CS698R: Deep Reinforcement Learning Project

Final Project Report

Team Name: Deep Treaders
Project Title: Patch Leaving Decisions
Project #: 1
Team Member Names: Roll No
Nibir Baruah: 190545
Neelabh Singhania: 190538
Romit Mohanty: 190720

1 Introduction

Our problem focuses on solitary foraging. We are trying to answer how an animal feeding on a particular food patch, and hence depleting it, decides when it is optimal to leave the patch in search of a richer one, provided that it has only a finite amount of time to forage in the environment. This problem is quite relevant in neuroscience because the ecological importance of foraging is thought to have profoundly shaped neural decision-making systems. Also, the behaviour of real animals often approach optimal foraging behaviour, but little is understood about the learning and decision mechanisms that lead them to show such behaviour. Substantial evidence suggests that the phasic activities of dopaminergic neurons in the primate midbrain represent a temporal difference (TD) error in predictions of future reward, which increases above and decreases below baseline consequent on positive and negative prediction errors, respectively (Niv [2005]). So, updates similar to TD can be used to study the learning procedure in any animal brain.

We made an effort to solve this classical problem using the average reward RL approach and did some experiments with value-based and policy-gradient deep RL methods. The GitHub link to the project can be found here.

2 Related Work

Shuvaev et al. [2020], showed both in theory and experimentation that stay-or-leave decisions in exponetially decreasing reward environments are consistent with deep R-learning both behaviorally and neuronally. The results they obtained suggest that real-world agents leave depleting resources when their reward rate falls below its exponential average. Based on this observation they proposed a biologically relevant decision rule, called the Leaky-MVT, and the corresponding learning mechanism accounting for individual decisions of real-world agents.

Constantino and Daw [2015], in 2015, examined human behaviour in sequential patch foraging experiments virtually. The environments differed in their travel time between patches and the resource depletion rate in individual patches. Using Bayesian model comparison, they found that subjects' behavior was better modeled by the trial-by-trial MVT learning rule than the model-free RL algorithms. Kolling and Akam [2017] in 2017 suggested a combination of model-free and model-based RL algorithms.

While an analytical solution exists to the problem, it only pertains to the patches in environments where the reward rate is monotonically non-increasing. But, animals behave intelligently and get better rewards from patches where rewards can also increase (such patches may be artificial), which is where RL comes into the picture. In the former environment, MVT agents perform faster and better than RL agents due to this constraint. MVT predicts leaving the environment whenever rewards decrease beyond a certain threshold which might be incorrect.

3 Problem Statement

We were provided with a linear foraging environment with reward patches as a optimization game with two levels.

Let t be the travel time between 2 patches, τ is the total time limit and R be the total reward obtained after completion of the game. The constraint are: $\tau \leq 4$ minutes, t is 10 seconds and 3 seconds for 2 levels of the game. The optimization problem required to solve is: $Maximise(R)$

4 Environment Details and Implementation

Let n be the number of times a patch is harvested.
We dealt with two environments, first a simple one where initial reward of each patch is 7, and second, an environment where there are rich and poor patches at random and their initial reward vary (**2-7 for a poor patch** and **10-15 for a rich patch**).
State Space: n, state space is consisting of the number of times we have harvested a patch,
Action Space: [0($Leave$), 1($Harvest$)],
Initial State: $h_t = 0$, **Terminal State:** Total Time > 4 mins,
Transition Functions: (Same for both environments)

$$P_{SS'}^{Harvest} = P[S_{t+1} = n+1 | S_t = n, A_t = Harvest] = 1$$

$$P_{SS'}^{Leave} = P[S_{t+1} = 0 | S_t = n, A_t = Leave] = 1$$

Reward Functions: for simple environment with constant initial reward for each patch

$$R_{SS'}^{Harvest} \sim 7 - 0.5n + \mathcal{N}(0, 0.025) | S_t = n, A_t = Harvest$$

$$R_{SS'}^{Leave} = 0 | S_t = n, A_t = Leave$$

Reward Functions: for complex (2nd) environment with rich and poor patches appearing in random order.

$$R_{SS'}^{Harvest} \sim \Re - 0.5n + \mathcal{N}(0, 0.025) | S_t = n, A_t = Harvest$$

where $\Re$ is uniformly distributed integer in the range-[2, 7] for a poor patch and [10, 15] for a rich patch, with poor and rich patches appearing in random order.

$$R_{SS'}^{Leave} = 0 | S_t = n, A_t = Leave$$

For the harvest time, we have sampled uniformly from a range of 0.6 to 1.4 seconds. Environment Rendering

5 Proposed Solution

We use the R-Learning algorithm, which is a kind of Average Reward RL. In this algorithm, average reward per time step under policy π is defined as,

$$p^\pi = \lim_{n \to \infty} \frac{1}{n} \sum_{t=0}^{n} E_\pi[r_t]$$

And then we redefine action-value function for a state-action pair as,

$$Q^\pi(s, a) = E_\pi[\sum_{t=0}^{\infty} r_{t+1} - p^\pi | s_0 = s, a_0 = a]$$

Here the action-value estimate would be given by,

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

And the estimate of average reward per time step would be,

$$\rho \leftrightarrow \frac{\rho * e + \mathbb{R}}{e + 1}$$

where e is the total number of episodes completed and $\mathbb{R}$ is the total reward per unit time in the $(e+1)^{th}$ episode.

For the rich-patch-poor-patch environment, we propose the VPG agent as the best performing agent for this environment in the domain of our experiments. The VPG agent has separate networks for policy and value function optimization. The respective loss functions are given by:

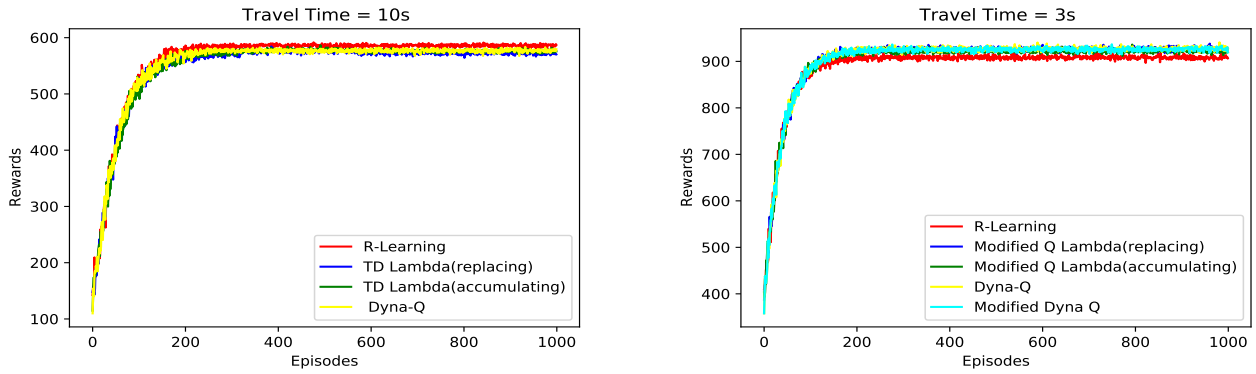
$$L_\pi(\theta) = -\frac{1}{N} \sum_{n=0}^{N} [(G_t - V(S_t; \phi)) log \pi(A_t | S_t; \phi) + (\pi(A_t | S_t; \phi))]$$

