Patch Leaving Decisions

Final Project Presentation

Group-1

Participants: Neelabh(190538), Romit(190720), Nibir(190545)
Instructor: Dr. Ashutosh Modi
Mentors: Akhilesh Tayade, Subham Hota
TA: Aishwarya Gupta

Introduction

- Foraging effectively is critical to the survival of all animals and this imperative is thought to have profoundly shaped brain evolution.
- Decision tasks inspired by foraging have increasingly attracted the attention of neuroscientists, in part because the ecological importance of foraging is thought to have profoundly shaped neural decision making systems.
- Decisions made by foraging animals often approximate optimal strategies, but the learning and decision mechanisms generating these choices remain poorly understood.
- In this project, we follow a reinforcement learning based approach to understand the underlying decision mechanisms of solitary foraging.

https://en.wikipedia.org/wiki/Foraging, https://www.jstor.org/stable/4308969,
https://pubmed.ncbi.nlm.nih.gov/28918312/,

The Stay or Leave Decision

- In our problem we are mainly interested in the decision behind an animal staying at the current depleting patch or leaving in expectation of a more rewarding patch.
- This type of sequential stay-or-leave decisions exist almost everywhere like whether to stay at a job or leave it, continue pursuing a line of enquiry or leave it, continue investing in a mutual relationship or not, etc.
- Most decision research concerns choice between simultaneously presented options, in many situations options are encountered serially, and the decision is whether to exploit an option or search for a better one.

https://en.wikipedia.org/wiki/Foraging, https://www.jstor.org/stable/4308969,
https://pubmed.ncbi.nlm.nih.gov/28918312/,

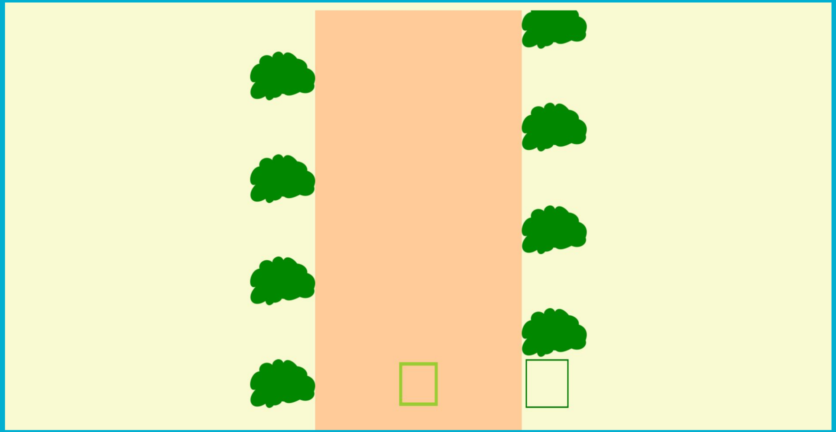
Research in this Field

- A paper submitted in 2017 by Nils Koling and Thomas Akam suggested that model-based reinforcement learning may provide a framework for understanding foraging strategies.
- Researchers in 2020 showed both in theory and experimentation that stay-or-leave decisions in exponentially decreasing reward environments are consistent with deep R-learning.
- People have also done experiments on whether model-free RL algorithms or the Marginal Value Theorem(MVT) Rule better describes the behaviour subjects engaged in foraging. They found MVT to be a better description.
- However, in our project we objective is to find the best possible RL agent in in this sequential decision-making environment.

https://pubmed.ncbi.nlm.nih.gov/28918312/,
https://proceedings.neurips.cc/paper/2020/file/da97f65bd113e490a5fab20c4a69f586-Paper.pdf/,

Environment Details

- We define states as the number of times harvesting has been done in a particular patch.
- There are two possible actions, leaving a patch or harvesting it.
- An action leads to a specific state.
- On leaving a patch the state resets to 0.
- Travel time is 3 or 10 seconds.
- Total time limit is 4 minutes.
- Travel time is U(0.6,1.4)



https://berry-game-pst.s3.ap-south-1.amazonaws.com/bananafarm2.html,

MDP Equations - Normal Environment

Transition

- $p_{ss'}^{Harvest} = P[S_{t+1} = n + 1 | S_t = n, A_t = Harvest] = 1$
- $p_{ss'}^{Leave} = P[S_{t+1} = 0 | S_t = n, A_t = Leave] = 1$

Reward

- $R_{ss'}^{Harvest} \sim (7 - 0.5n + N(0, 0.025))$, where n is current state
- $R_{ss'}^{Leave} = 0$

MDP Equations - Rich patch Poor Patch

Transition

- $p_{ss'}^{Harvest}$ =P [S_{t+1} = n + 1|S_t = n, A_t = Harvest] = 1
- $p_{ss'}^{Leave}$ = P[S_{t+1} = 0|S_t = n, A_t = Leave] = 1

Reward

- $R_{ss'}^{Harvest}$ ~ (Q−0.5n +N(0, 0.025)), where n is current state and Q is a value random chosen between two values A and B. A is chosen randomly from integers 2 to 7 and B is chosen randomly from 10 to 15.
- $R_{ss'}^{Leave}$ = 0

R-learning Algorithm

- Average reward per time step and action-value function,

$$p^{\pi} = \lim_{n \to \infty} \frac{1}{n} \sum_{t=0}^{n} E_{\pi}\{r_t\} \qquad Q^{\pi}(s, a) = E_{\pi}\{\sum_{t=0}^{\infty} r_{t+1} - p^{\pi} | s_0 = s, a_0 = a\}$$

- TD learning algorithm which is the average reward equivalent to Q-learning algorithm:

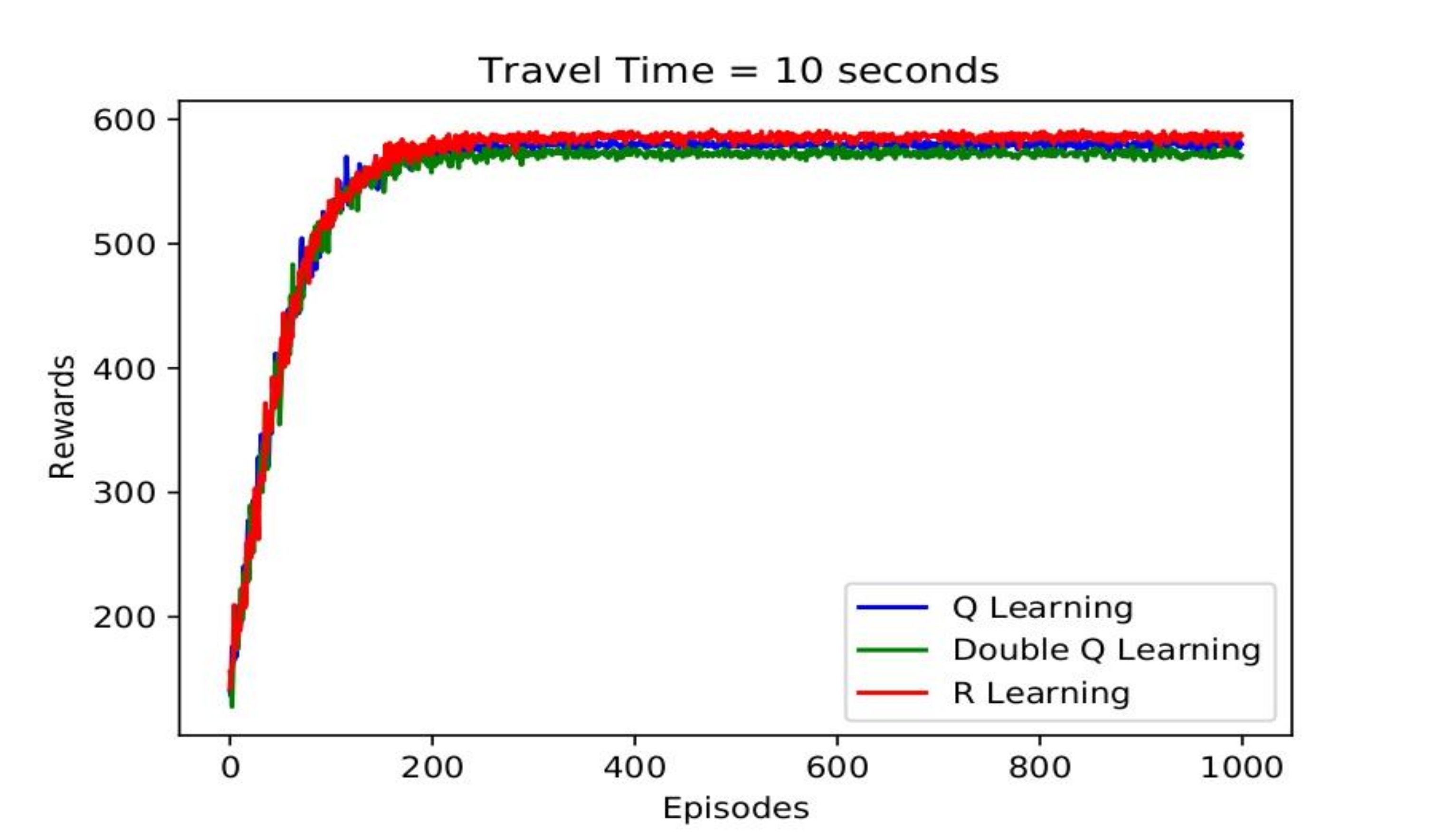
$$Q(s, a) \leftarrow Q(s, a) + \alpha(r - p + \max_{a'}(Q(s', a')) - Q(s, a))$$
$$p \leftarrow p + \beta(r - p)$$

