Comparison of Deep Reinforcement Learning Algorithms in Partially Observable Environments

COS 703 Final Project

Daniel Lucas Thompson

Goals

Goals

- Convert the Deep Q-Learning algorithm to use features instead of images
- Test the Deep Q-Learning algorithm on fully observable and partially observable problems
- Test the Deep Recurrent Q-Learning algorithm on fully observable and partially observable problems
- Compare the results of each algorithm to determine which works best for the partially observable environment

Tools Used

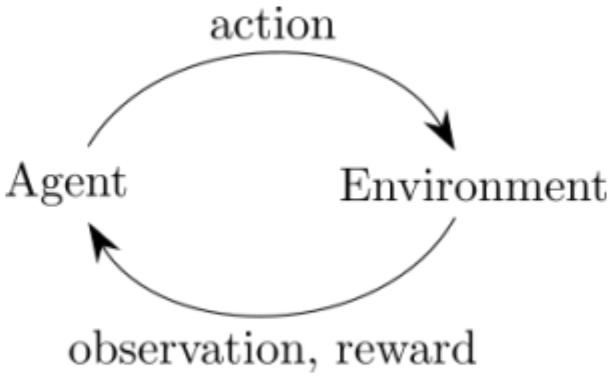
Tools Used

1. Python
2. OpenAI Gym framework
3. Keras Deep Learning library
4. Jupyter Notebooks
5. Pandas
6. Matplotlib

Reinforcement Learning

Reinforcement Learning

1. Agent receives a starting observation
2. Agent decides on an action and sends it to the environment
3. Agent receives a reward from the environment and a new observation
4. Continues until a stopping condition is met

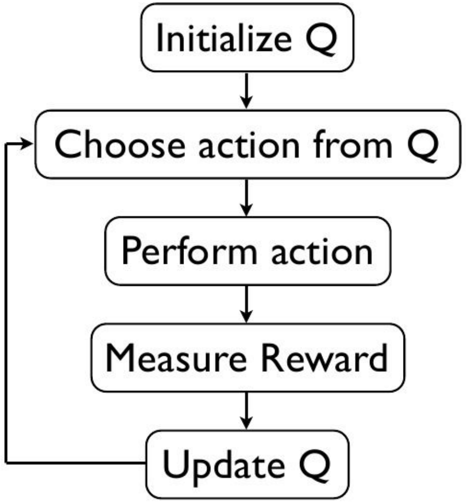


Q-Learning

Q-Learning

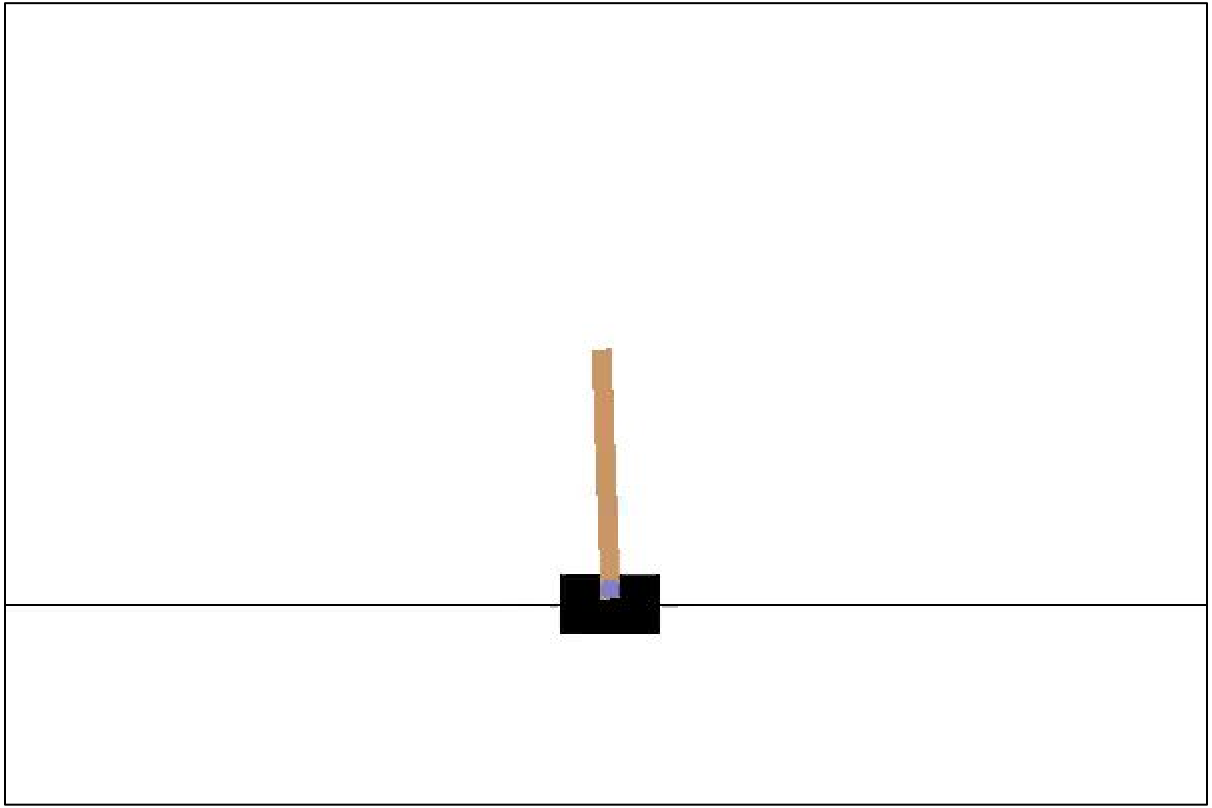
$$Q(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha_t}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t,a_t)}_{\text{old value}} \right)$$

1. Initialize the values
2. Choose and perform an action
3. Use the received reward to update the Q-Values
4. Repeat until convergence