$$L_v(\phi) = -\frac{1}{N}\sum_{n=0}^{N}(G_t - V(S_t;\phi))^2$$

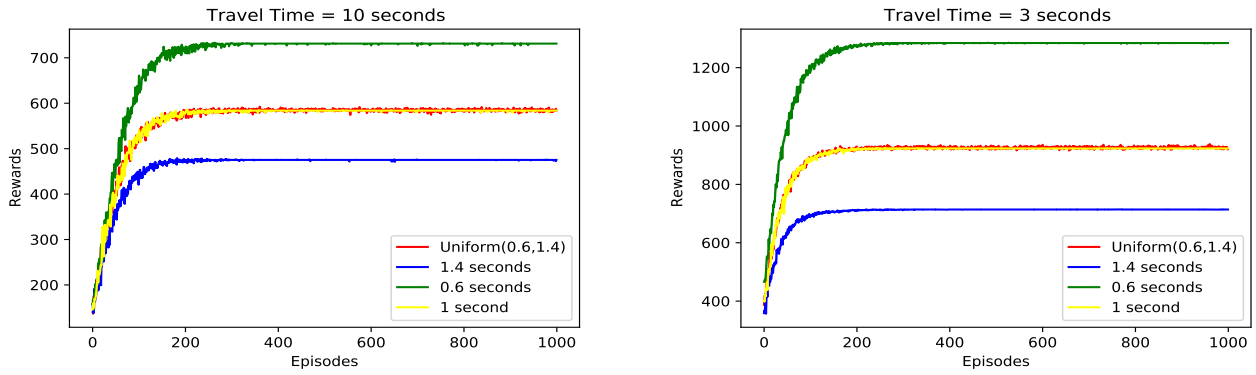
Here $L_\pi(\theta)$ is the loss function corresponding to the policy network θ and $L_v(\phi)$ is the loss function corresponding to the value network ϕ. S_t, A_t are current state and action respectively. $V(S_t;\phi)$ is the value function of the current state following a policy parameterized by ϕ, and G_t is the discounted return from the current state. π is the paramterized probabilistic policy and N is the number of input samples.

6 Experiments

Single Patch Environment: The environment is finite, with finite state space and only two possible actions, either to harvest or to leave the current patch. Therefore we decided to implement various off-policy control algorithms to solve the problem. After comparing Q-learning, Double Q-learning and R-learning, we concluded that the average reward updation works best for the problem. We modified various algorithms like Q(λ), Dyna-Q and Trajectory-sampling to update their values using the average reward function and obtained the following results.

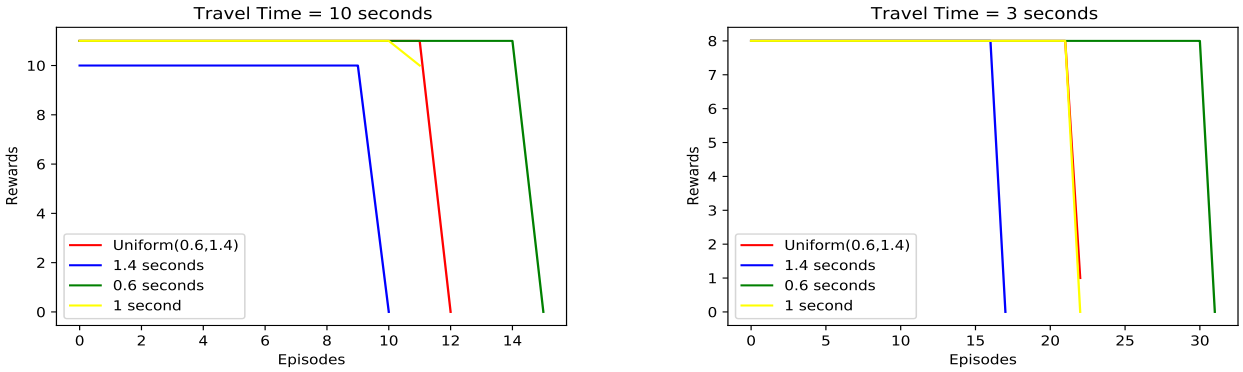


From the above plots, it is clear that R learning performs most optimally in the environment where leave time is 10 seconds, and modified Q(λ) performs best for the 3 seconds leave time environment. We then decided to look into the effect of harvest time on the total rewards. While it is clear that greater harvest time would yield a smaller reward in total, the difference in the total rewards for a slight change in harvest time was pretty vast. We compared for harvest time of 1.4 seconds, 1 second, 0.6 seconds, and $\mathcal{U}$ (0.6, 1.4) seconds.

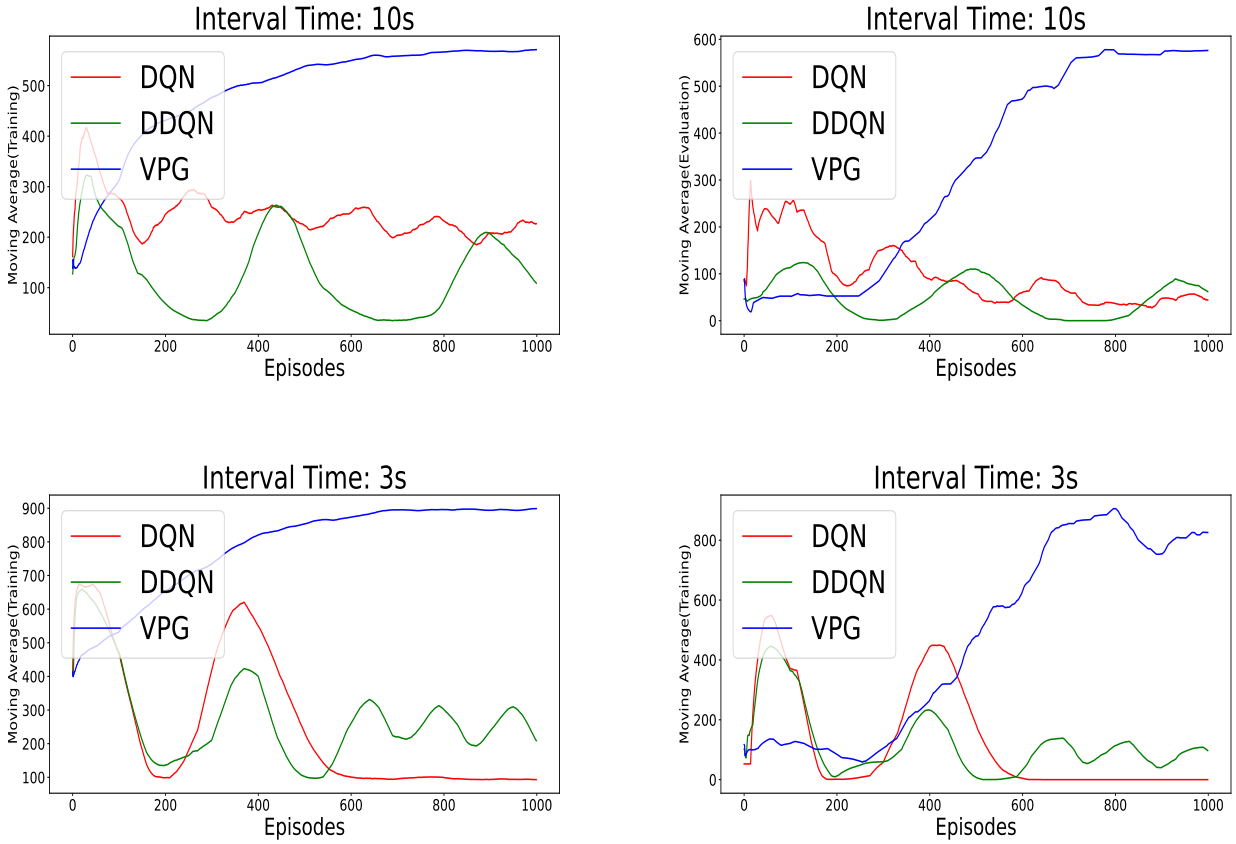


We also compared the relative actions taken by the two agents during different harvest times. The RL agents here harvest 11 times compared to 10 times for a leave time of 10 seconds. This proves that the RL agents learn the model better than the MVT solution, as total rewards considering boundary case is better for 11 harvests. For

a leave time of 3 seconds, the RL agents harvest 8 times which has the same reward as the MVT solution of 7 harvests considering boundary cases.