https://pubmed.ncbi.nlm.nih.gov/28918312/,

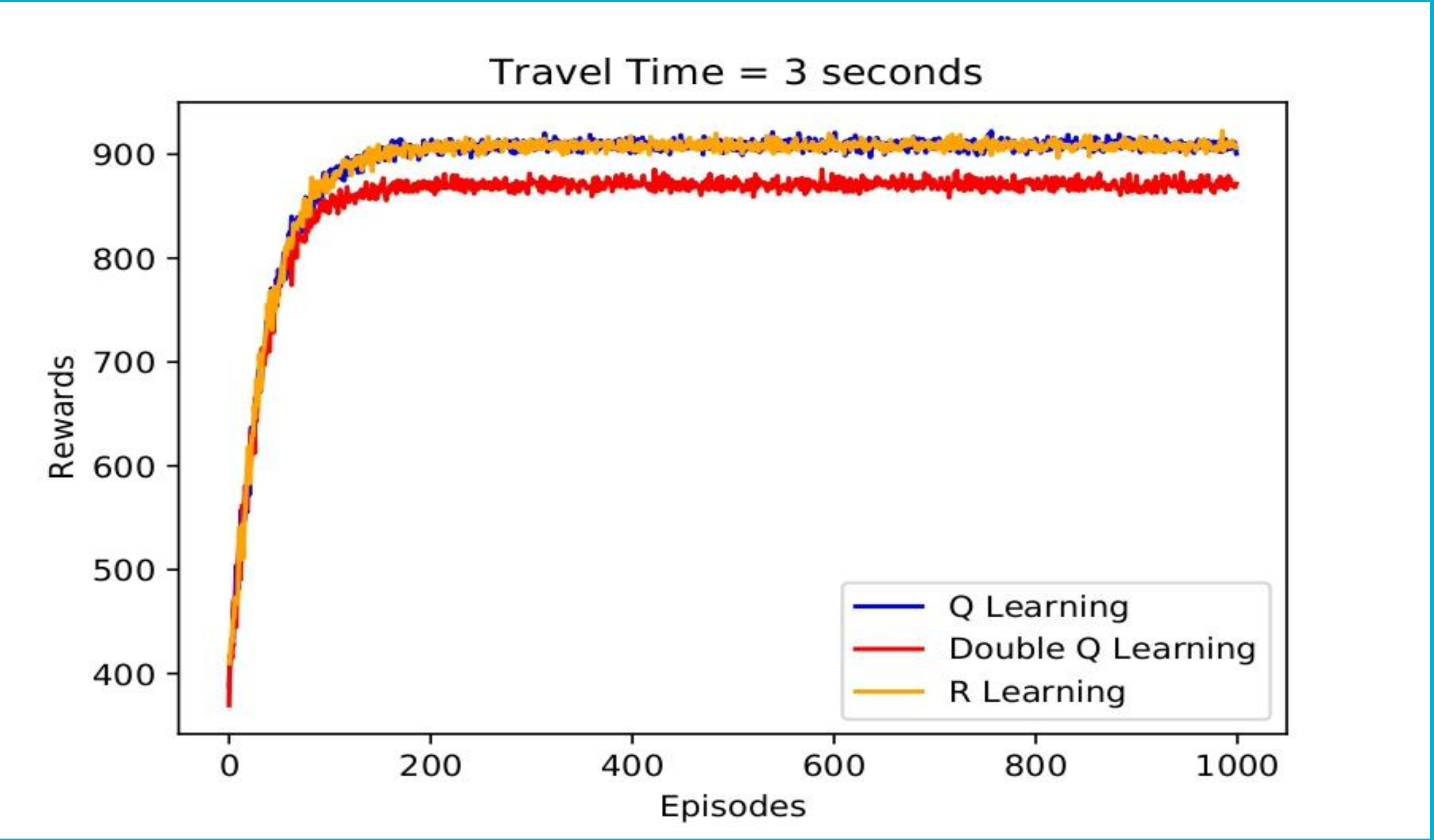
Conventional RL Algorithms-Design Choices

- Number of Environments: 10
- Number of episodes: 1000
- Alpha: Exponential Decay from 1 to 0.01 in 1000 steps divided by 1000
- Epsilon: Exponential decay from 1 to 0.0001 in 400 steps and then constant for 600 steps
- Gamma: 1
- Seeds: 50-59
- Lambda: 0.5
- noPlanning=2
- maxTrajectory(Trajectory Sampling) =2
- maxTrajectory(Modified Trajectory Sampling) =3