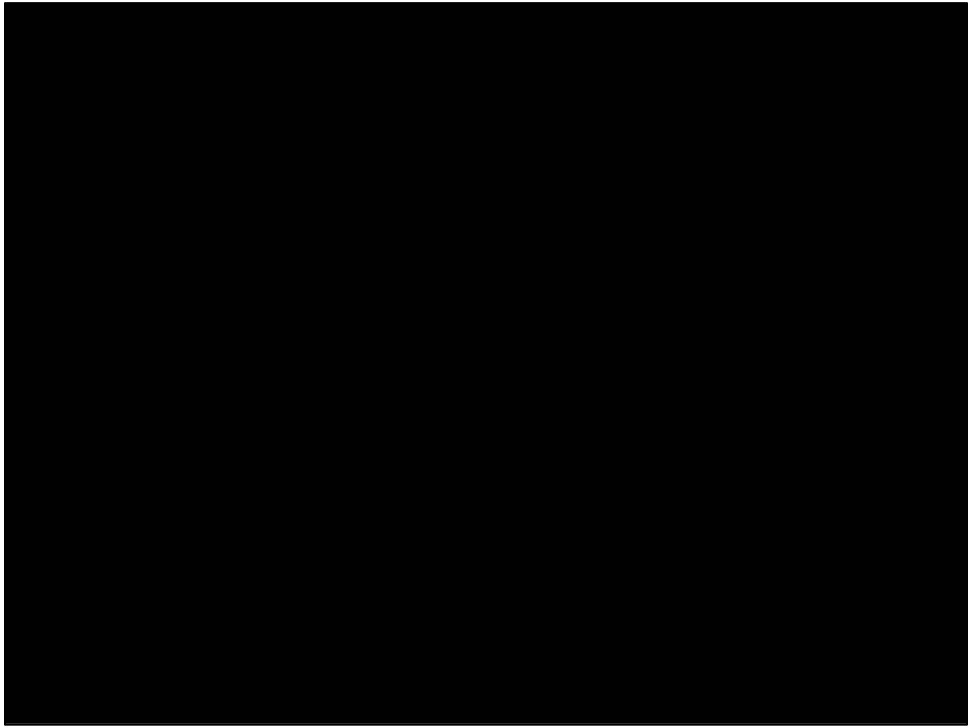


Problem and Data

Cart Pole Problem



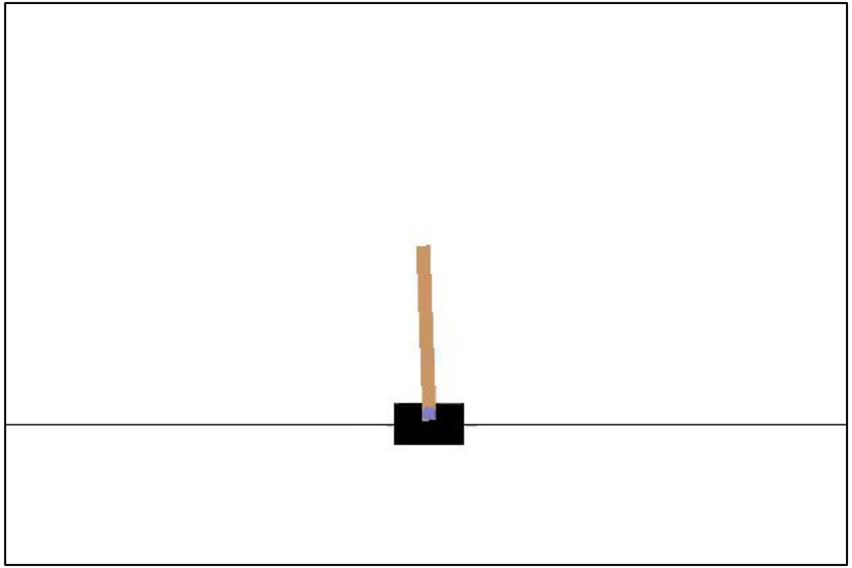
Cart Pole Problem



Cart Pole Fully Observable Problem

Observation	**Min**	**Max**
Cart Position	-2.4	2.4
Cart Velocity	-Inf	Inf
Pole Angle	-41.8°	41.8°
Pole Velocity at Tip	-Inf	Inf

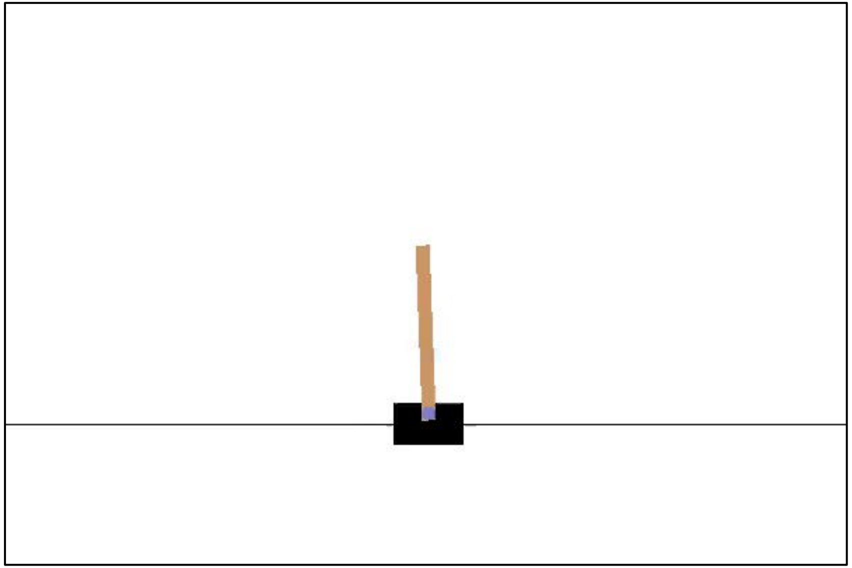
Action
Push cart left
Push cart right



Cart Pole Partially Observable Problem

Observation	Min	Max
Cart Position	-2.4	2.4
Pole Angle	-41.8°	41.8°

Action
Push cart left
Push cart right



Deep Q-Learning (DQN)

Deep Q-Learning (DQN)

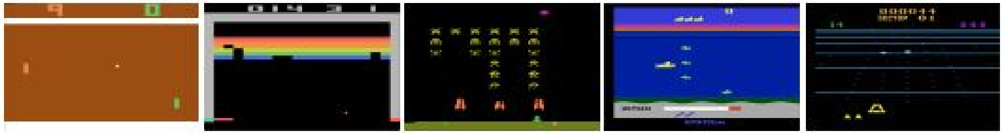
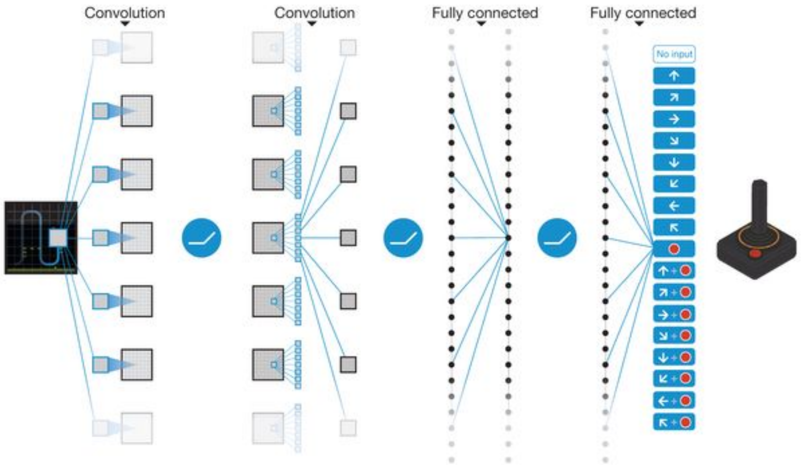


Figure 1: Screen shots from five Atari 2600 Games: (*Left-to-right*) Pong, Breakout, Space Invaders, Seaquest, Beam Rider



