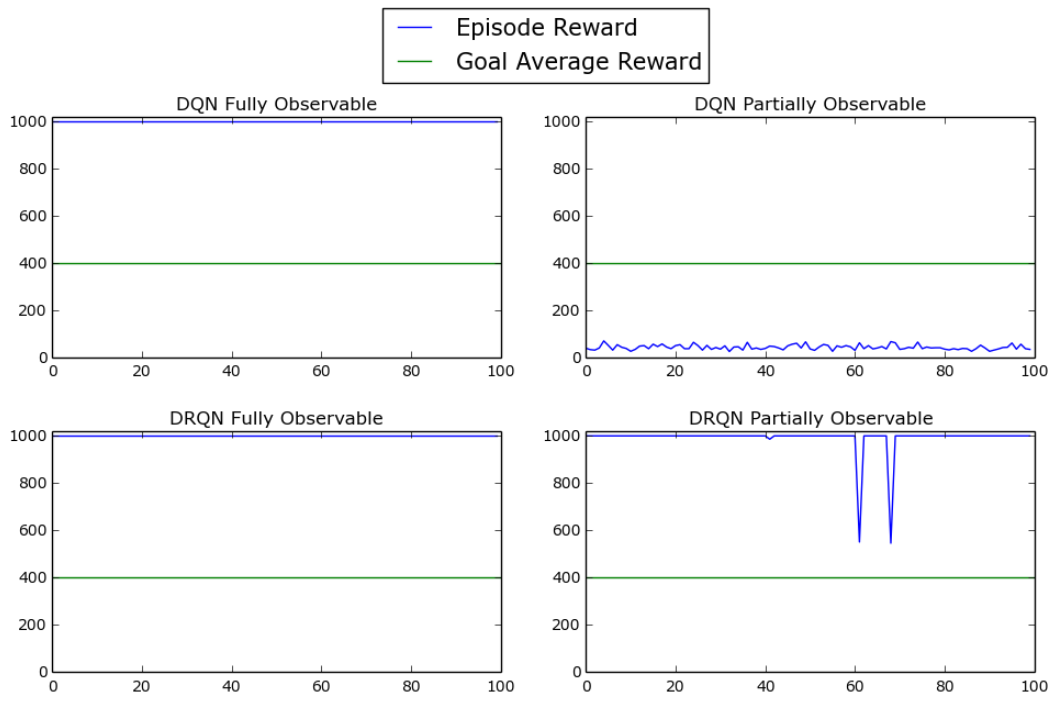

Figure 11: Analysis Testing Comparison

Maximum Average Test Scores	
DQN Full	DQN Partial
1000.0	44.8
DRQN Full	DRQN Partial
1000.0	990.8

In summary, it was shown that the DQN algorithm is unable to successfully solve the given partially observable problem. However, the DRQN algorithm was able to solve the partially observable problem on 6 out of the 10 runs. For the remainging 4 out of 10 runs, the value was below the goal, but still well above the DQN values. This shows that given a partially observable environment, the DRQN algorithm would be a better choice over the DQN algorithm.

10 Conclusion and Future Work

In this project, the main goal was to show that a recurrent LSTM network can aid in Deep Q-Learning depending on how the problem you are trying to solve is constructed. The DRQN algorithm successfully solved the partially observable problems where the DQN algorithm failed.

This was easy to test using the OpenAI Gym test bench problems. However, it can be more difficult in other problems where the observations aren't as clearly defined. When desiging other problems, choosing the correct observation features might not always be an easy task. Using the DRQN algorithm could help with this problem by reducing the number of observation features that need to be used.

Two areas of future work are planned for the use of this project. The first being drone target tracking in the Robotics Operating System with the Gazebo Simulator which is currently under development. The second part is the incorporation of images and convolutional neural networks like the original papers used. This second part requires more computation power, so it will be delayed until appropriate hardware is available.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[2] Matthew J. Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *CoRR*, abs/1507.06527, 2015.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.