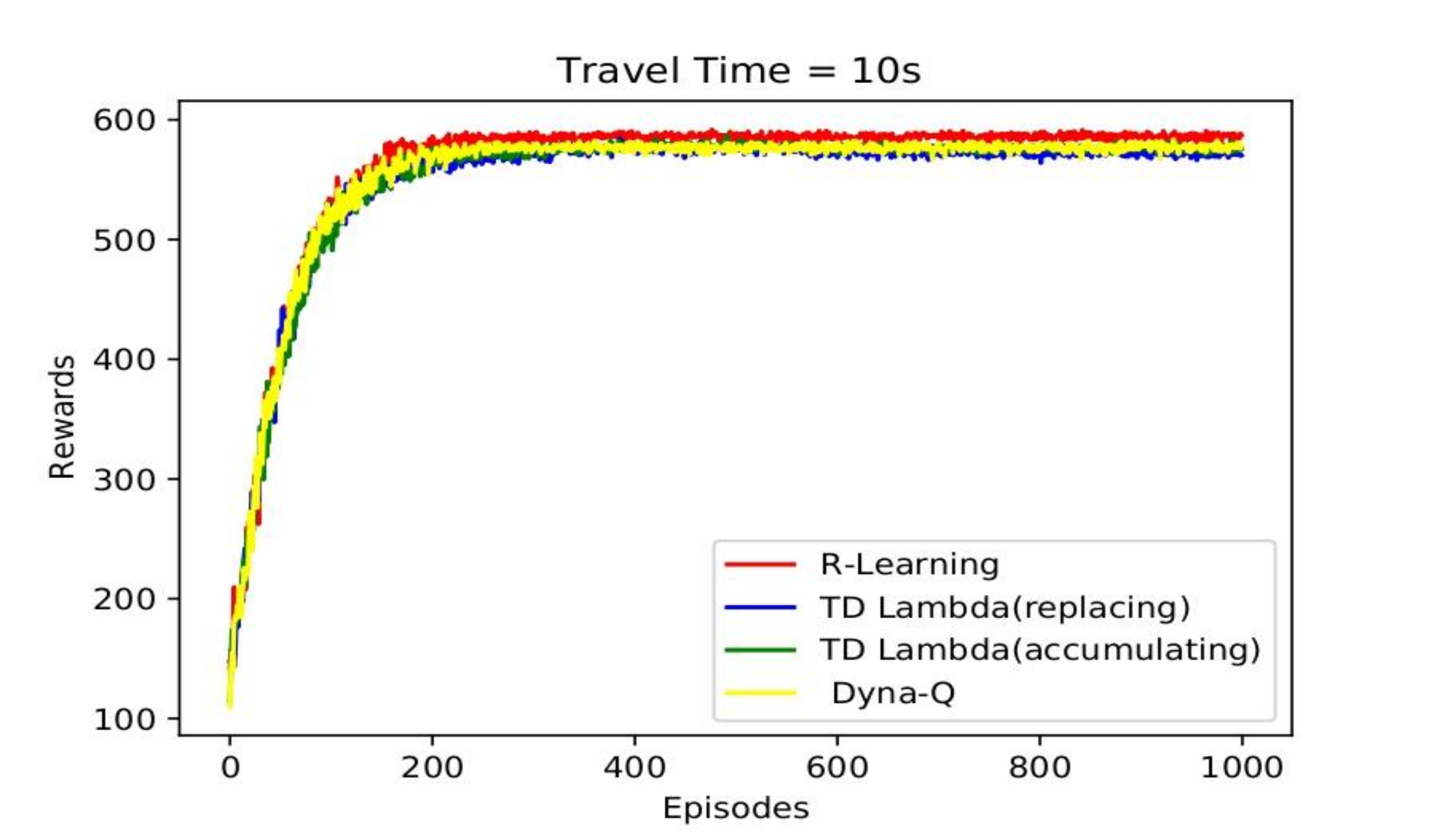
Comparing Algorithms



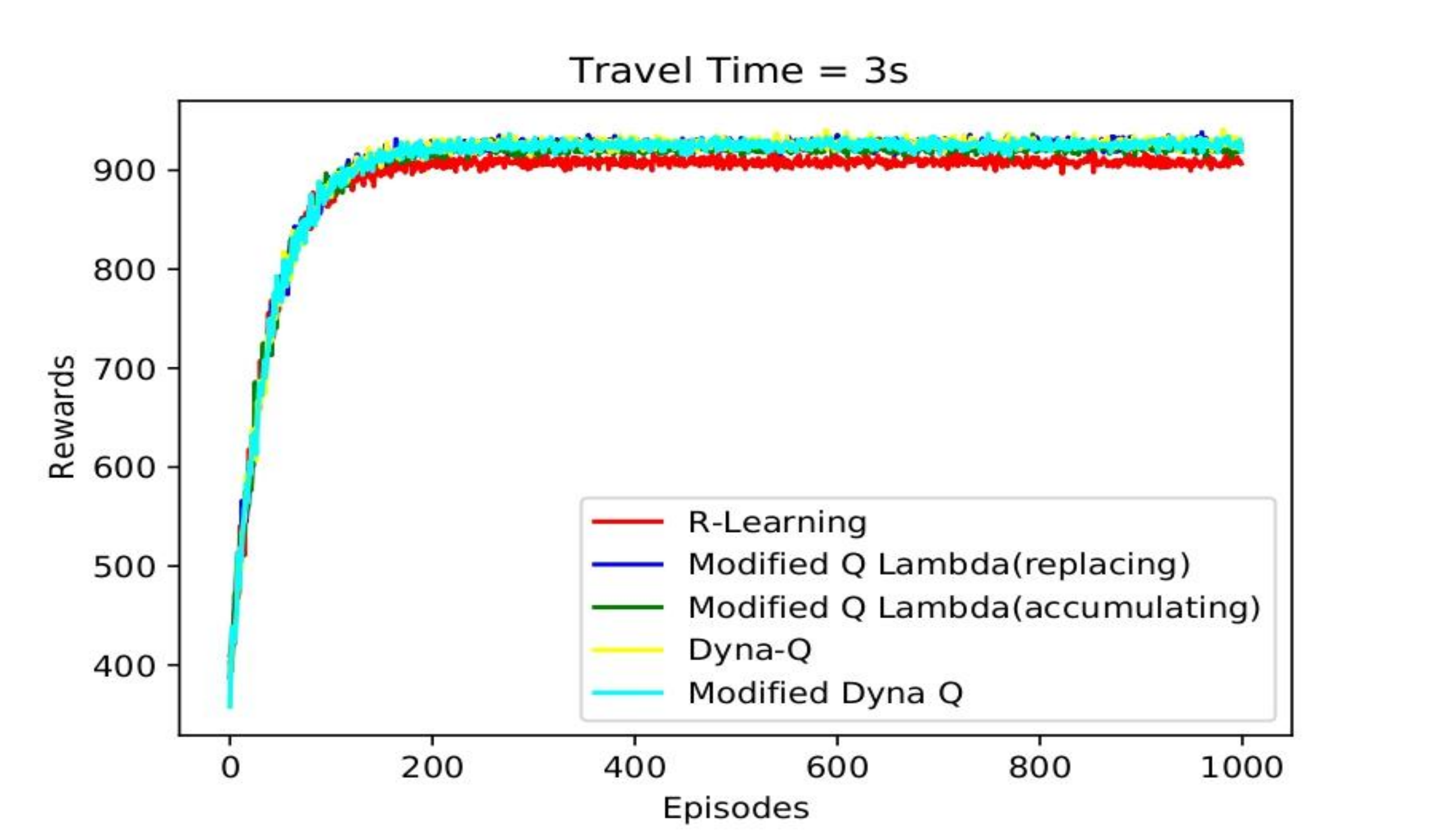
Comparing Algorithms



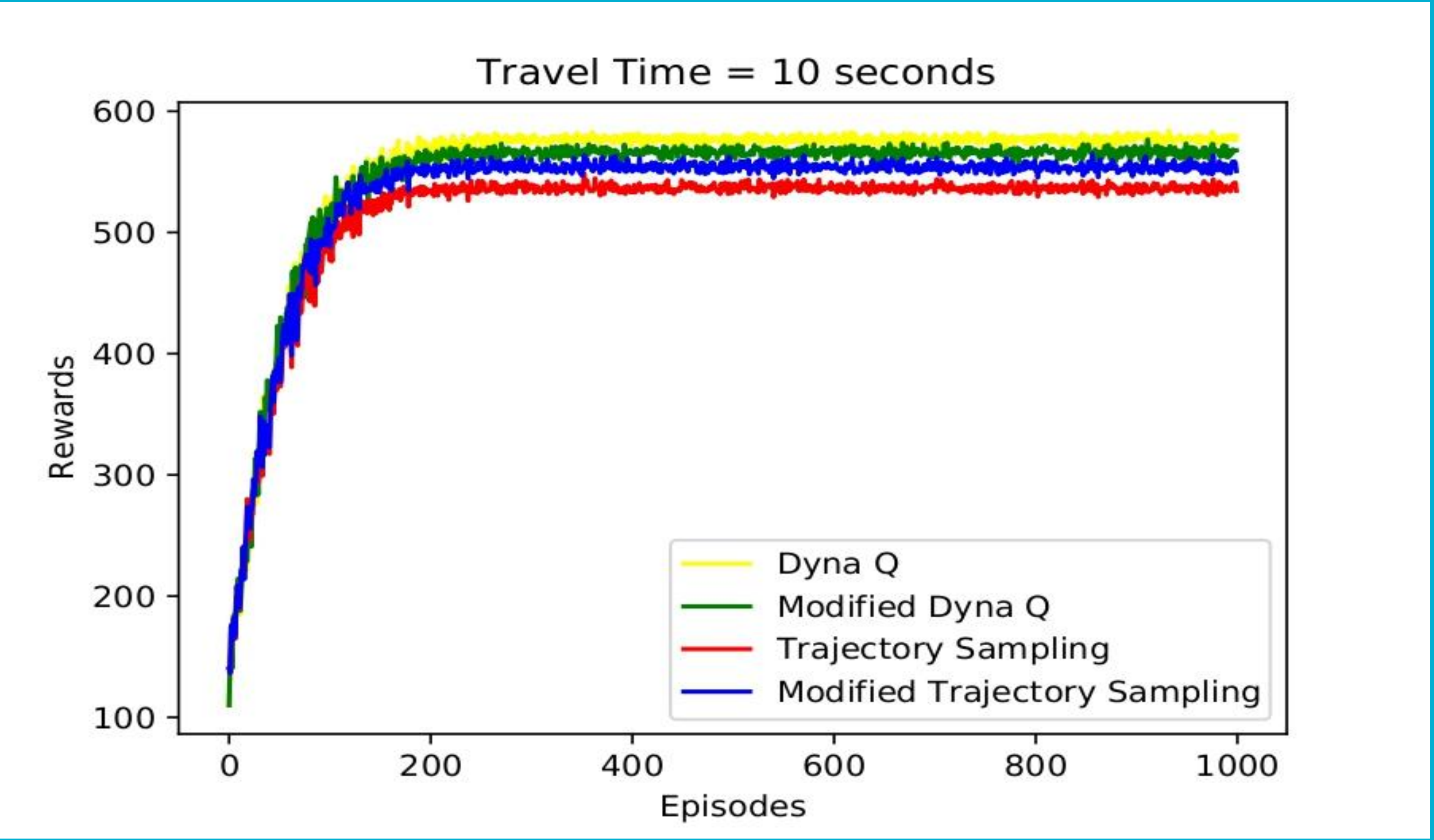
Comparing Algorithms



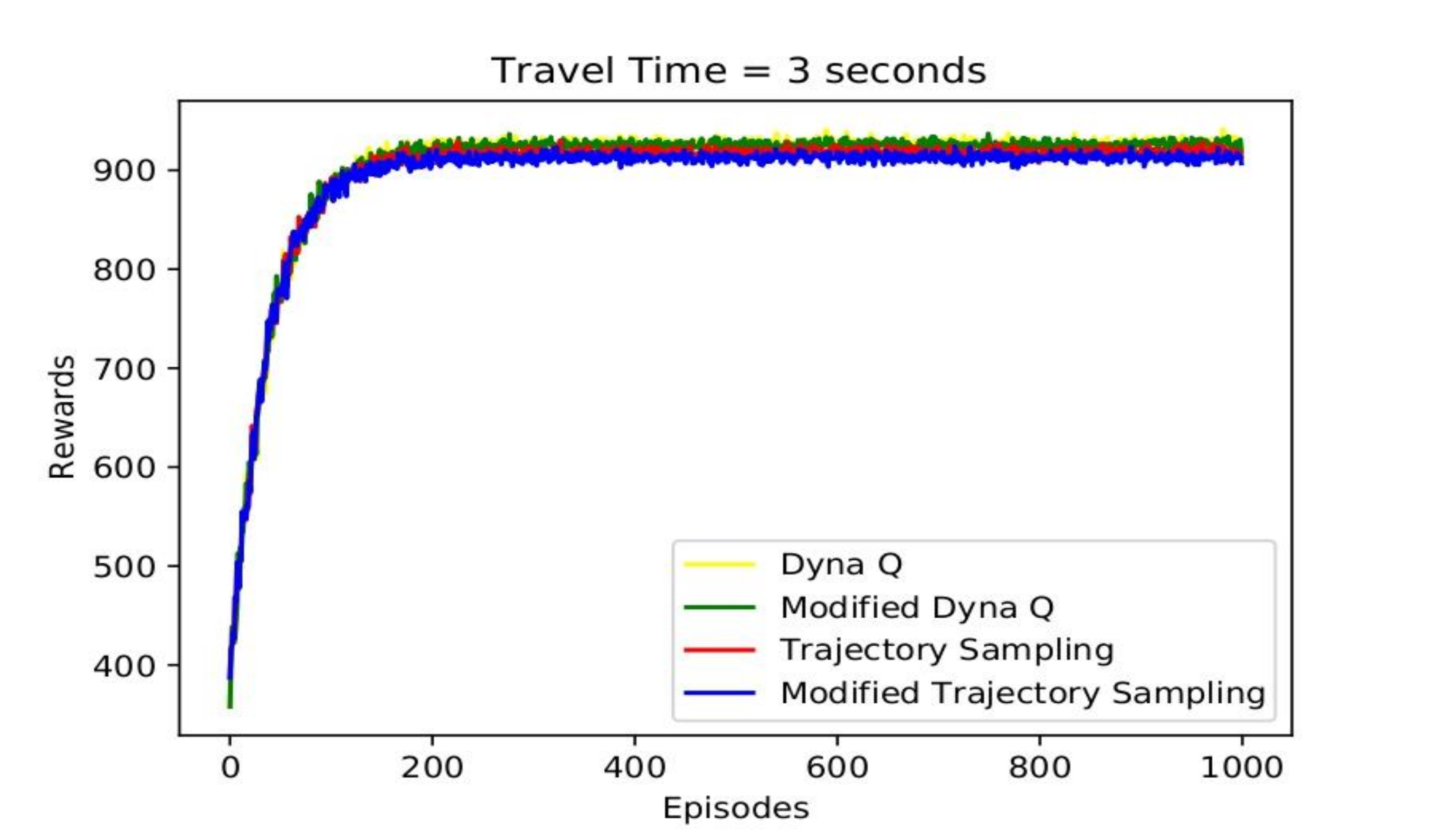
Comparing Algorithms



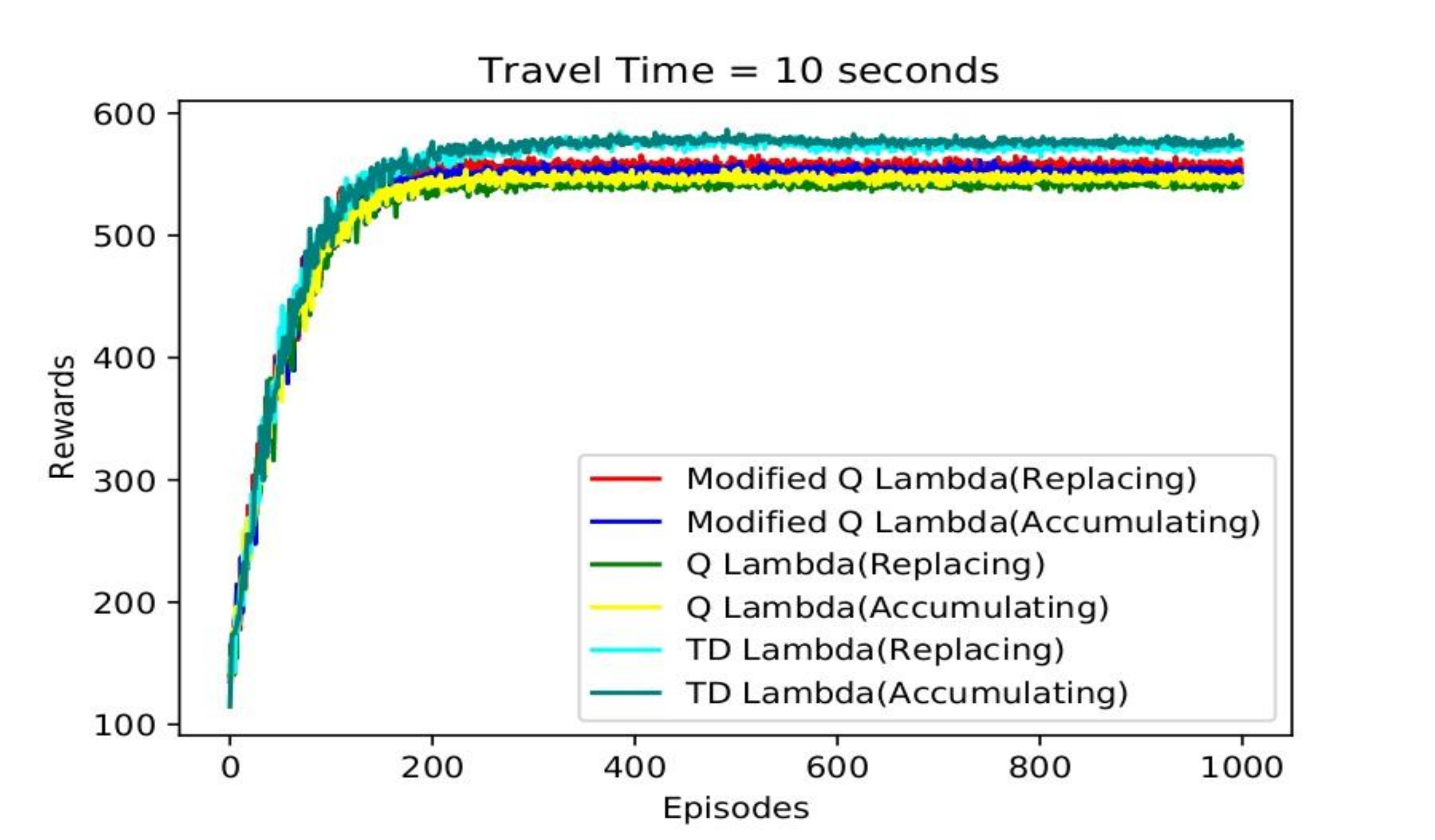
Comparing Algorithms