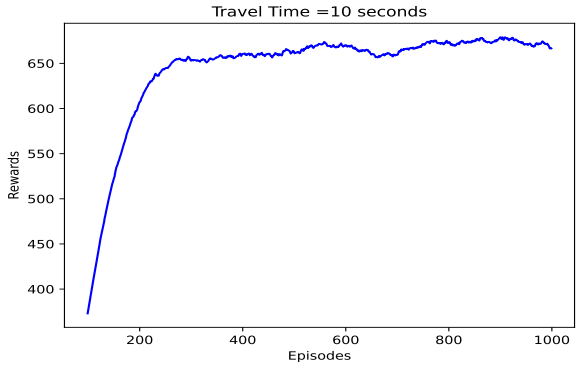


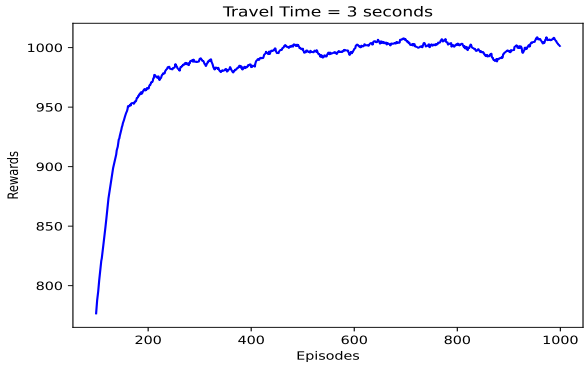
For the deep agents we use a state space representation, given in the paper by Shuvaev et al. [2020]. So we define the current state(S_t) as a vector of 5 latest effective rewards and the latest action. Providing agents with information about the past rewards enables them to learn various behaviors following the same latest reward and the latest action allows the models to distinguish the expected rewards from violations of the reward schedule (i.e., surprises). We implemented DQN, DDQN, and Vanilla Policy Gradient(VPG) in this environement. The average reward plots obtained during training and evaluation for interval time of 10 seconds and 3 seconds is given below(the results are averaged for 6 instances of the environment):



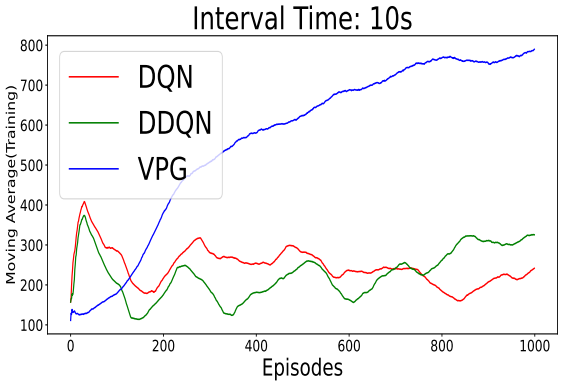
Rich Patch Poor Patch Environment: With the agents performing optimally for the normal environment, we decided to extend the study to a more complex environment with poor and rich patches. The initial reward

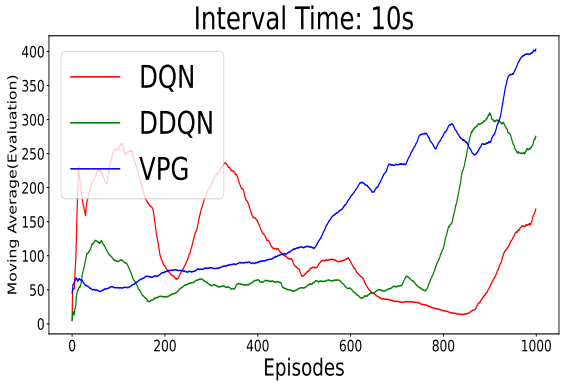
for the rich patch takes in a random integer from [10,15], and the poor patch lies between [2,7]. To frame it into a control algorithm, we took a 2 - dimensional state space. The state space is a [patch] x [harvest] matrix that computes Q values for separate patches separately. We decided to run the R-learning algorithm for this case, as it gave the best reward for the simple environment among Q learning and Double Q learning.

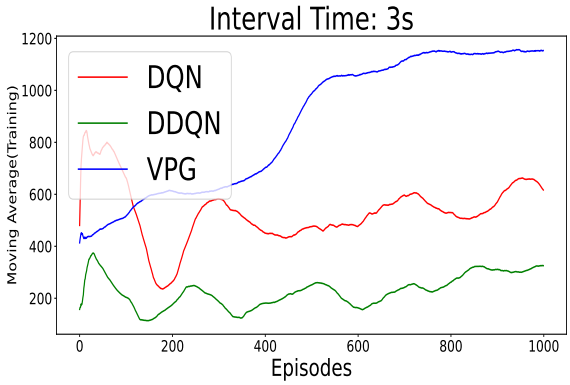


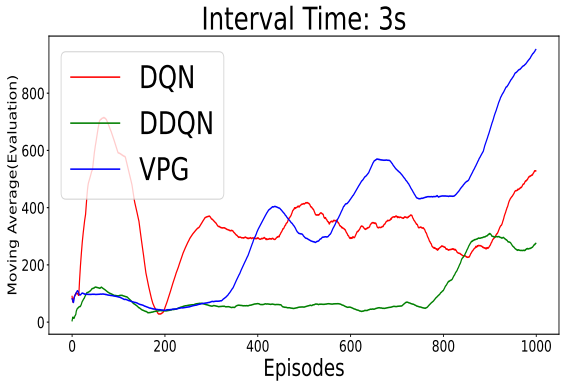


For deep agents, we use the same state space used for the former case.The deep agents that we implemented here were again DQN, DDQN, and Vanilla Policy Gradient(VPG). The average reward plots obtained during training and evaluation for interval time of 10 seconds(the plots were similar for interval time of 3 seconds) is given below(the results are averaged for 6 instances of the environment):









7 Results and Analysis

The conventional RL control algorithms perform optimally for the limited state space environment. The agents have very competing rewards, with average reward updation performing optimally in both the cases. The time to train an instance of tabular RL agents is very less as compared to the deep agents and yields better results as compared to the latter. The tabular control RL agents however are not able to learn the MDP as well as the deep RL agents and yield sub-optimal results.

For the deep agents, the time taken to train the DQN and DDQN agents was quite high. Both of them took around 90 minutes to run for six instances of the environments. And even after that yielded sub-optimal results. In both the environments, they seem to not learn anything even after a period of thousand episodes. The policy-gradient agent VPG, not only is able to perform much better but is also very fast and sample-efficient. It takes less than 30 minutes to train the VPG agent. It is able to generalise and perform better than the value based ones in both the environments. From these results we can also infer that for environments which are changing constantly(in our case for the second environment we have changing reward distribution for each patch), the value based methods would not have good performance as they calculate deterministic policies. What we are in need of are stochastic policies.