Created by Google Deep Mind

Learns to play Atari games with only image inputs using Convolutional Networks

Plays at or above human level on 29 out of 49 tested Atari games

Deep Q-Learning (DQN)

1. Uses a deep neural network to approximate the Q function
2. Uses replay memory to simplify training, but trains in an online fashion
3. Uses target networks to aid in training stability
4. Uses ε-greedy exploration to aid in fully exploring the environment

Algorithm 1 Deep Q-Learning

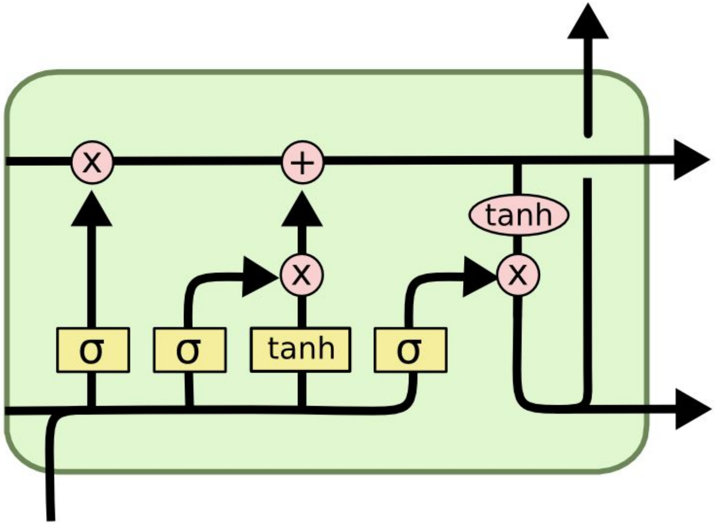
Initialize the Replay Memory D
Initialize the Q network with random weights
$s \leftarrow$ the initial problem state
repeat
  $a \leftarrow$ random with probability ϵ, or $argmax_a Q(s,a)$
  $s' \leftarrow$ perform action a in the environment to get s'
  $r \leftarrow$ reward from performing a in s
  $D \leftarrow$ append (s, a, r, s')
  $B \leftarrow$ sample a batch from replay memory D
  for each transaction in batch B **do**
    if s is the terminal state **then**
      $target \leftarrow r$
    else
      $target \leftarrow r + \lambda max_{a'} Q(s', a')$
    end if
  end for
  Train the network using $(targets - Q(s,a))^2$ as loss
  $s \leftarrow s'$
until average reward reached

Deep Recurrent Q-Learning (DRQN)

Deep Recurrent Q-Learning (DRQN)

1. Identical to DQN with two exceptions
2. Replaces the last feed forward neural network layer with a Long Short Term Memory (LSTM) layer
3. Modification of Replay Memory to use transactions in sequential order