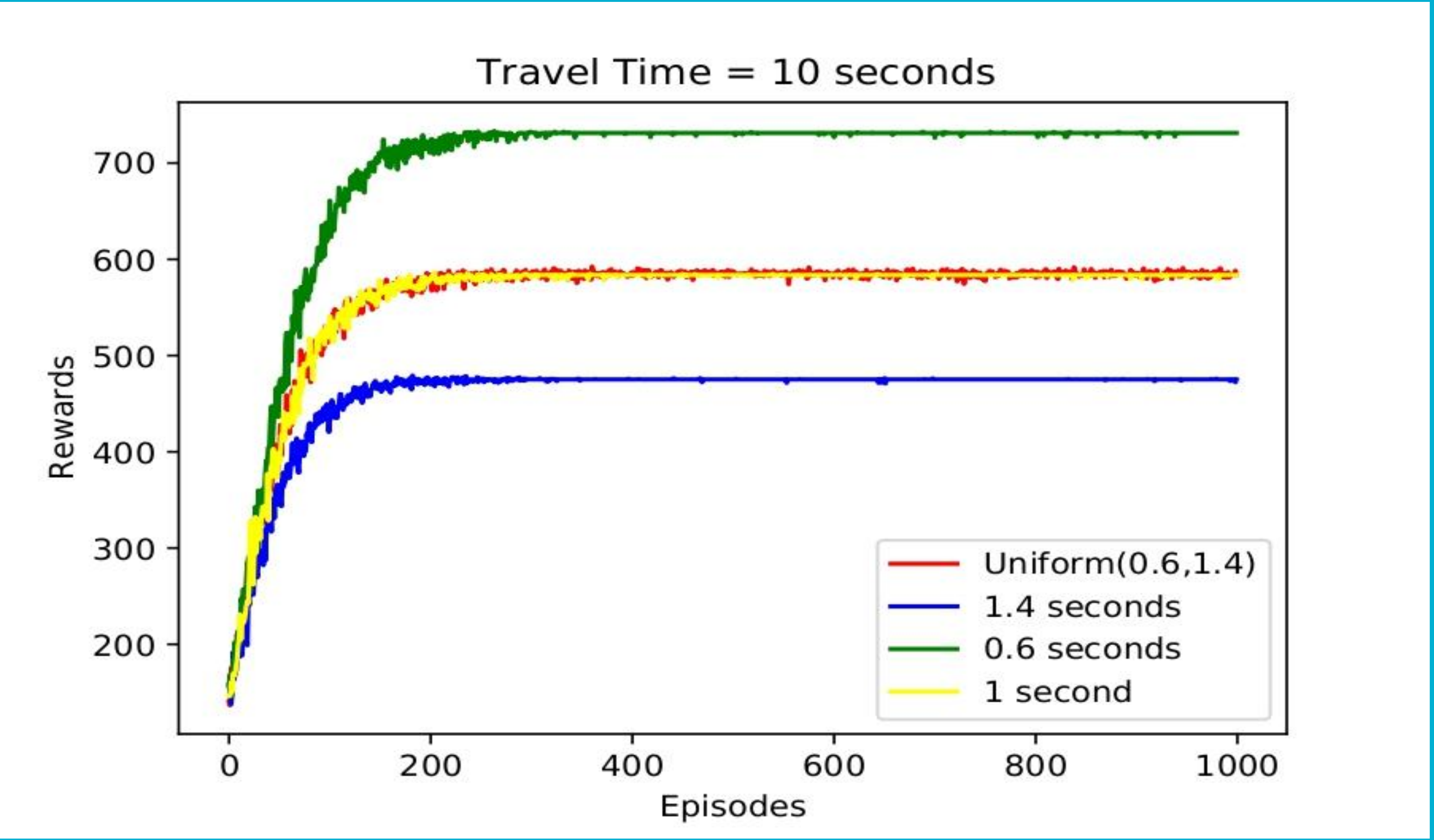
Comparing Algorithms



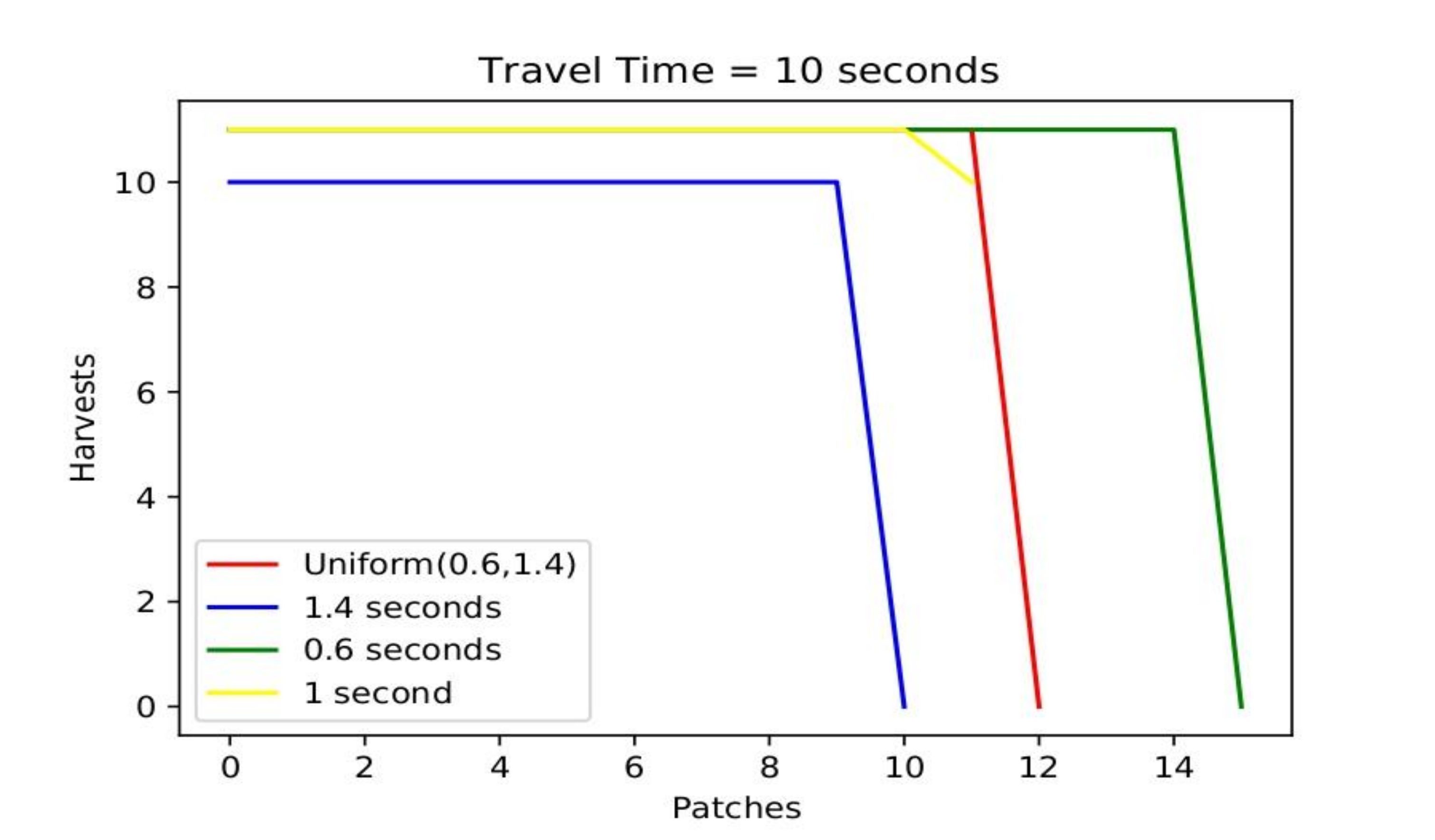
Comparing Algorithms



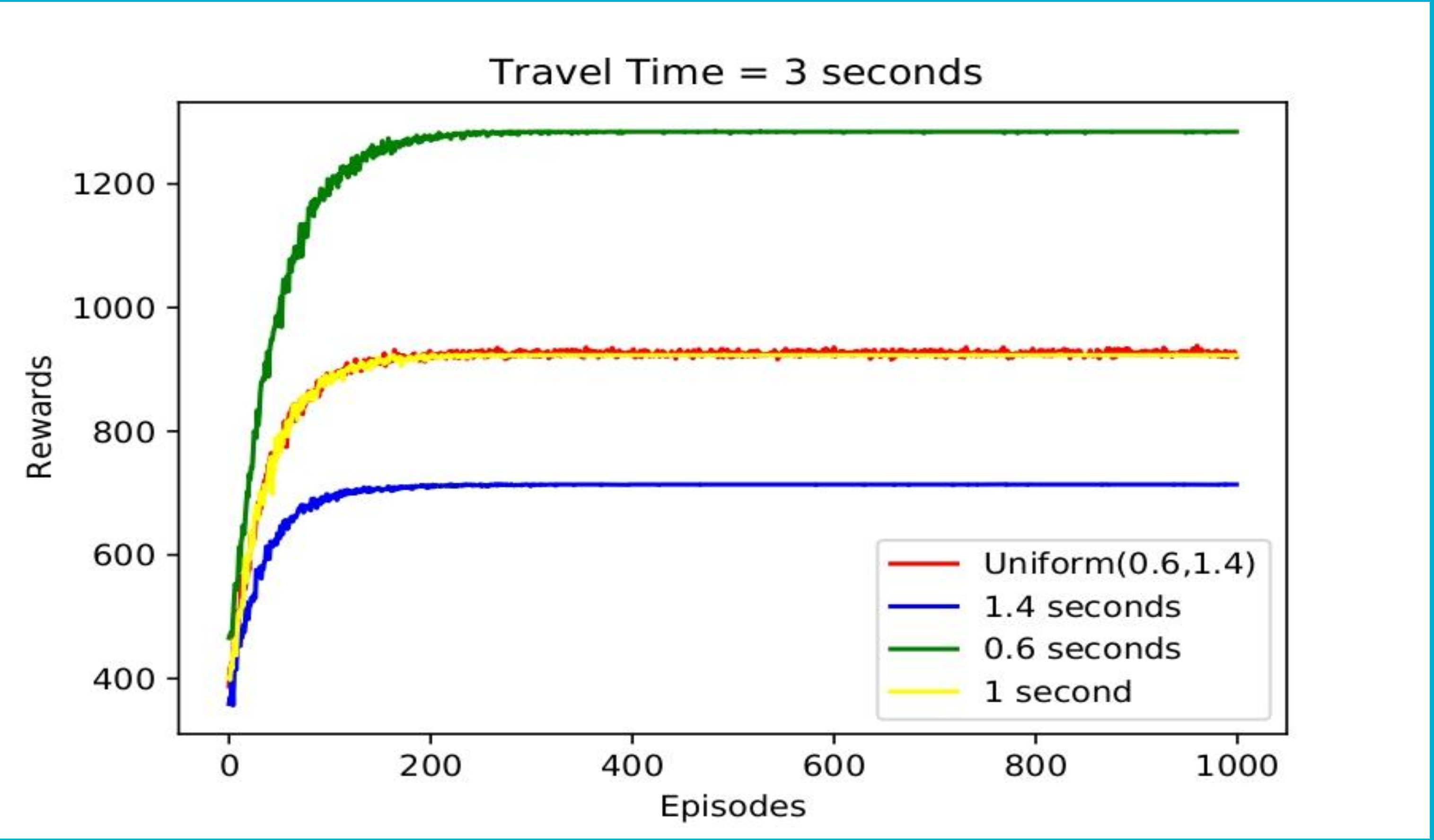
R-Learning


Travel Time = 10 seconds
Rewards
Episodes
700
600
500
400
300
200
0
200
400
600
800
1000
Uniform(0.6,1.4)
1.4 seconds
0.6 seconds
1 second