8 Future Directions and Conclusions

Considering our findings that policy-gradient methods perform better than value based methods, in future we can move to more general policy-based methods like gradient-free or black-box methods. We can also move towards model-based RL approaches such as Monte Carlo Tree Search, using a combination of model-based RL and model-free RL. This is to enable prospective prediction of the future state of the patch facilitated by model-based RL, and make use of model-free RL to ameliorate planning costs due to deep and precise decision tree search required by model based methods. Research could be also extended to environments where rewards in the patches decreases exponentially on average. We can also experiment with the non-linearity of time. This non-linearity might arise as a flawed perception of time due to a sense of urgency. The algorithms that we implemented could serve as a baseline for such complex environments. We can also experiment with decreasing decision times resulting from the agent learning the MDP of the environment towards the end of the experiment.

We reasoned that R-learning might perform more similarly to the MVT rule because it optimizes the same objective as MVT and does so, in part, by learning an additional average-reward rate term similar to the MVT's decision variable(Constantino and Daw [2015]). But, MVT suffers from the boundary rule where it wrongly predicts the action when an episode is about to end. These RL agents outperform MVT here.

9 Member Contributions

Name	Contribution
Nibir	Implementation of deep agents for both environments - DQN, DDQN and Vanilla Policy Gradient, 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(λ), TD(λ) 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(λ), 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.

References

Sara M. Constantino and Nathaniel D. Daw. *Learning the opportunity cost of time in a patch-foraging task*. PhD thesis, 2015.

Nils Kolling and Thomas Akam. (reinforcement?) learning to forage optimally. *Current Opinion in Neurobiology*, September Edition, 2017.

Duff M.O. Dayan P. Niv, Y. Dopamine, uncertainty and td learning. In *Behav Brain Funct 1, 6*, 2005. URL https://rdcu.be/cBjNu.

Sergey Shuvaev, Sarah Starosta, Duda Kvitsiani, Adam Kepecs, and Alexei Koulakov. R-learning in actor-critic model offers a biologically relevant mechanism for sequential decision-making. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 18872–18882. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/da97f65bd113e490a5fab20c4a69f586-Paper.pdf.

Appendix

A Experiment Details

HYPER-PARAMETER SETTINGS

1. **Q learning**
   (a) Number of Environments: 10
   (b) Number of episodes : 1000
   (c) α: Exponential decay from 1 to 0.01 in 1000 steps divided by 1000
   (d) ϵ: Exponential decay from 1 to 0.0001 in 400 steps then constant at 0.0001 for rest 600 steps
   (e) γ : 1
   (f) Seeds: 50 to 59
2. **Double Q learning**
   (a) Number of Environments: 10
   (b) Number of episodes : 1000
   (c) α: Exponential decay from 1 to 0.01 in 1000 steps divided by 1000
   (d) ϵ: Exponential decay from 1 to 0.0001 in 400 steps then constant at 0.0001 for rest 600 steps
   (e) γ : 1
   (f) Seeds: 50 to 59
3. **R-learning**
   (a) Number of Environments: 10
   (b) Number of episodes : 1000
   (c) α: Exponential decay from 1 to 0.01 in 1000 steps divided by 1000
   (d) ϵ: Exponential decay from 1 to 0.0001 in 400 steps then constant at 0.0001 for rest 600 steps
   (e) γ : 1
   (f) Seeds: 50 to 59
4. **Q(λ)**
   (a) Number of Environments: 10
   (b) Number of episodes : 1000
   (c) α: Exponential decay from 1 to 0.01 in 1000 steps divided by 1000
   (d) ϵ: Exponential decay from 1 to 0.0001 in 400 steps then constant at 0.0001 for rest 600 steps
   (e) γ : 1
   (f) λ : 0.5
   (g) Seeds: 50 to 59
5. **TD(λ)**
   (a) Number of Environments: 10
   (b) Number of episodes : 1000
   (c) α: Exponential decay from 1 to 0.01 in 1000 steps divided by 1000
   (d) ϵ: Exponential decay from 1 to 0.0001 in 400 steps then constant at 0.0001 for rest 600 steps
   (e) γ : 1

(f) λ : 0.5

(g) Seeds: 50 to 59

6. **Modified Q(λ)**

(a) Number of Environments: 10

(b) Number of episodes : 1000

(c) α: Exponential decay from 1 to 0.01 in 1000 steps divided by 1000

(d) ϵ: Exponential decay from 1 to 0.0001 in 400 steps then constant at 0.0001 for rest 600 steps

(e) γ : 1

(f) λ : 0.5

(g) Seeds: 50 to 59

7. **Dyna Q**

(a) Number of Environments: 10

(b) Number of episodes : 1000

(c) α: Exponential decay from 1 to 0.01 in 1000 steps divided by 1000

(d) ϵ: Exponential decay from 1 to 0.0001 in 400 steps then constant at 0.0001 for rest 600 steps

(e) γ : 1

(f) noPlanning : 2

(g) Seeds: 50 to 59

8. **Modified Dyna Q**

(a) Number of Environments: 10

(b) Number of episodes : 1000

(c) α: Exponential decay from 1 to 0.01 in 1000 steps divided by 1000

(d) ϵ: Exponential decay from 1 to 0.0001 in 400 steps then constant at 0.0001 for rest 600 steps

(e) γ : 1

(f) noPlanning : 2

(g) Seeds: 50 to 59

9. **Trajectory Sampling**

(a) Number of Environments: 10

(b) Number of episodes : 1000

(c) α: Exponential decay from 1 to 0.01 in 1000 steps divided by 1000

(d) ϵ: Exponential decay from 1 to 0.0001 in 400 steps then constant at 0.0001 for rest 600 steps

(e) γ : 1

(f) maxTrajectory : 2

(g) Seeds: 50 to 59

10. **Modified Trajectory Sampling**

(a) Number of Environments: 10

(b) Number of episodes : 1000

(c) α: Exponential decay from 1 to 0.01 in 1000 steps divided by 1000

(d) ϵ: Exponential decay from 1 to 0.0001 in 400 steps then constant at 0.0001 for rest 600 steps

(e) γ : 1

(f) maxTrajectory : 3

(g) Seeds: 50 to 59

11. **DQN Design Choices:**

(a) Value Network: State-in values-out feedforward neural network (nodes: 6, 512, 128,2)

(b) Train Exploration Strategy: Exponentially Decaying Epsilon Greedy, decay from 1.0 to 0.3 in 20000 steps

(c) Replay Buffer Maximum Size = 50000, Sample Batch Size = 64

(d) Update Frequency = 10 steps

(e) Evaluation Exploration Strategy: Greedy

(f) Train Episodes: 1000, Evaluation episodes: 1

(g) RMSProp as optimizer (Learning Rate = 0.0007)

(h) Mean Square Error Loss Function

(i) $\gamma = 0.99$

12. **DDQN Design Choices:**

(a) Target and Online Value Network: State-in values-out feedforward neural network (nodes: 6, 512, 128,2)

(b) Train Exploration Strategy: Exponentially Decaying Epsilon Greedy, decay from 1.0 to 0.3 in 20000 steps

(c) Replay Buffer Maximum Size = 50000, Sample Batch Size = 64

(d) Update Frequency = 15 steps

(e) Evaluation Exploration Strategy: Greedy

(f) Train Episodes: 1000, Evaluation episodes: 1

(g) RMSProp as optimizer (Learning Rate = 0.0007)

(h) Huber Loss Function(Gradient norm clipping to 1e7)

(i) $\gamma = 0.99$

13. **VPG Design Choices:**

(a) Policy Network: State-in values-out feedforward neural network (nodes: 4, 128, 64, 2)