LSTM Memory Cell



Results

Results Overview

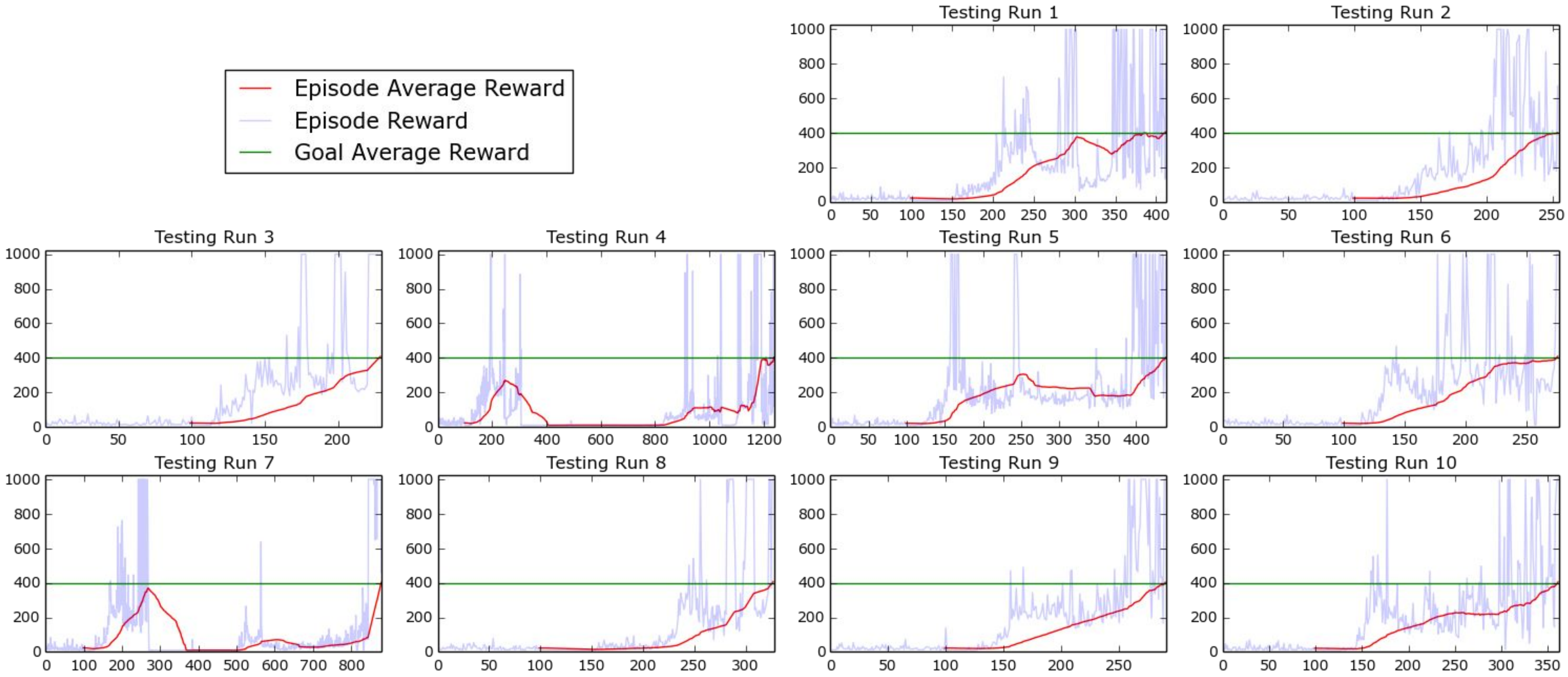
For Each Algorithm and Problem

- 10 Training Runs
- 10 Testing Runs

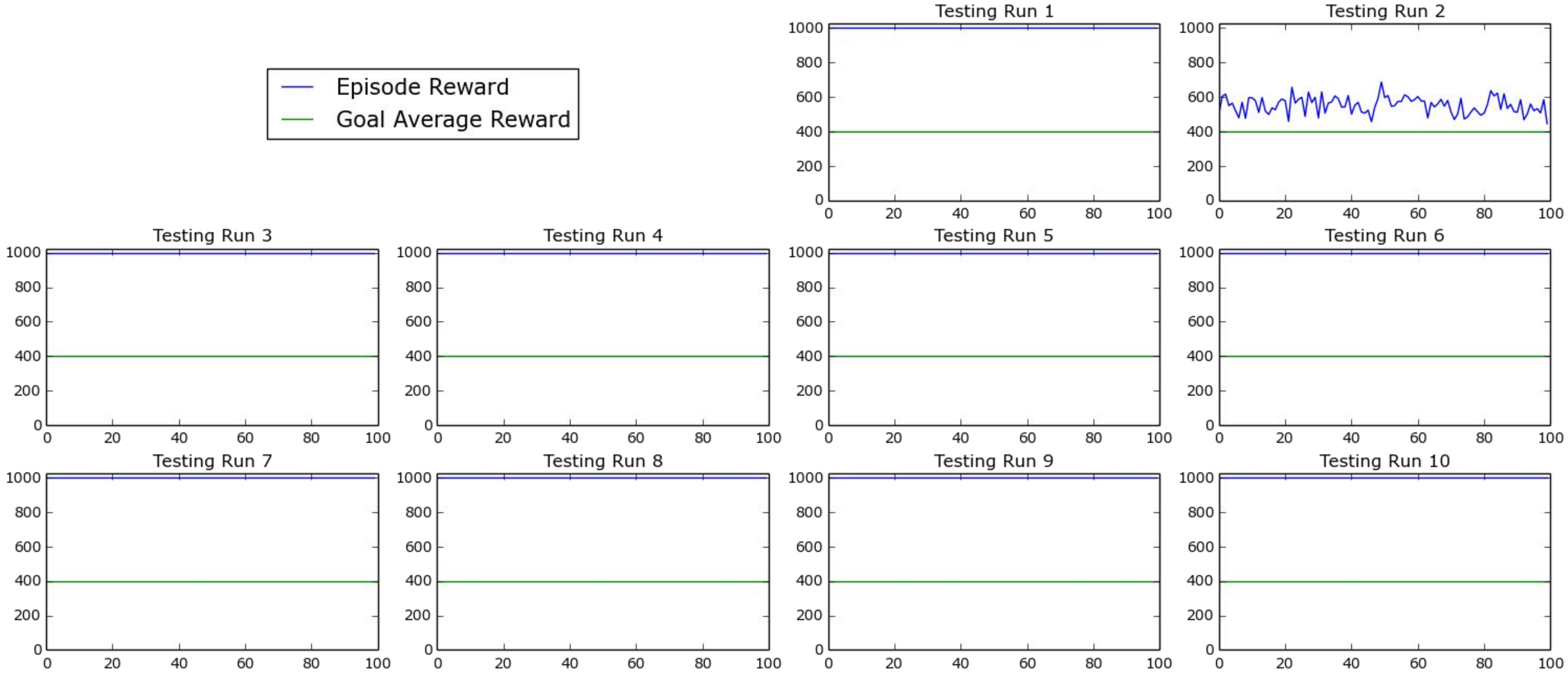
Algorithm and Problem

1. DQN Fully Observable
2. DRQN Fully Observable
3. DQN Partially Observable
4. DRQN Partially Observable

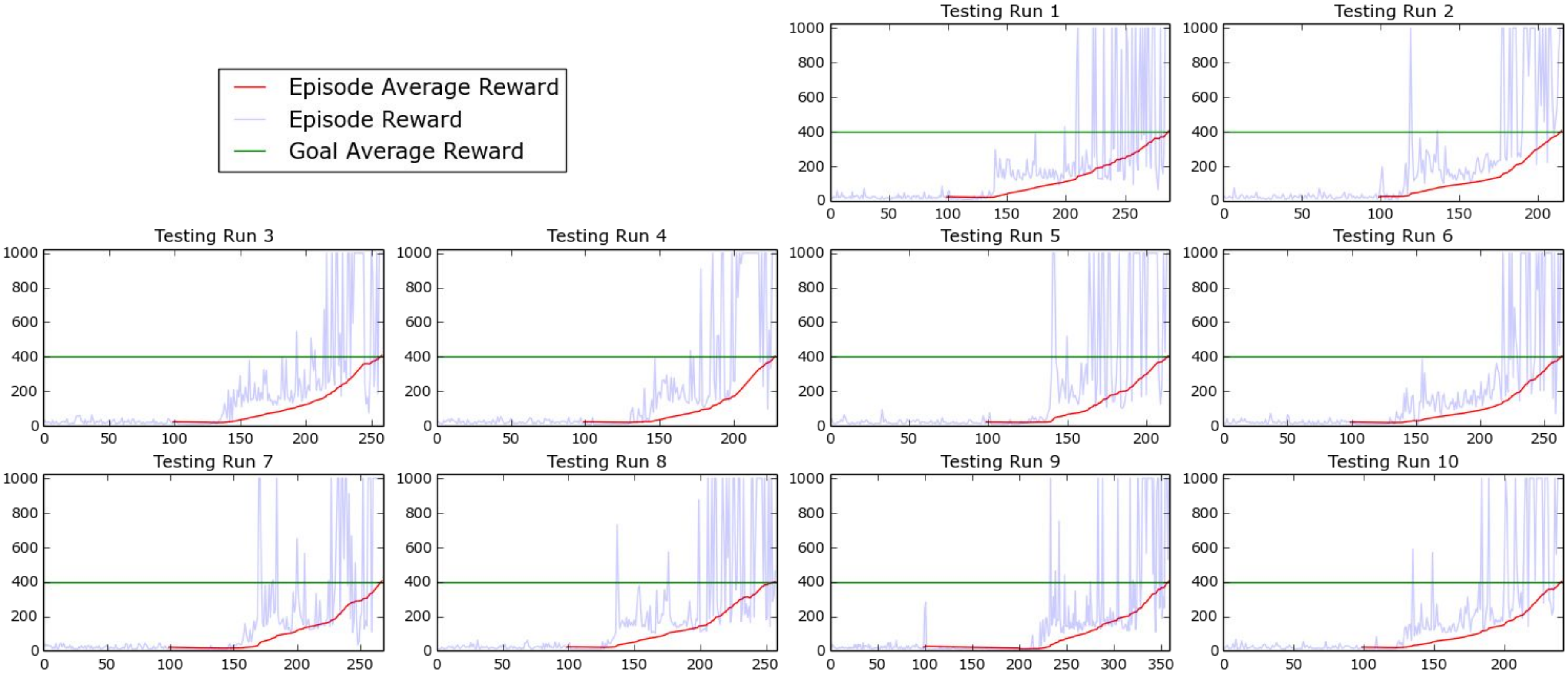