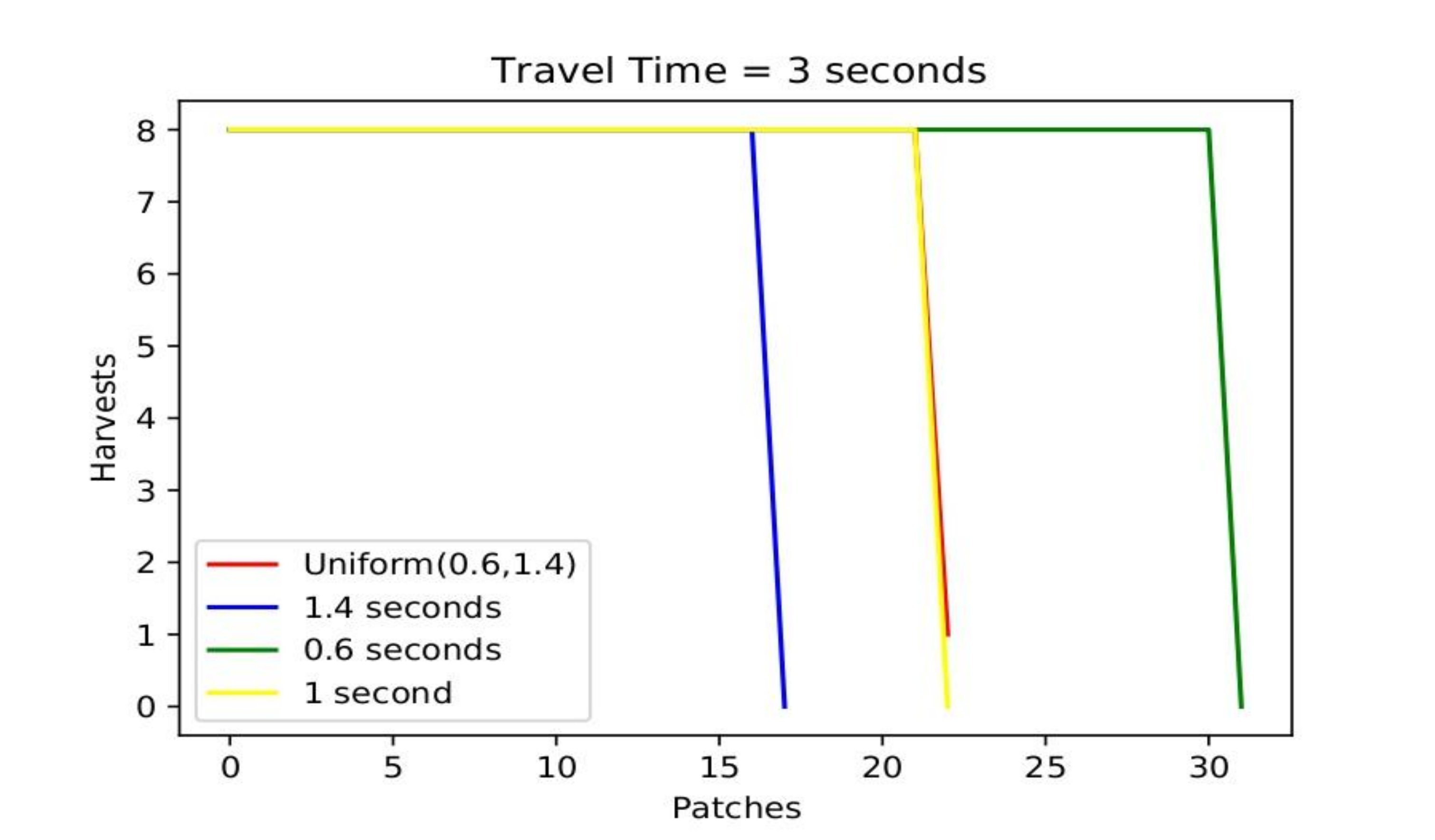
Patch harvests: R-Learning



Modified Q (lambda)



Patch harvests: Modified Q(lambda)

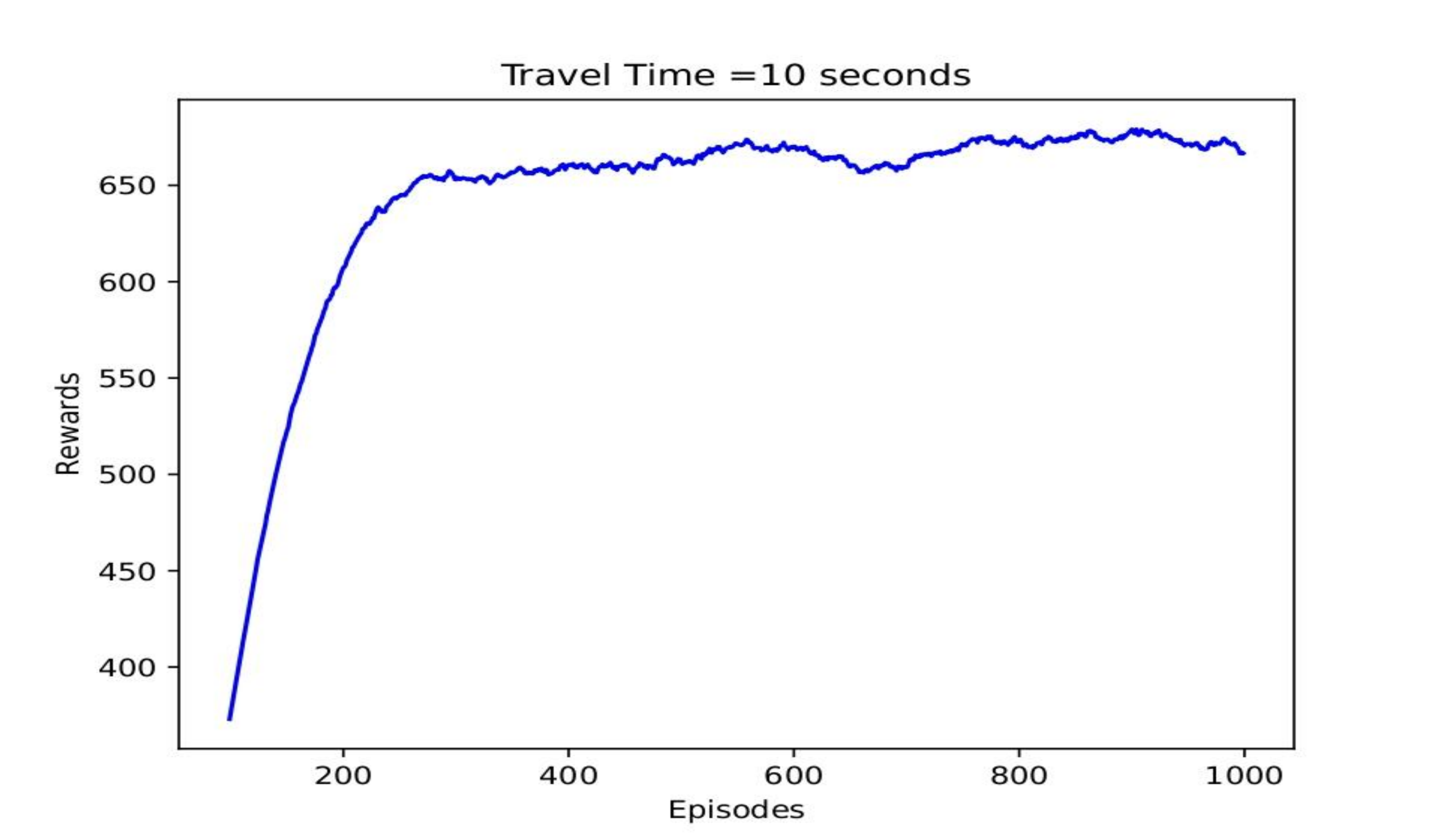


Conventional RL Algorithms for Rich and Poor Patch Environment

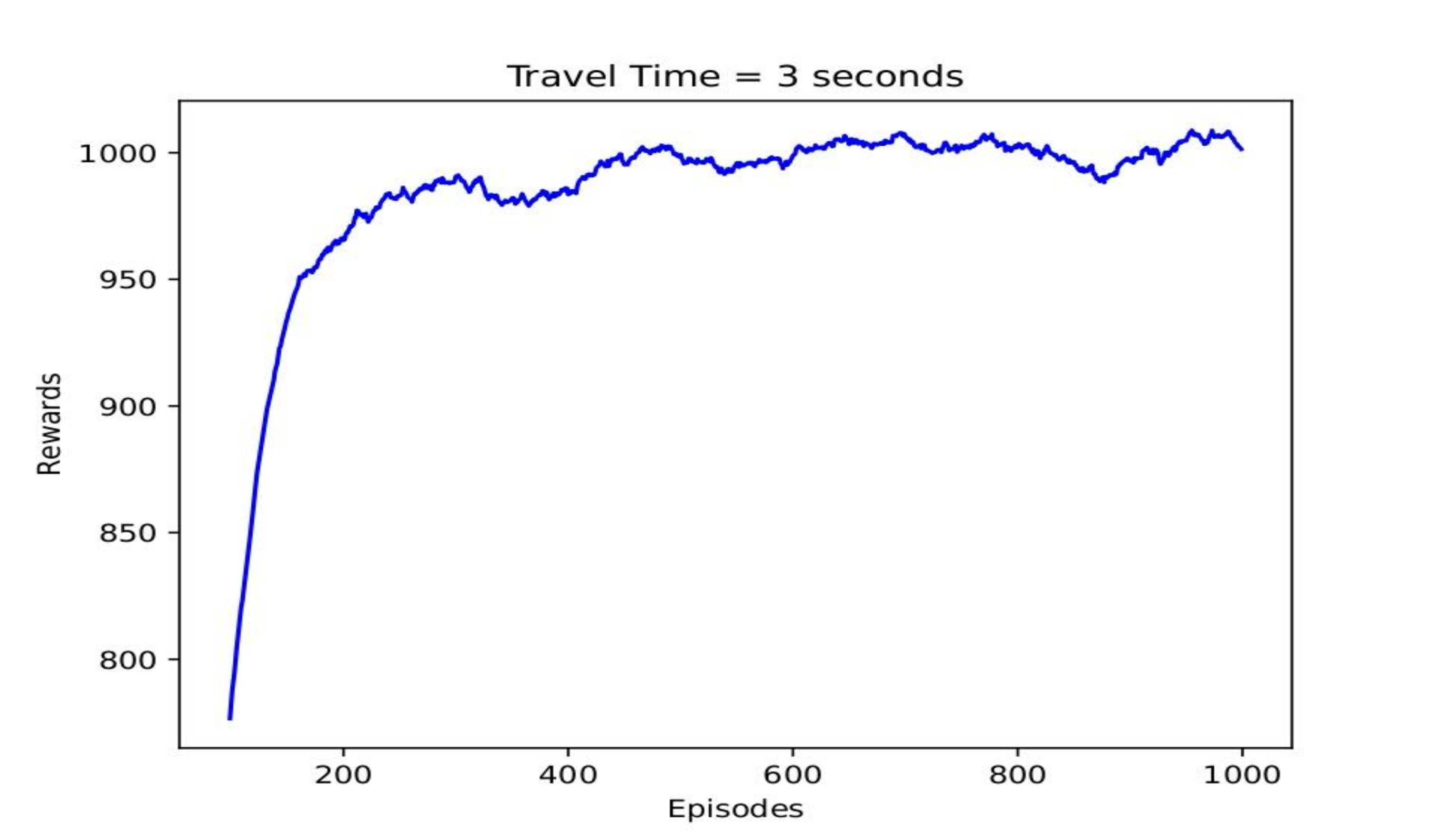
For the conventional algorithms, we used a 2-D state space [patches]x[harvests].