(b) Value Network: State-in values-out feedforward neural network (nodes: 4, 256, 128, 1)

(c) Policy Optimizer: Adam(Learning Rate = 0.0005)

(d) Value Optimizer: Adam(Learning Rate = 0.0007)

(e) Train Episodes: 1000, Evaluation episodes: 1

(f) $\gamma = 0.99$

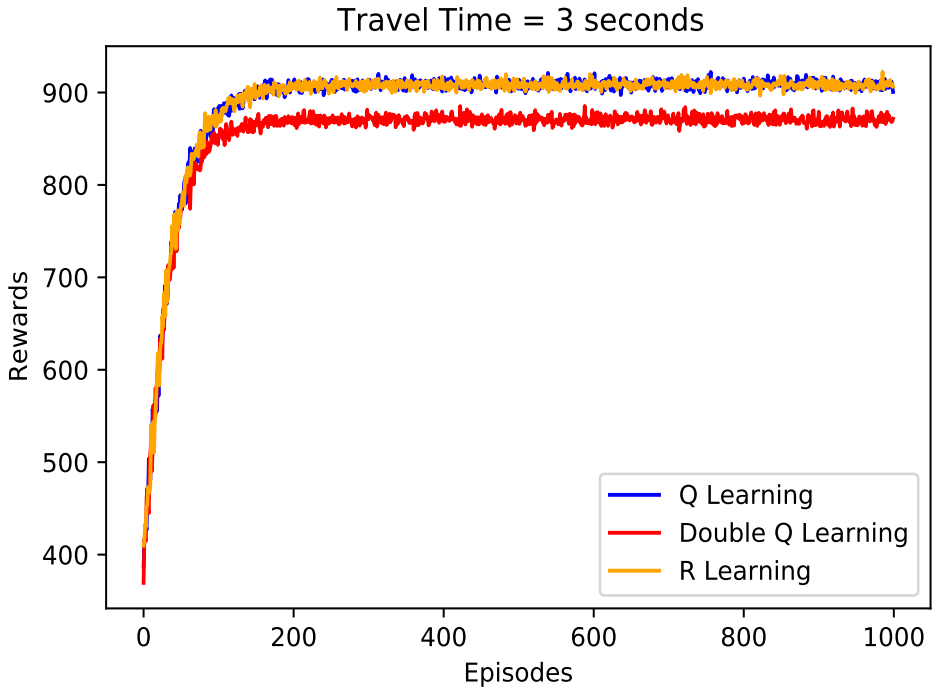
(g) Entropy loss weight $\beta = 0.001$

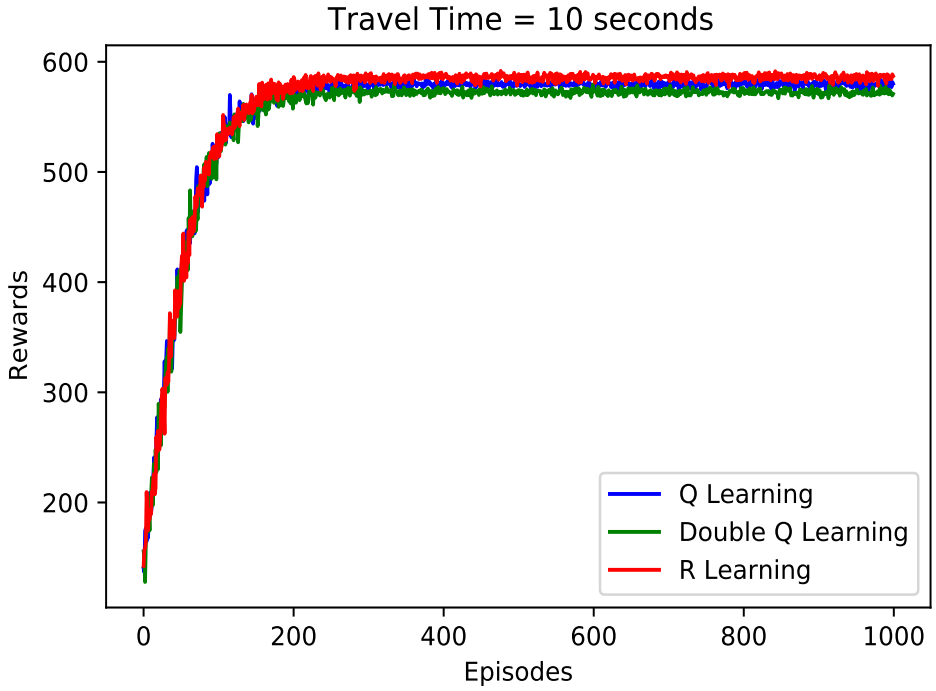
(h) Policy Gradient Norm Clipping to 1.0