DQN Fully Observable Training



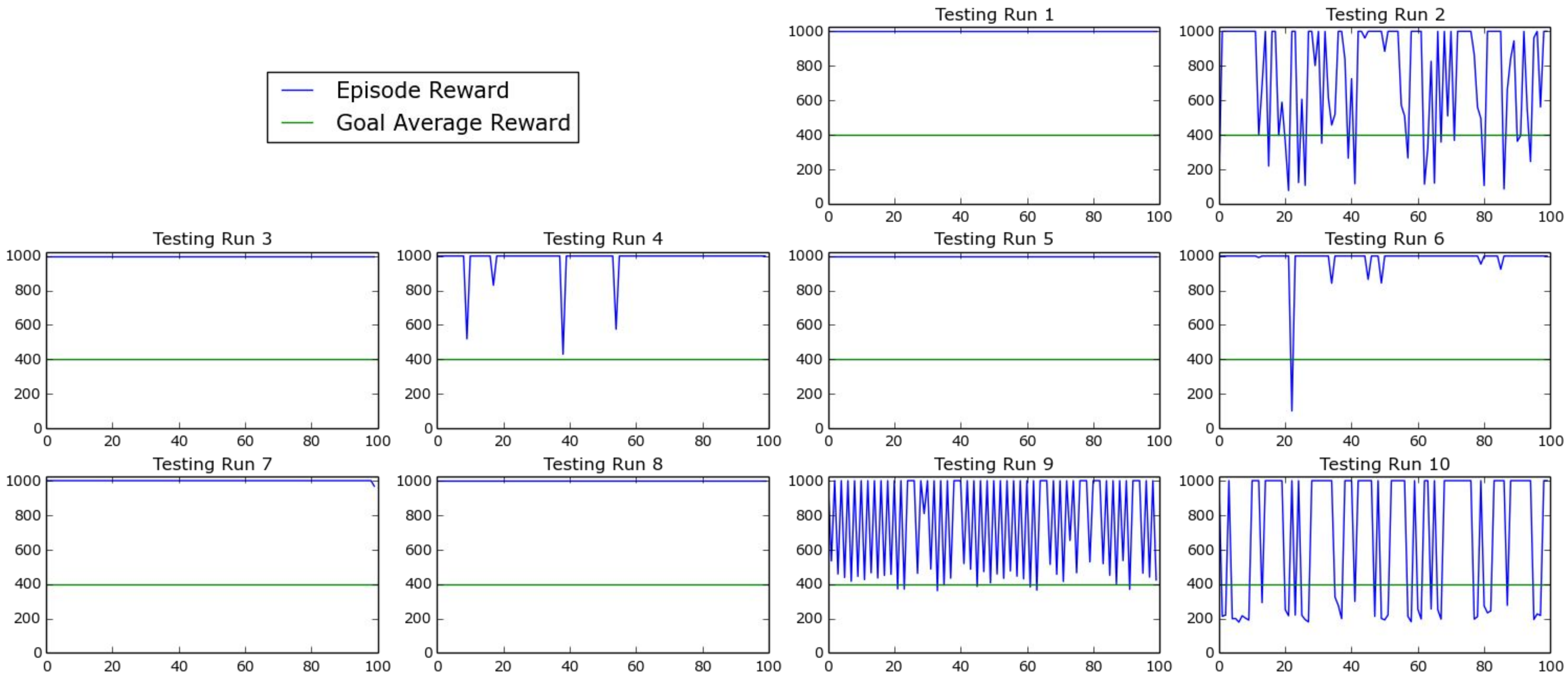
DQN Fully Observable Testing



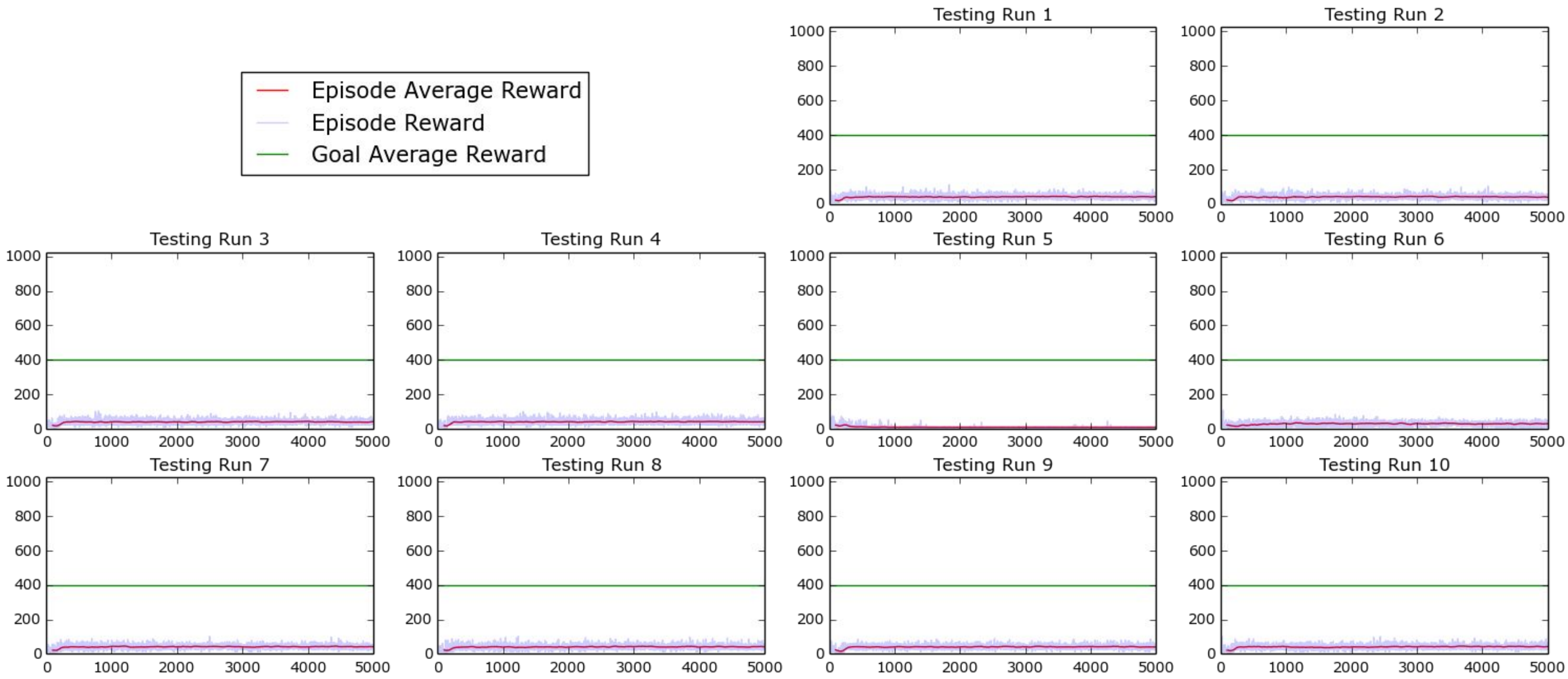
DRQN Fully Observable Training



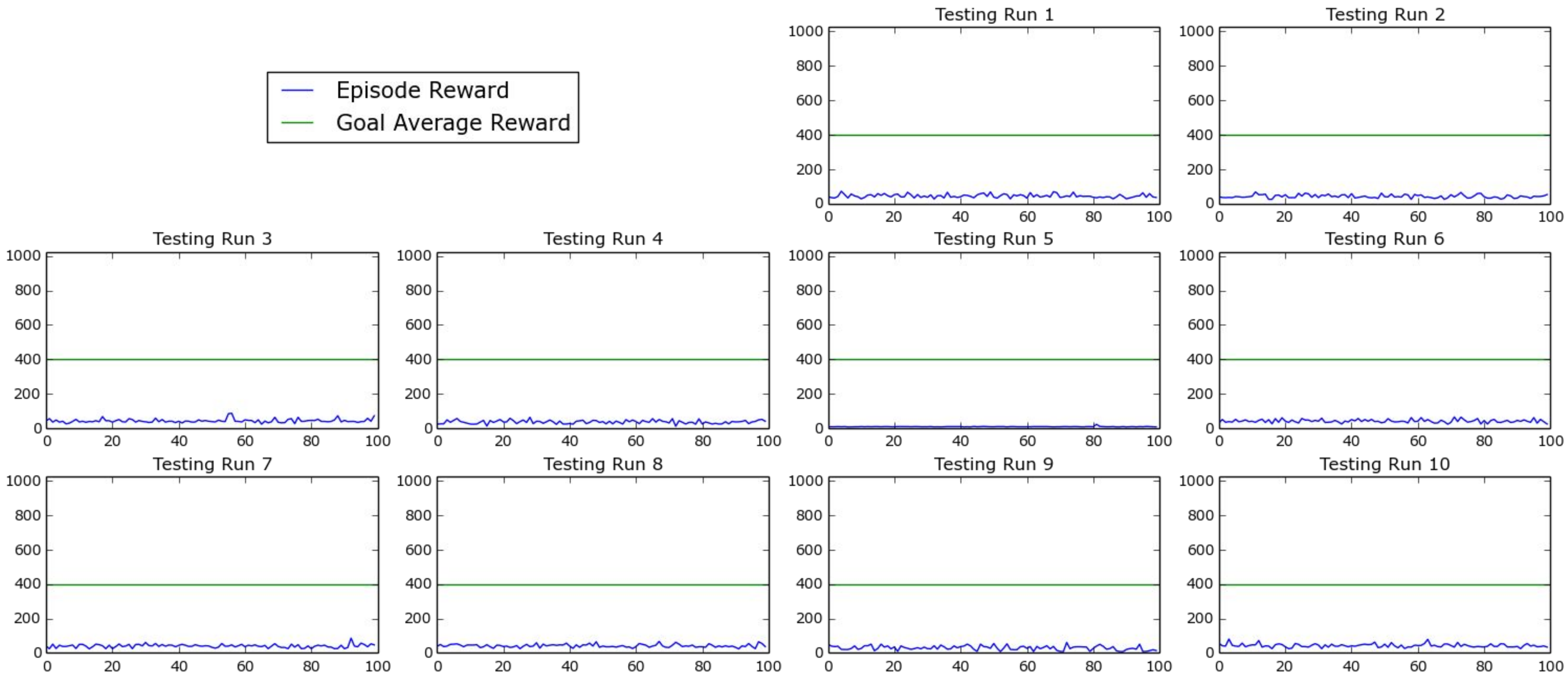
DRQN Fully Observable Testing



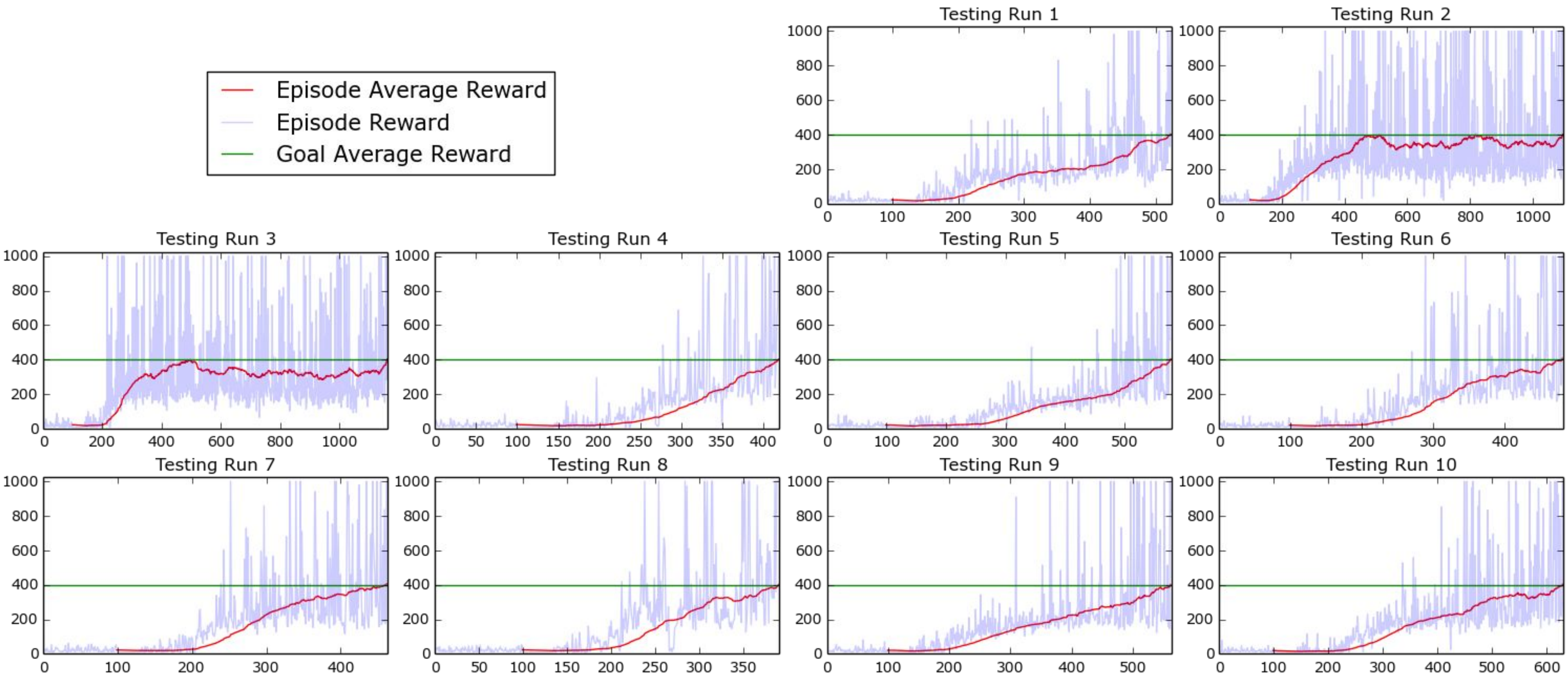
DQN Partially Observable Training