This gives the agent an extra dimension to work with, (keeping patches and harvests independent) ensuring better learning .

Moving Average Reward (R-learning)



Moving Average Rewards(R-Learning)



State Space Representation for Deep RL Approach

- For implementing the deep RL agents for this problem, we consider a state space given by a vector of the 5 latest rewards and the last action.
- Providing agents with information about the past rewards gives them the scope of learning various behaviours following the same immediate action
- The information about the latest action allows the models to distinguish the expected rewards from violations of the reward schedule.
- For all the agents, the results were averaged for six instances of the environments.

https://proceedings.neurips.cc/paper/2020/file/da97f65bd113e490a5fab20c4a69f586-Paper.pdf/,

DQN Design Choice

- Value Network (6, 512, 128, 2)
- Exponentially Decaying Epsilon Greedy Strategy: 1.0 to 0.3 in 20000 steps
- Replay BUffer Size: 50000
- Batch Size: 64
- Update Frequency: 10
- Greedy Evaluation Strategy
- Training Episodes: 1000, Evaluation Episodes: 1
- RMSProp Optimizer(Learning Rate: 0.0007)
- Mean Square Error Loss Function
- Gamma: 0.99

DDQN Design Choice

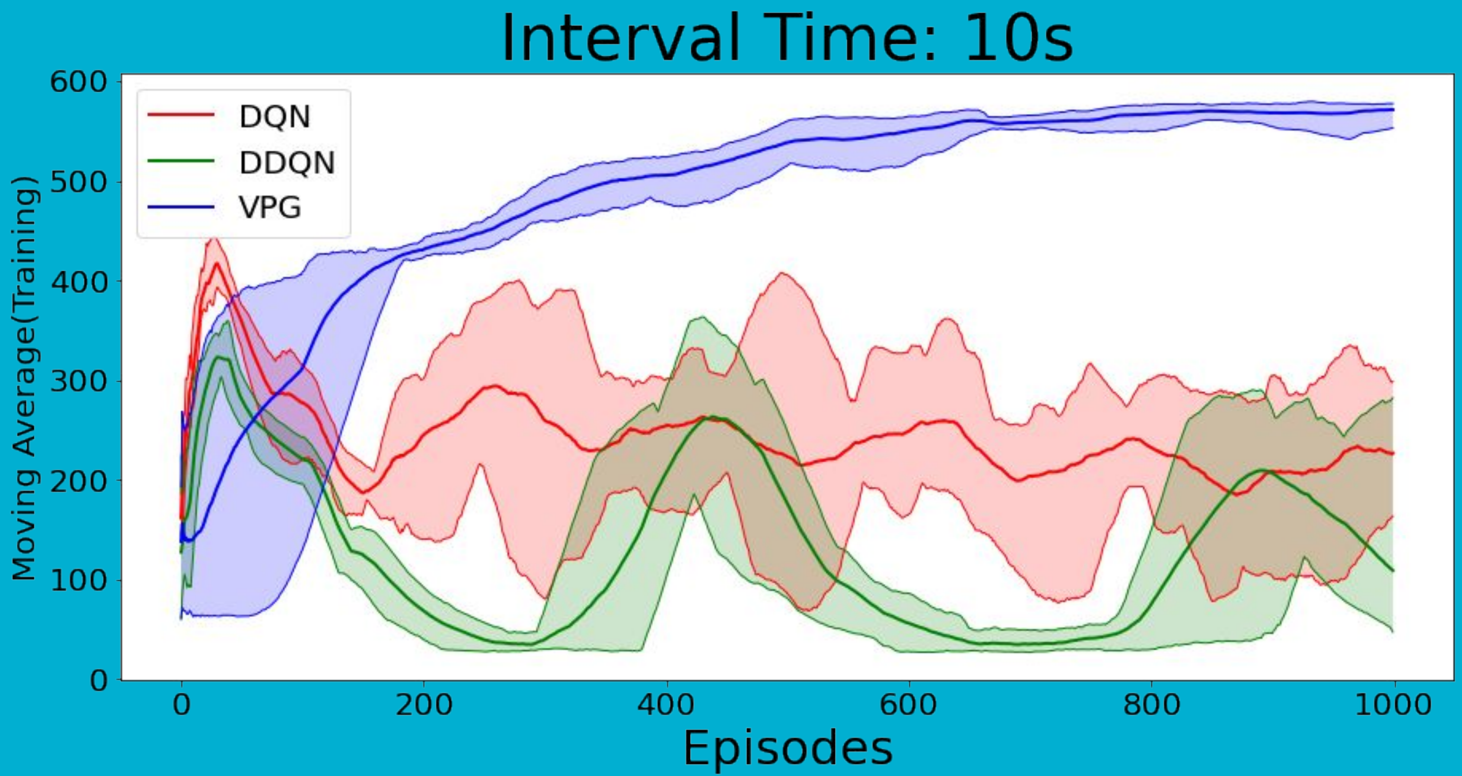
- Target And Online Value Network (6, 512, 128, 2)
- Exponentially Decaying Epsilon Greedy Strategy: 1.0 to 0.3 in 20000 steps
- Replay BUffer Size: 50000
- Batch Size: 64
- Update Frequency: 15
- Greedy Evaluation Strategy
- Training Episodes: 1000, Evaluation Episodes: 1
- RMSProp Optimizer(Learning Rate: 0.0007)
- Huber Loss Function(Gradient Clipping to 1e7)
- Gamma: 0.99

VPG Design Choice

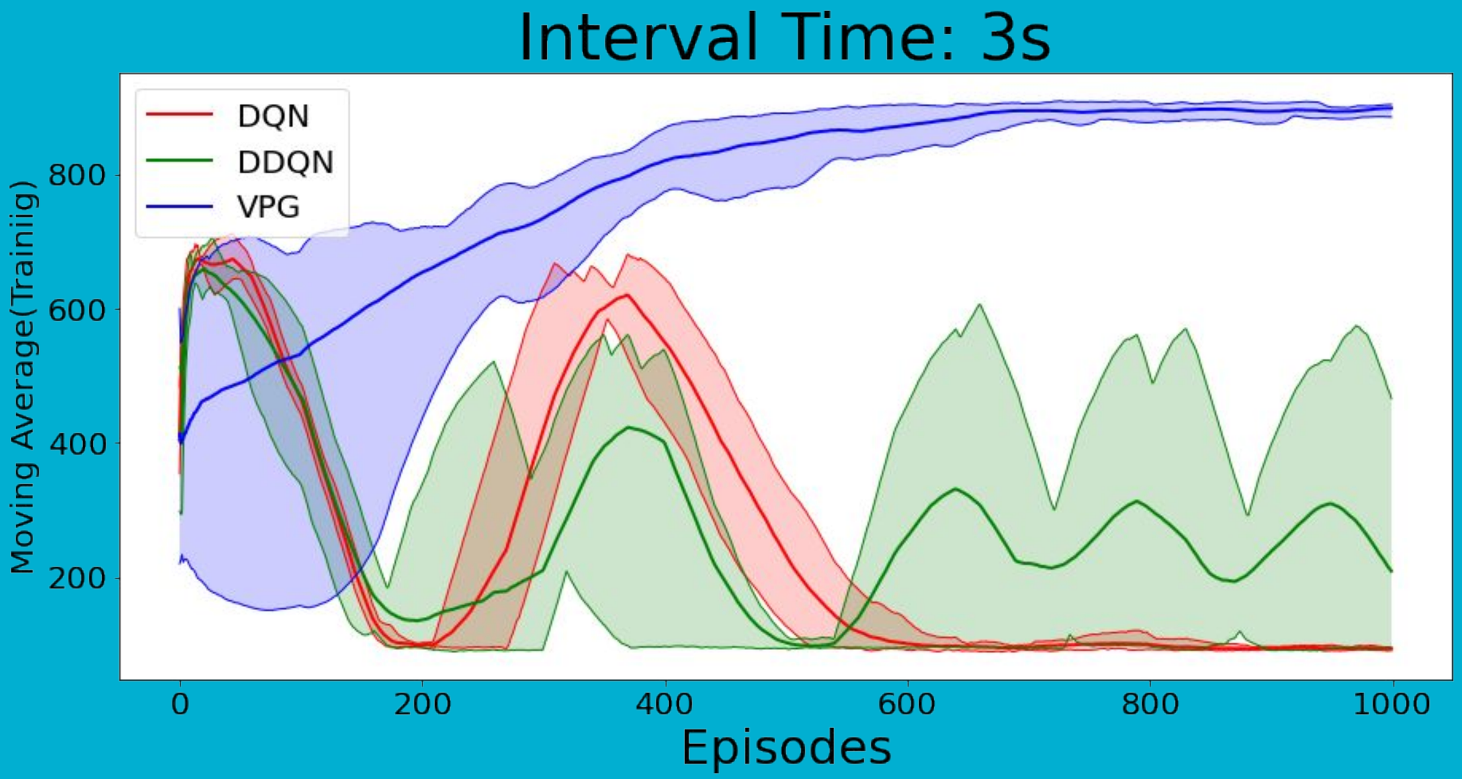
- Value Network (6, 512, 128, 2)
- Exponentially Decaying Epsilon Greedy Strategy: 1.0 to 0.3 in 20000 steps
- Replay BUffer Size: 50000
- Batch Size: 64
- Update Frequency: 10
- Greedy Evaluation Strategy
- Training Episodes: 1000, Evaluation Episodes: 1
- RMSProp Optimizer(Learning Rate: 0.0007)
- Mean Square Error Loss Function
- Gamma: 0.99