(i) Value Gradient Norm Clipping to ∞

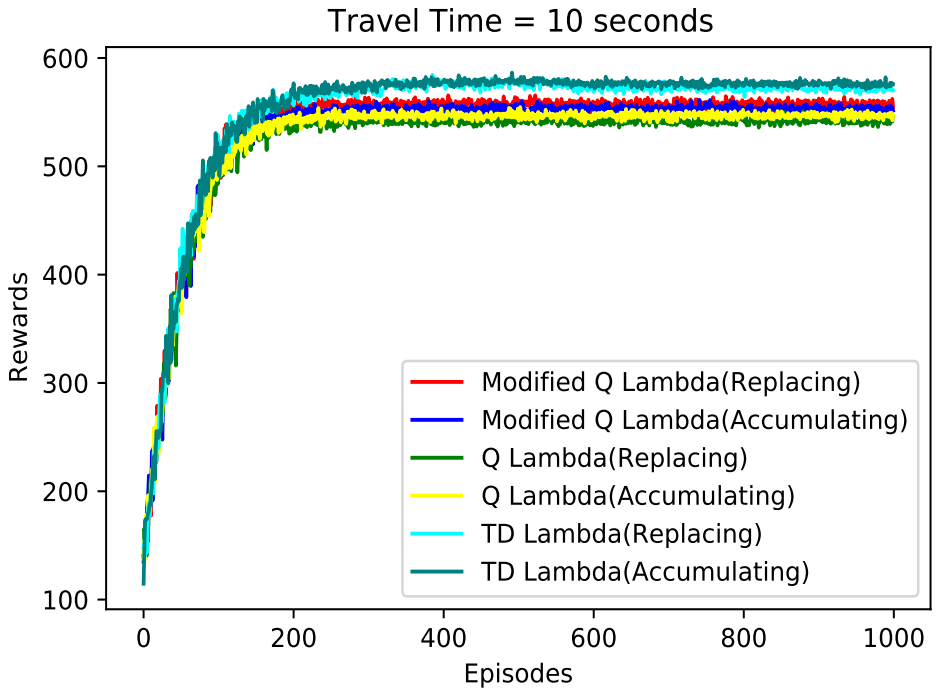
B Extra Plots

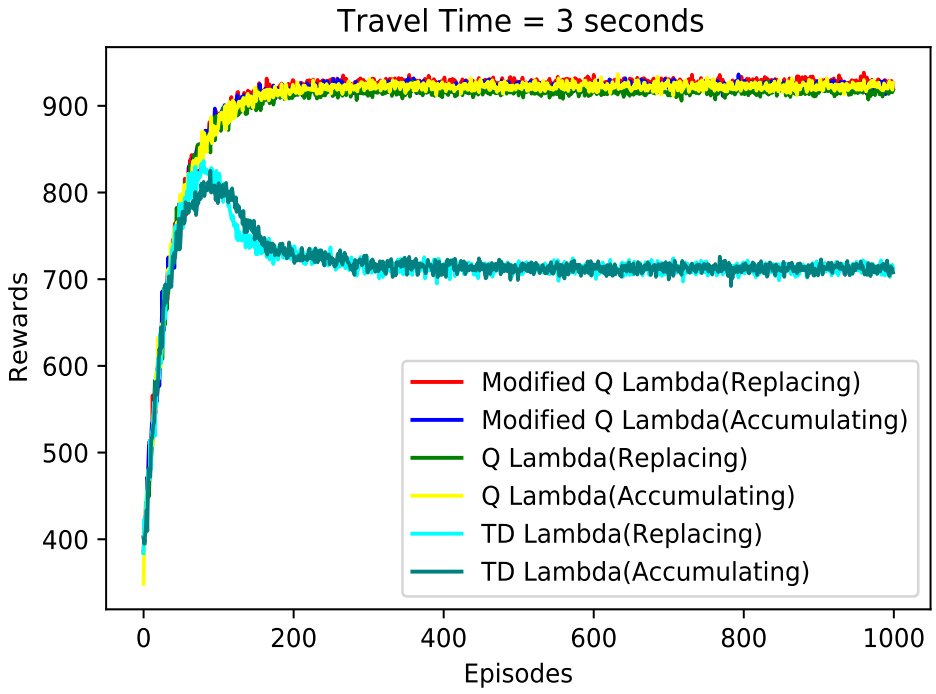
Single Patch Environment:
Comparison between Q learning, double Q learning, and R learning



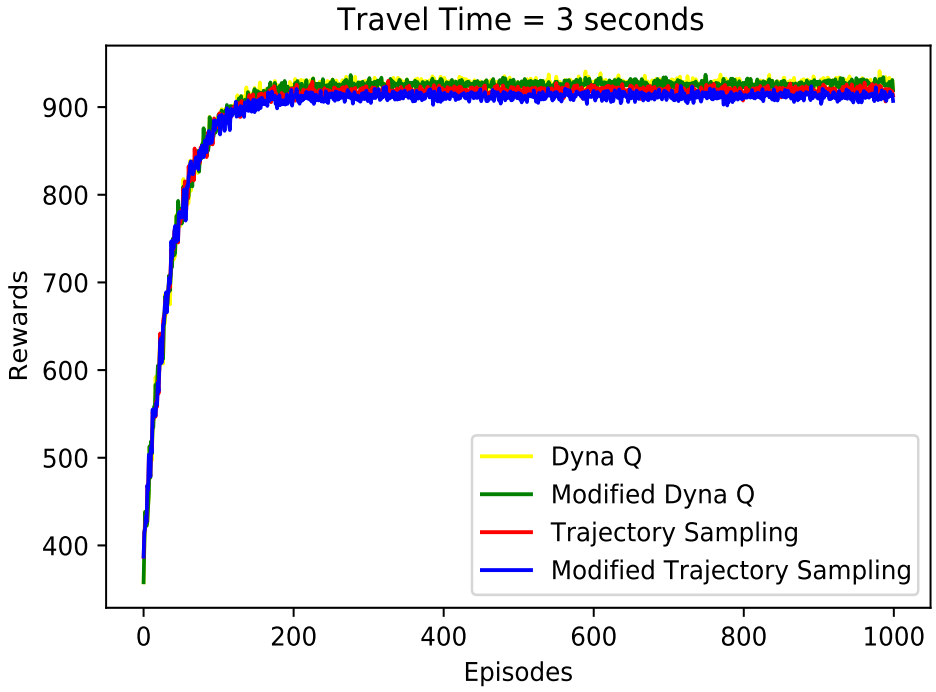


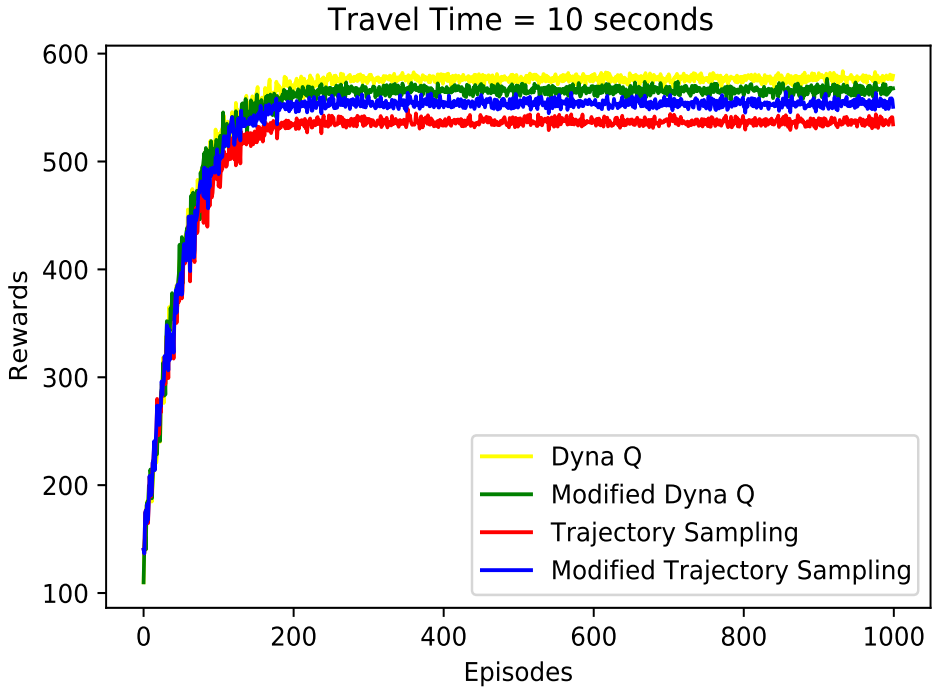
Comparison between Q(λ), modified Q(λ) and TD(λ)