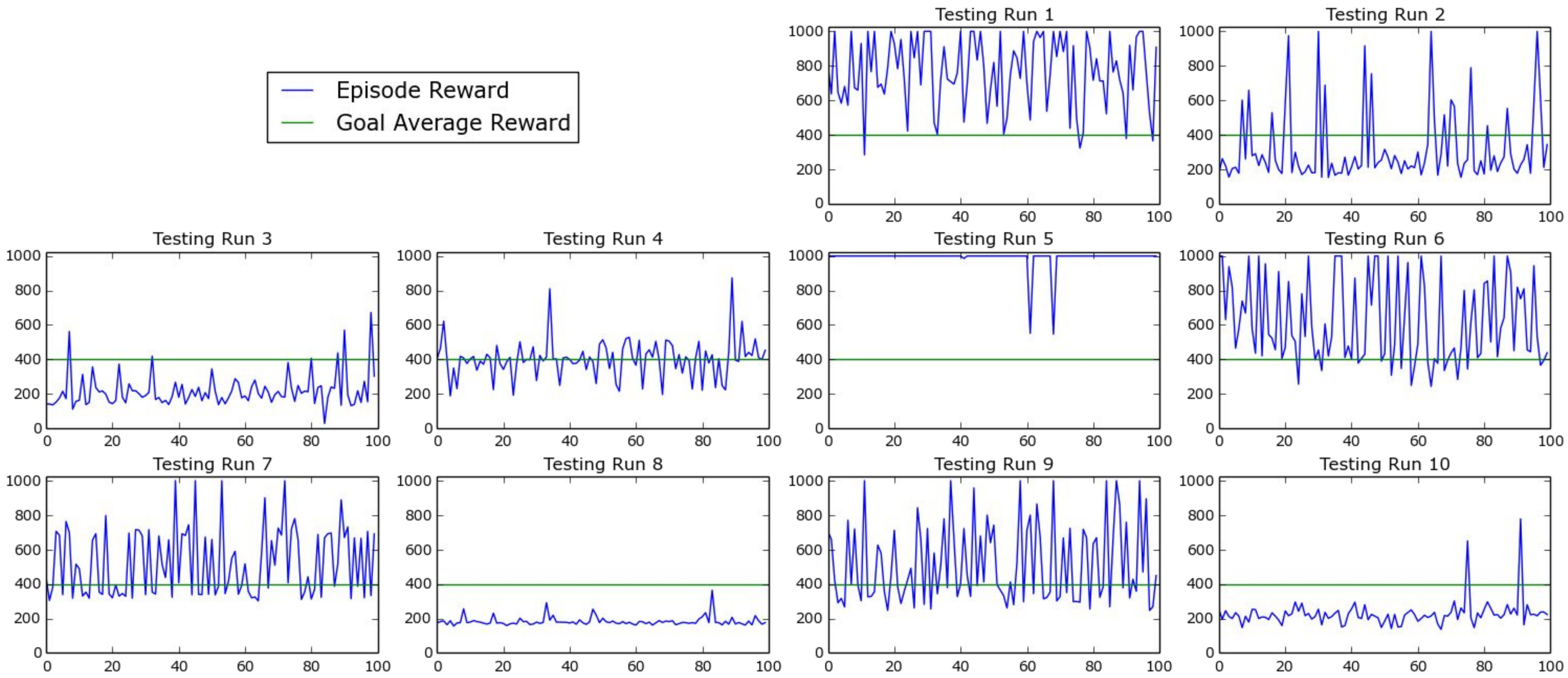
DQN Partially Observable Testing



DRQN Partially Observable Training

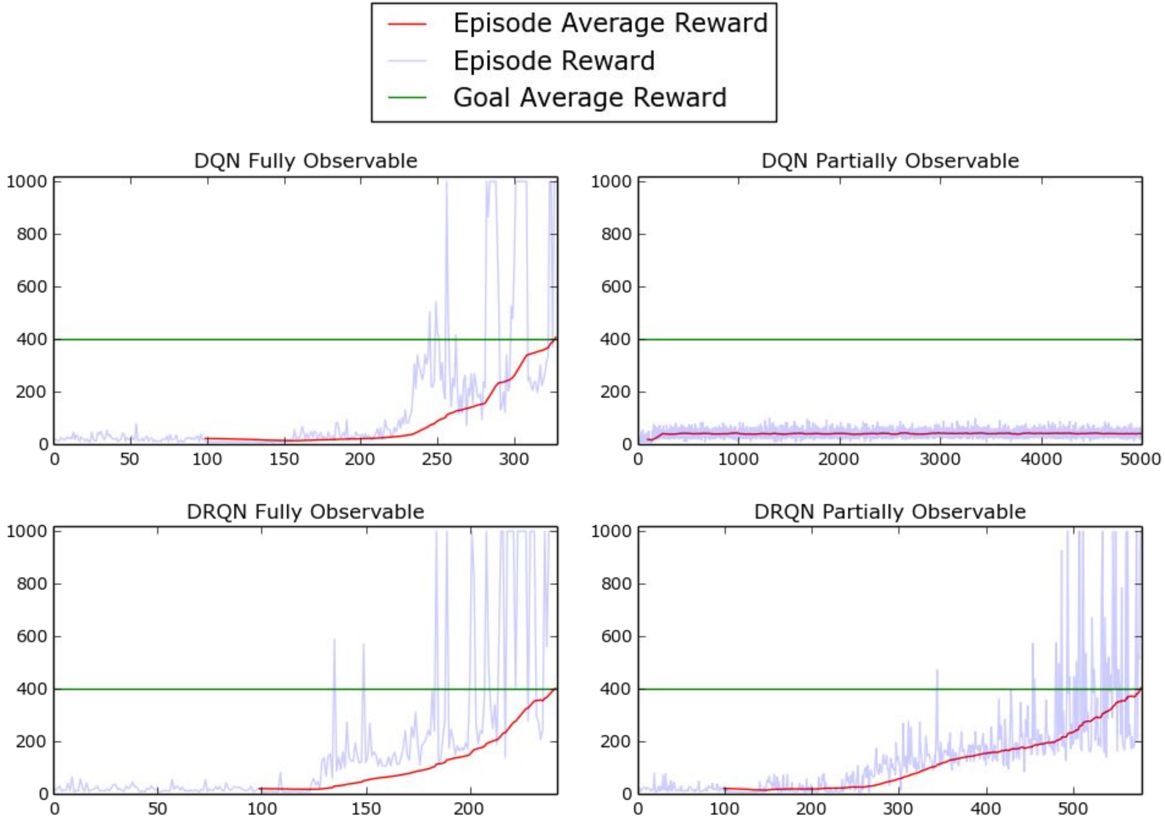


DRQN Partially Observable Testing

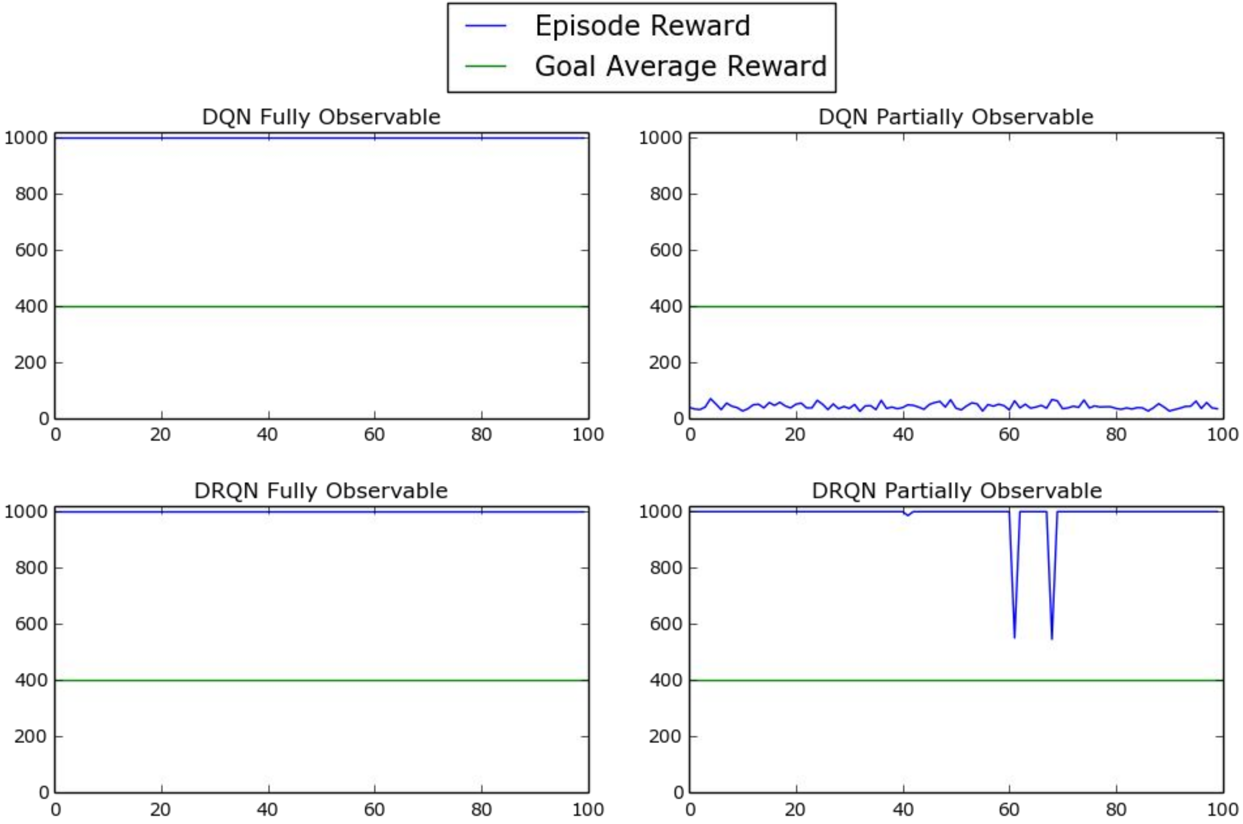


Analysis

DQN vs DRQN Training



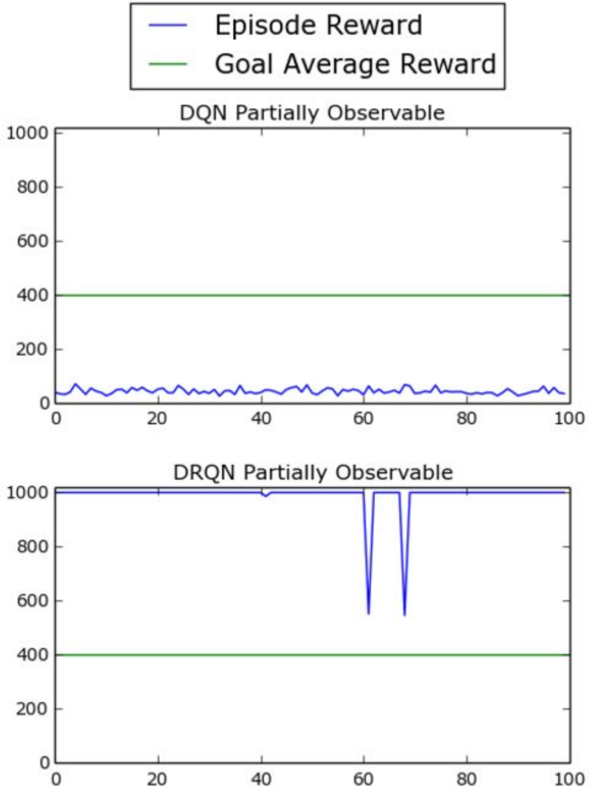
DQN vs DRQN Testing



Conclusion

Conclusion