Simple Patch Environment

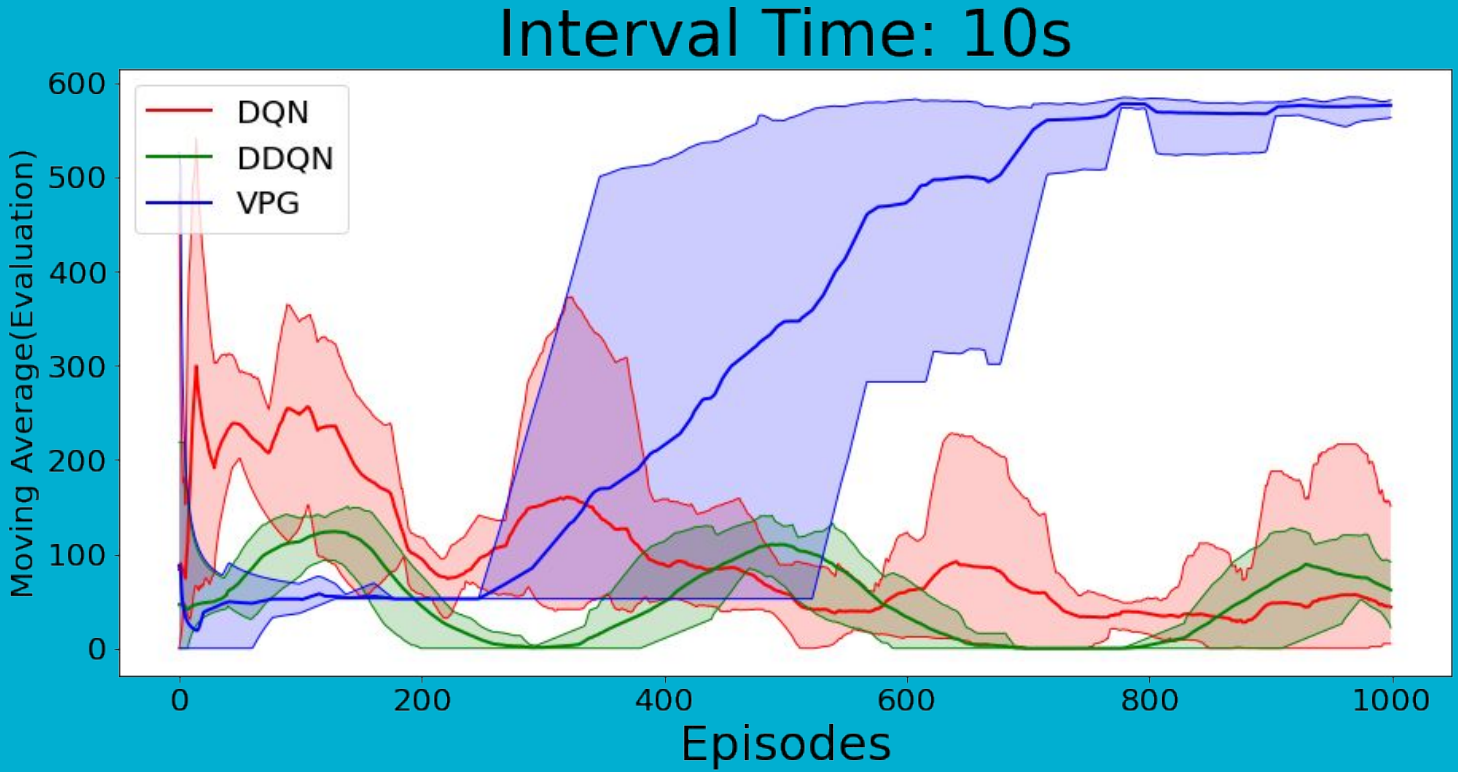
Moving Average Reward during Training



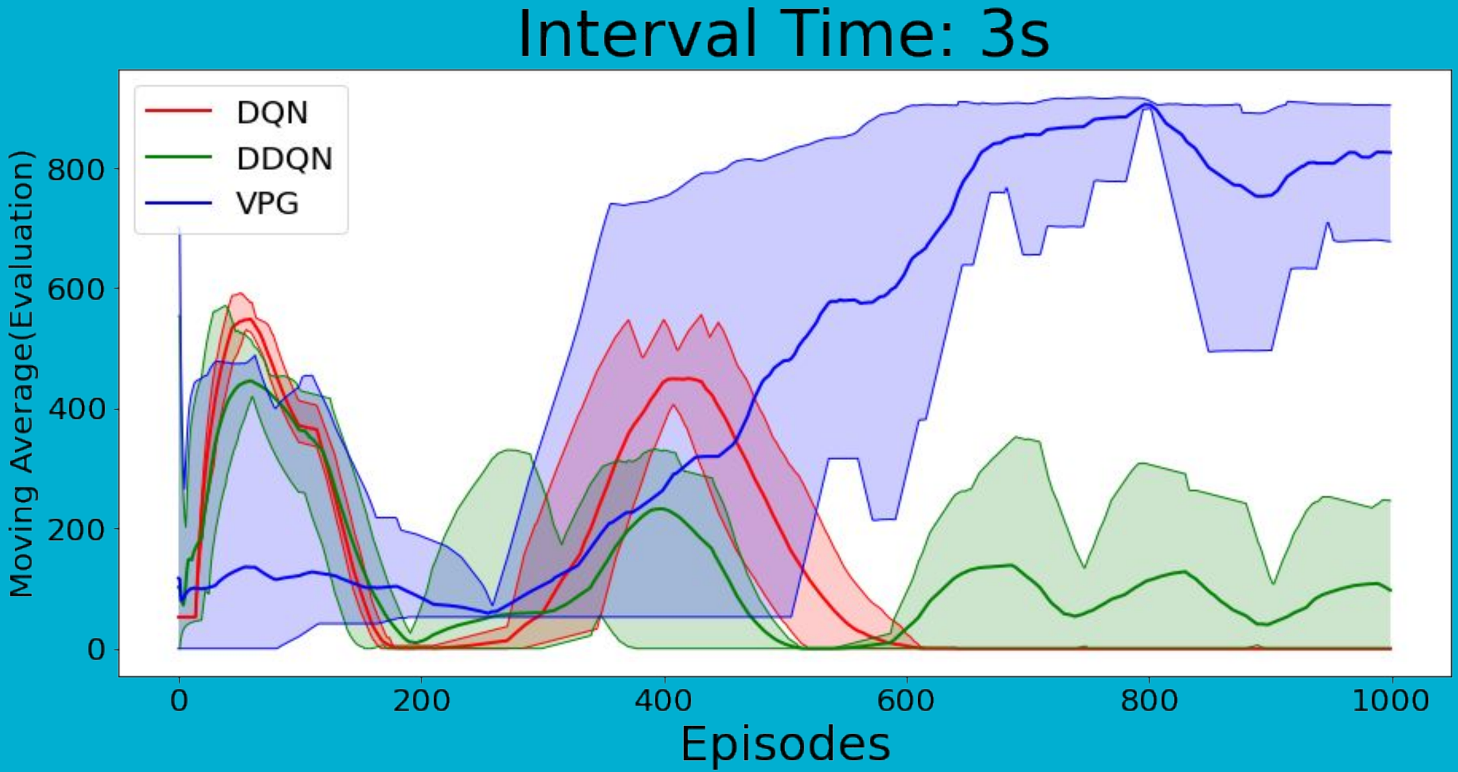
Moving Average Reward during Training



Moving Average Reward during Evaluation

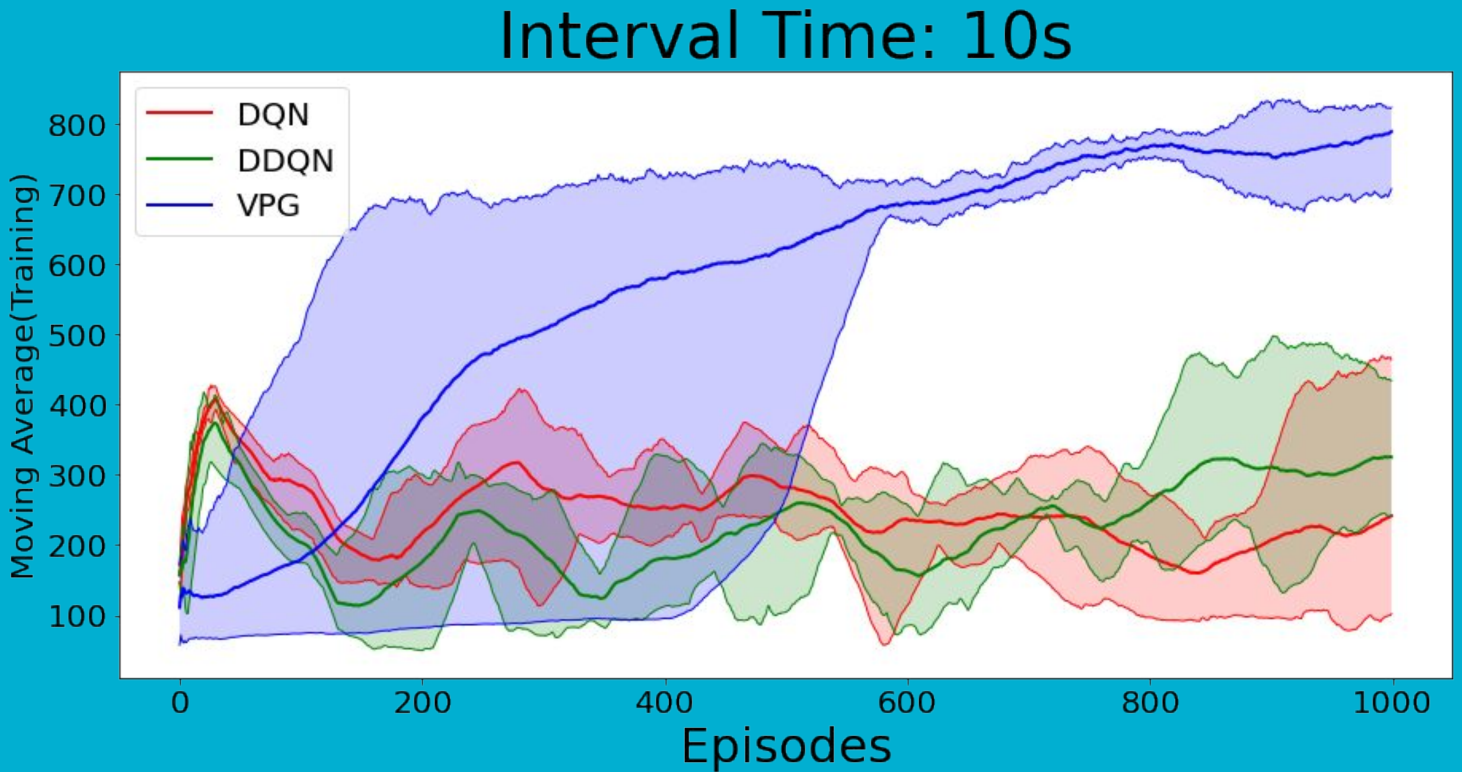


Moving Average Reward during Evaluation

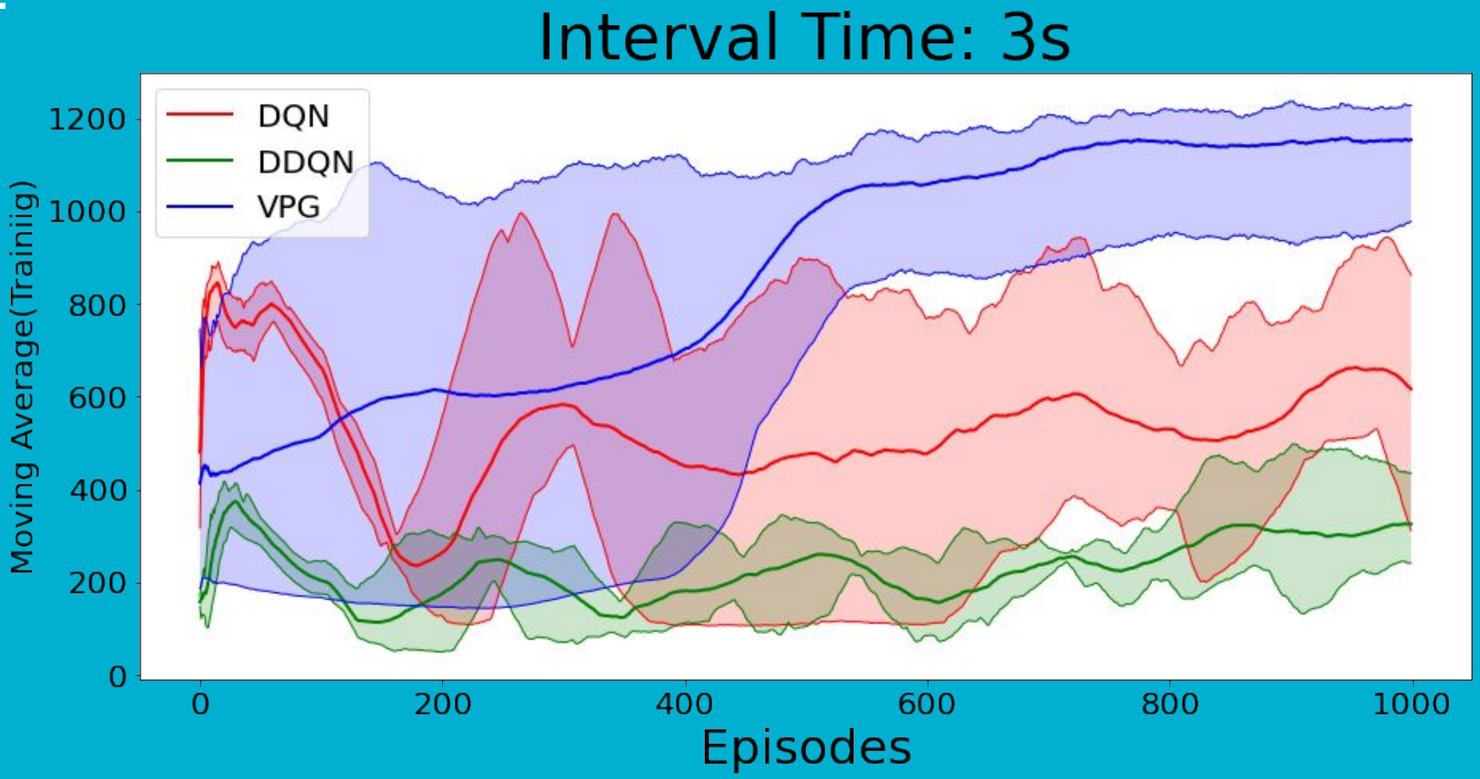


Rich And Poor Patch Patch Environment

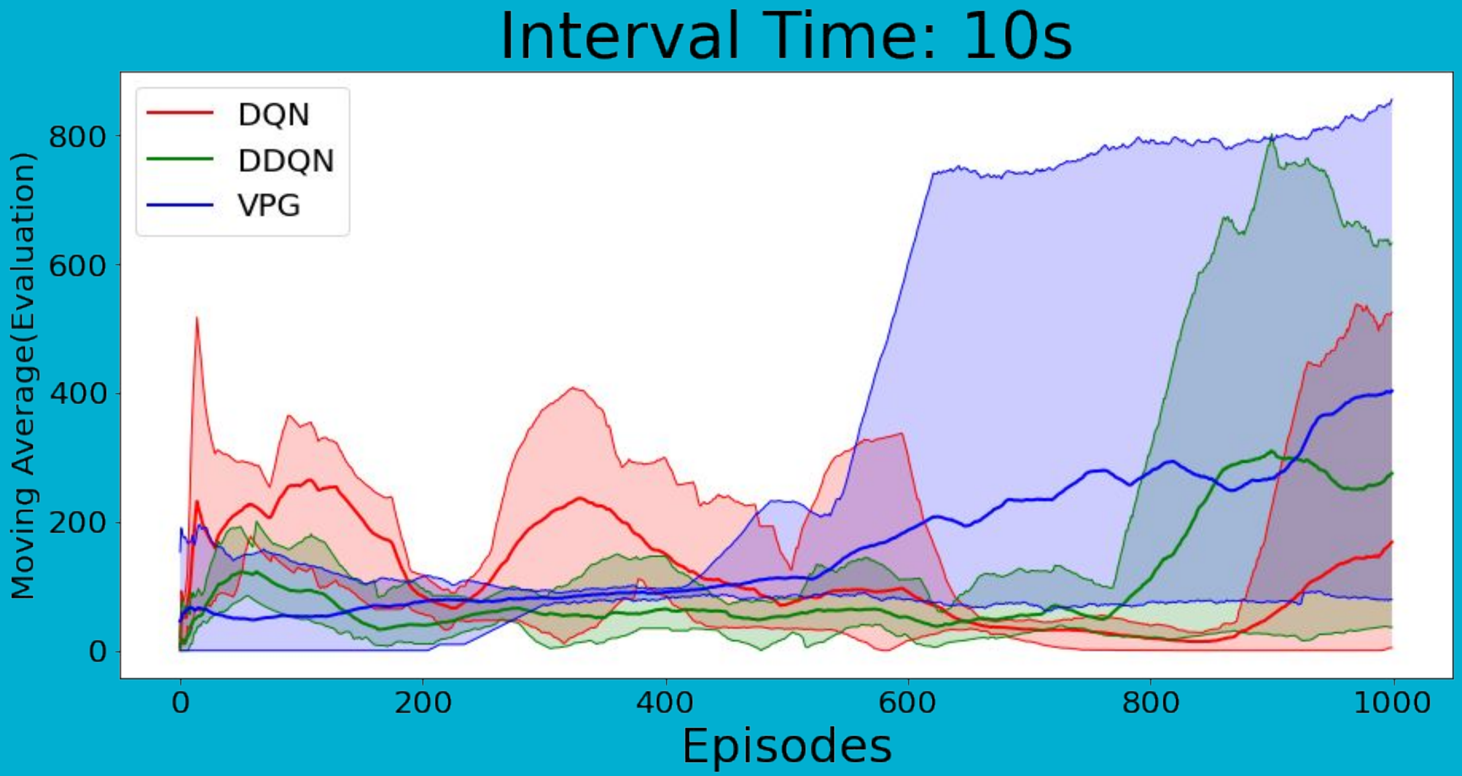
Moving Average Reward during Training



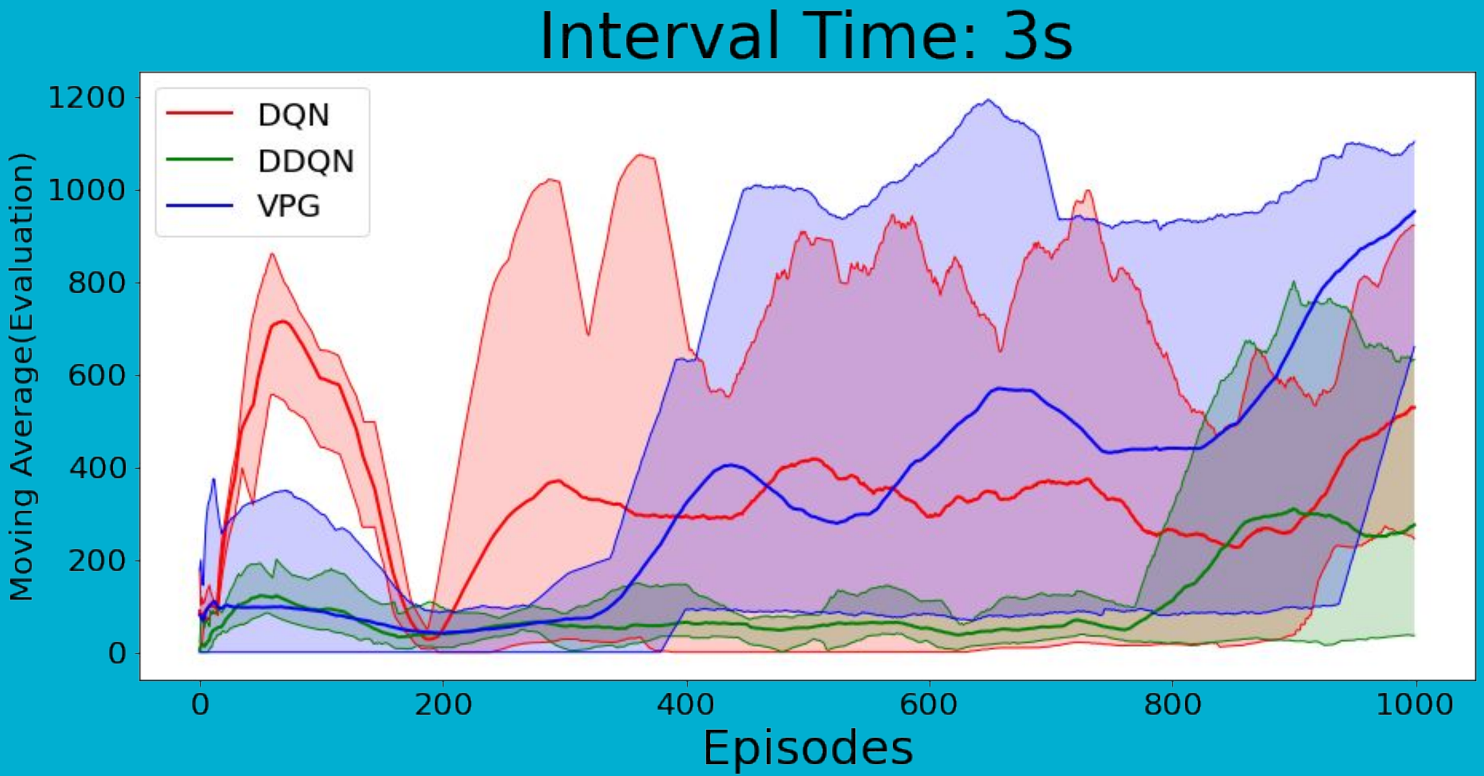
Moving Average Reward during Training



Moving Average Reward during Evaluation



Moving Average Reward during Evaluation



Results

Conventional RL Agents

- For a travel time of 10 seconds in simple environment, R-Learning outperforms the rest, while the others still give competitive rewards. Total reward is ~580.
- Similarly for a travel time of 3 seconds in simple environment, Modified Q-Learning outperforms the rest, total reward is ~930.
- For the complex environment, the reward for travel time of 10 seconds, using R-Learning algorithm for 2-D state space is ~685.
- Similarly, the reward for interval time of 3 seconds is ~1000.

Results

Deep RL Agents:

- The DQN and DDQN agents took around 90 minutes to run each and VPG took just less than 30 minutes for six instances of the environment.
- We see that the VPG agents consistently outperforms the DQN and DDQN agents, but is not better than the R-learning agent in the simple environment.
- However in the rich and poor patch environment, VPG is able to beat the R-learning agent giving rewards touching 800 and 1200 for interval time of 10 seconds and 3 seconds respectively.