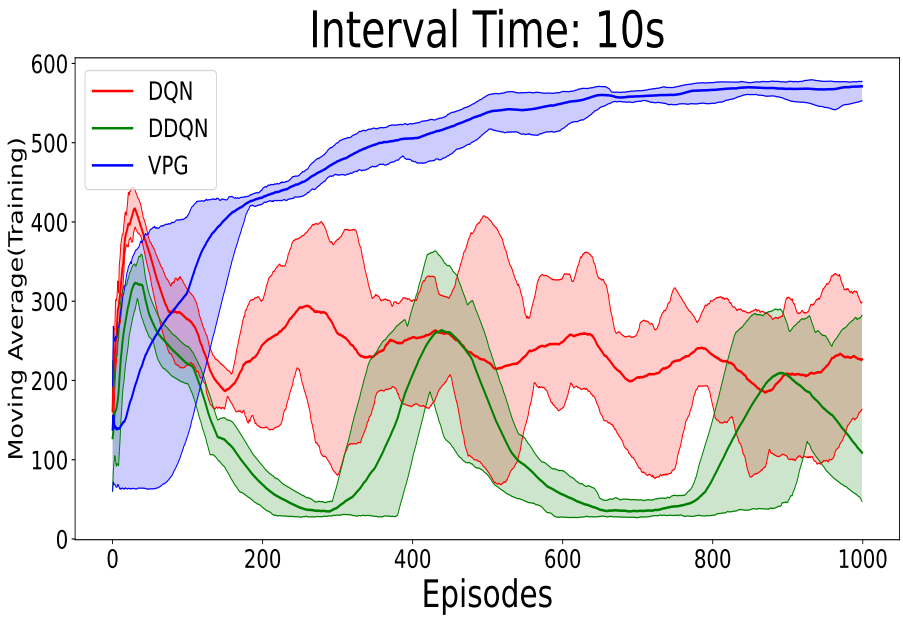


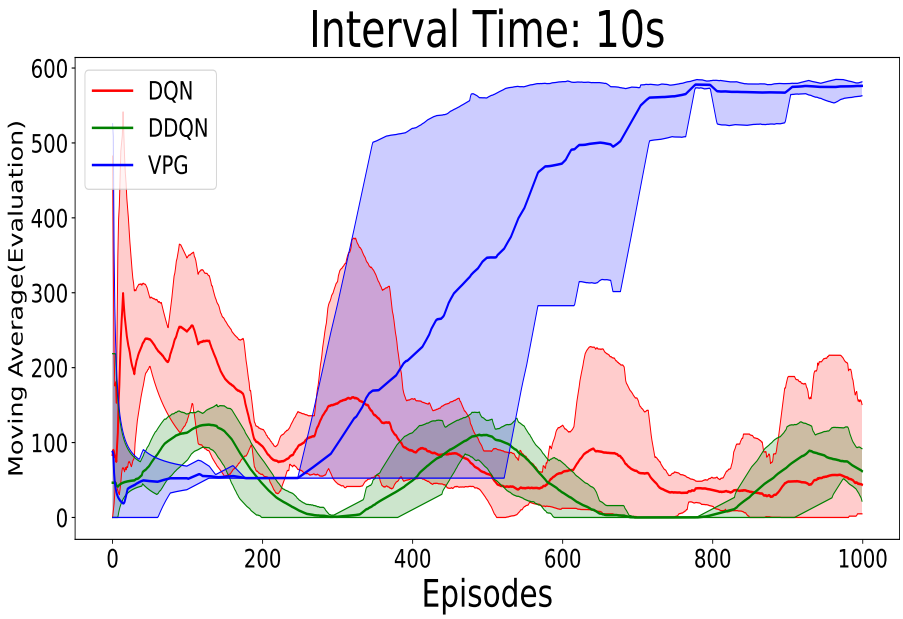
Comparison Dyna Q, modified Dyna Q, Trajectory sampling, modified Trajectory sampling

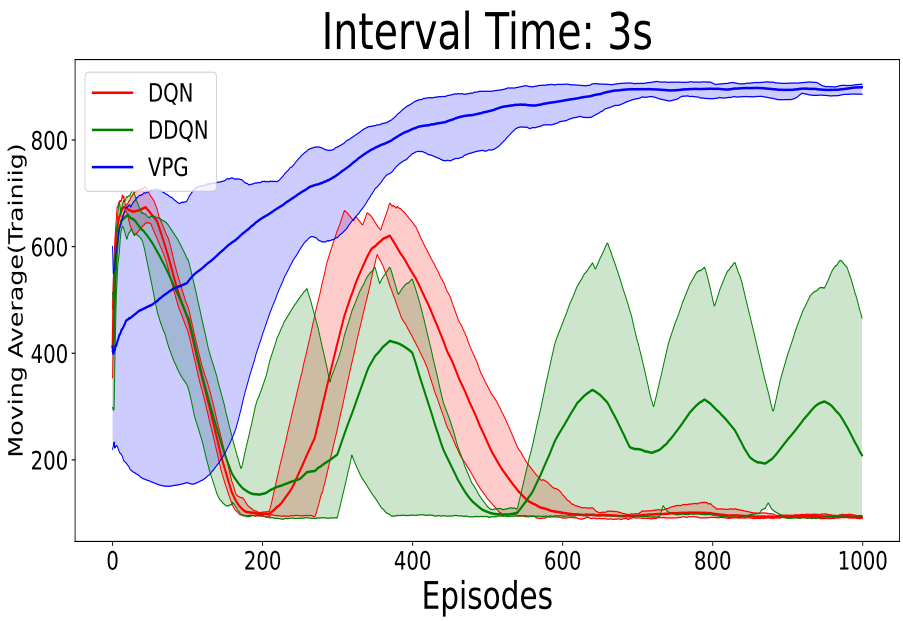


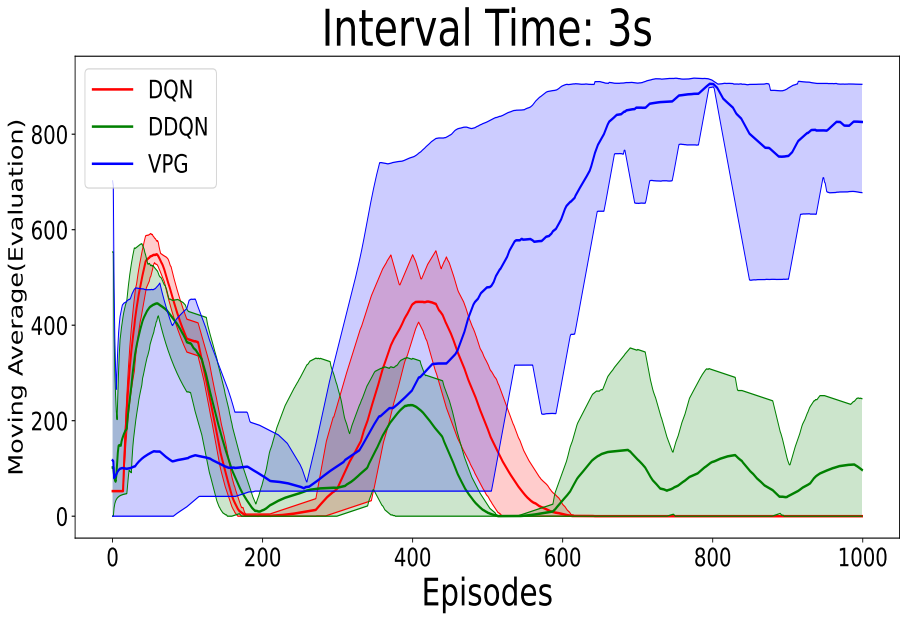


Single Patch Environment deep RL approach plots:

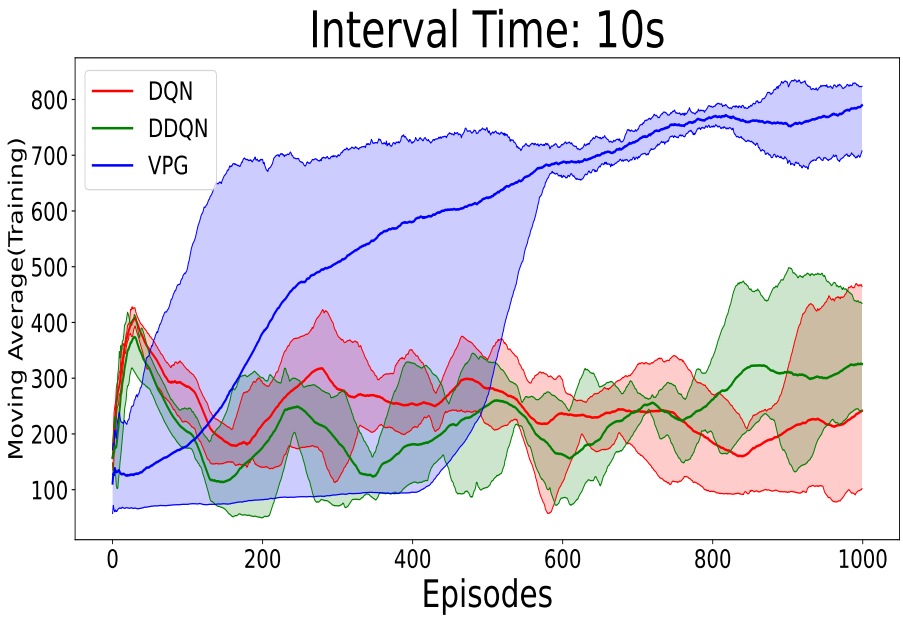


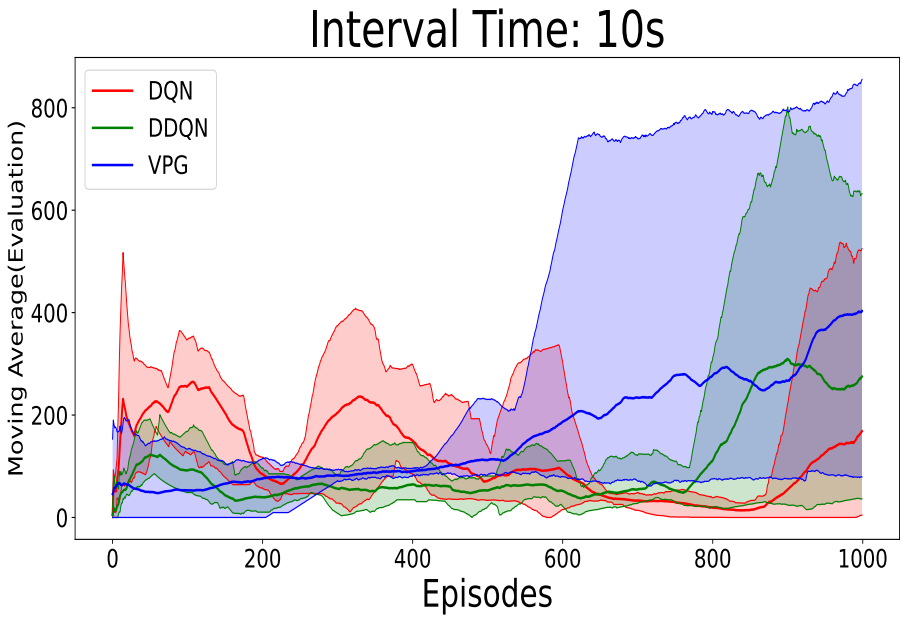


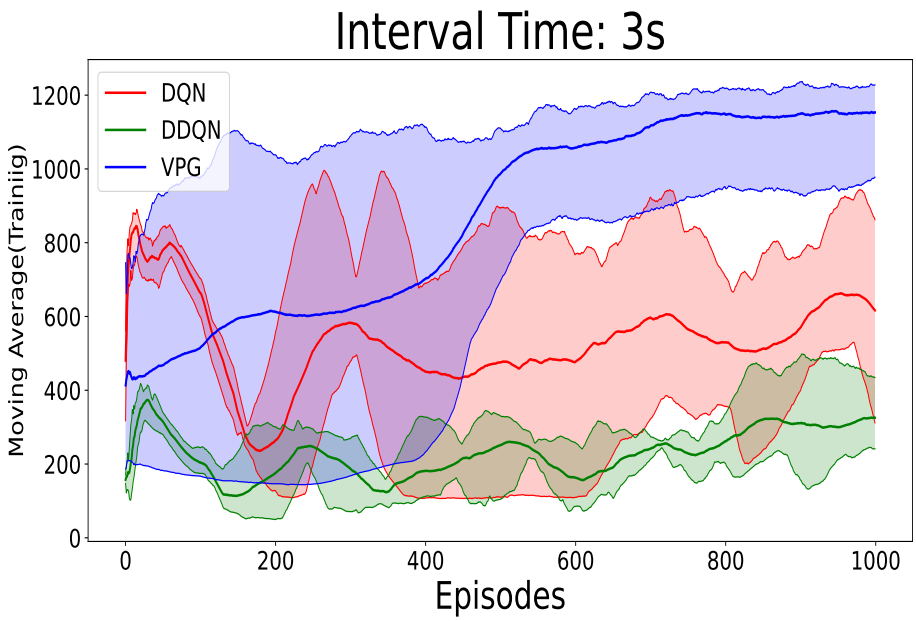

Interval Time: 3s
DQN
DDQN
VPG
Moving Average(Trainiig)
800
600
400
200
0
200
400
600
800
1000
Episodes

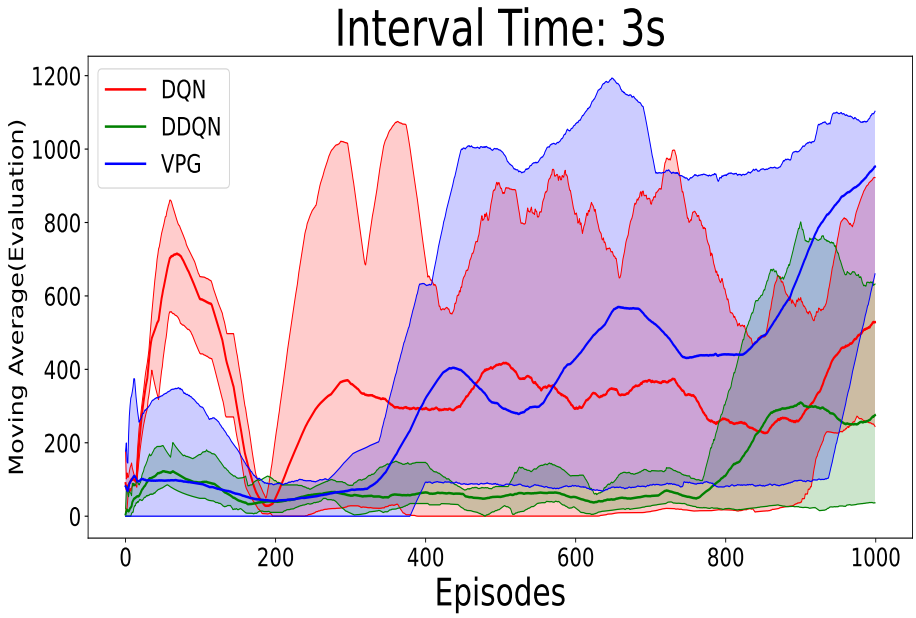

Interval Time: 3s
DQN
DDQN
VPG
Moving Average(Evaluation)
800
600
400
200
0
0
200
400
600
800
1000
Episodes

Rich Patch Poor Patch Environment deep RL approach plots:






Interval Time: 3s
DQN
DDQN
VPG
Moving Average(Trainiig)
Episodes
0
200
400
600
800
1000
1200


Interval Time: 3s
DQN
DDQN
VPG
Moving Average(Evaluation)
Episodes
0
200
400
600
800
1000
1200

C Conclusion

The benchmark of MVT outperforming RL agents is valid for the given environment and true in general for all settings found in nature. But there are some exceptions in some specific kinds of environments. Let's say we successfully create an artificial environment where rewards do not decrease monotonically, i.e., may show rising and falling reward trends, here MVT predicts to leave the patch once the prize goes beyond a certain threshold. Still, maybe it would be even more rewarding to stay and harvest to get a larger reward. An RL agent can learn these things. MVT's assumption of this monotonically decreasing reward helps it to learn faster than RL agents.
These can be found in foraging for knowledge on the internet or money-making on stock markets, and the results found above can be extended to these problems as well. However, their modeling may be highly computationally intensive and challenging, which may be achievable with the use of advanced algorithms like Soft Actor-Critic(SAC), Proximal Policy Optimization(PPO) and Trust Region Policy Optimization (TRPO) and advent of quantum computers.