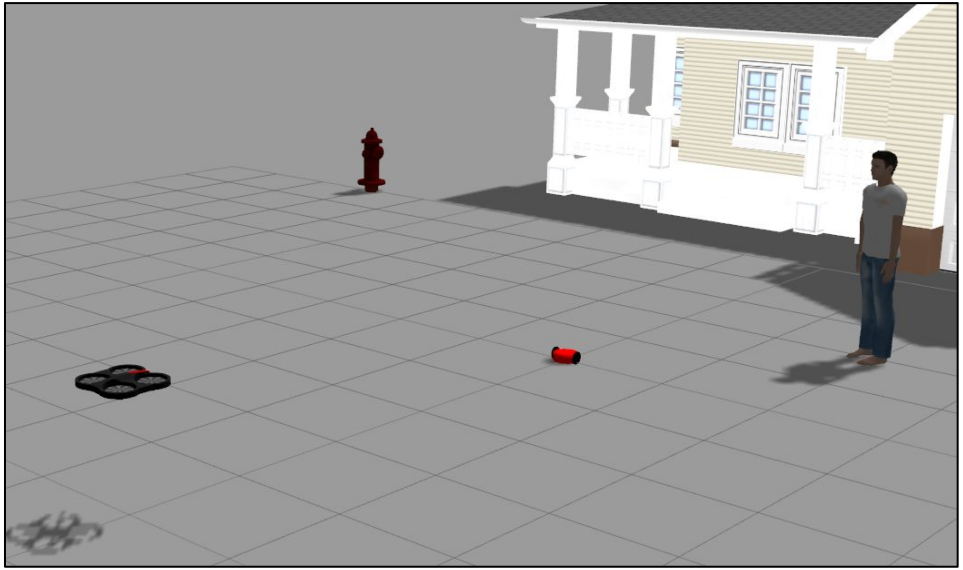
1. DQN performs poorly in partially observable environments
2. DRQN can obtain the goal in partially observable environments
3. DRQN is much harder to train and find correct parameters



Future Work

Future Work

Drone Target Tracking - using the DQN and DRQN algorithms, train drones in a simulator to follow a moving target



Convolutional Neural Networks - modify the drone target tracking to use convolutional neural networks when appropriate hardware is available for training

References

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

[2] Matthew J. Hausknecht and Peter Stone. Deep recurrent q-learning forpartially 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.