Future Directions

- Keeping in mind our findings of better performance of the Vanilla Policy Gradient Algorithm, we can move to more general policy-based methods like gradient-free or black box methods.
- Would be worthwhile to consider model-based deep RL methods like Monte Carlo Tree Search(MCTS) in these types of environments.
- Could use a combination of model-free and model-based RL, with model-free method being used to avoid planning costs due to deep decision tree search and at the same time getting a proper model of the environment to obtain high rewards due to model-based RL.

https://pubmed.ncbi.nlm.nih.gov/28918312/,

Contributions

Nibir: Implementation of deep agents for both environments - DQN, DDQN and Vanilla Policy Gradient(VPG), Literature Research concerning solving the problem using deep learning approaches, Directions for future work concerning model-based deep RL approaches

Romit: Implementation of Q learning, double-Q learning, R-learning, Q(lambda), TD(lambda) for normal environment and R-learning for rich patch poor patch environment, Literature Research, Directions for future Work concerning control in highly stochastic environments (like stock markets), Came up with the idea of 2-D state space for rich-poor patches.

Neelabh: Implementation of modified Q(lambda), Dyna Q, modified Dyna Q, trajectory sampling and modified trajectory sampling for normal environment and R learning for rich patch poor patch environment. Literature Research involving implementation of r-learning, Future Work concerned with varying time perception and decrease in decision time.

Thank You