Evaluating robustness of DQN variants for offline/batch learning

Fahmid Morshed Fahid (ffahid), Tasmia Shahriar (tshahri), and Dan Carpenter (dcarpen2)

Abstract

Deep Q-Networks (DQN) are the modern standard for deep reinforcement learning. Recently, many popular variations in DQN algorithms have emerged, such as Dueling DQN, DQN with experience replay, and priority-based DQN. While these models have demonstrated excellent performance on model-based learning environments, where the agent acts in a completely known environment that can be simulated and learned from, the applicability of DQN models to batch learning has not been thoroughly investigated. In batch learning, where the agent's model of its environment is incomplete and uncertain, many of these algorithms fail to achieve satisfactory performance. Also, in the case of a delayed reward, DQN algorithms often suffer with limited data. In this paper, we analyze and evaluate the robustness of different existing DQN algorithms in a batch learning context. Our results show that DQNs are somewhat effective in a batch learning context when there are immediate rewards available in the environments but are substantially worse in the case of delayed rewards. We also found that increasing data size only helps when the environment is large and complex enough to learn. Our results suggest that Vanilla DQN performs better than other more complex DQN variants in terms of convergence time and reward collection in presence of immediate reward in a deterministic environment.

Introduction

Recent developments in deep reinforcement learning (DRL) algorithms have shown significant improvement in sequential decision making problem domains such as self-driving cars, robot control, and board games [1, 2, 3]. Usually, to learn a good policy, such algorithms need millions of examples so that the agent can self-learn using some given reward model. As such, much of the work is solely based on simulated environments where the agents know the transition models and have infinite supply of example episodes. This is not the case in a real world scenario. For example, in the education domain, we don't have a perfect simulated environment to model students' learning processes and knowledge acquisition, as it is hard to model human behaviour. This is also true for medical diagnosis. In such cases, we only have a limited supply of collected data and collecting further data is often not feasible or not cost effective.

Generally speaking, there are two broad categories of reinforcement learning (RL), namely online learning and offline or batch learning[4]. In online reinforcement learning, the agent has access to an environment (i.e., they have knowledge about the transition model and reward model) and learns an optimum policy by continuously interacting with the environment. This means that the agent has an infinite supply of data and can always infer what will happen if certain decisions are made. This type of learning is appropriate for domains where the state representation is clear, the environment model is proper, and interactions with the environments are relatively cheap. On the other hand, real world environments are rarely known, cannot be modeled perfectly, and collecting new data is often prohibitively expensive. In such cases, we only have a handful amount of data to learn from. This type of learning is known as batch reinforcement learning or offline reinforcement learning. In batch reinforcement learning or offline reinforcement learning, the agent has a fixed amount of data and does not have any access to the environment (i.e., the transition model is unknown) and cannot generate new data. Prior work has shown that this type of learning is challenging [5].

Deep Q Networks (DQN) combine Q learning with deep neural networks and often scales nicely across different environments [3]. Having a neural network embedded in the process, this particular approach can approximate the Q values in large state space and action spaces. The Q learning model is also model-free, meaning that it does not require a transition model to learn from, instead using a bootstrap maximization according to the Bellman equation [4]. Having such perks, this particular approach has been exhaustively investigated by researchers in recent years and many significant improvements or variants have been made to address certain shortcomings or challenges faced by DQNs [6]. One particular drawback of DQN is that it requires a large number of examples. Thus, many variants of DQNs are experimented on and observed in online learning contexts. Nothing much has been said about their usage in case of limited data, more specifically, model free or batch learning.

In this work, we investigated the performance of different DQN variants in a batch learning context. Specifically, we evaluated the performance of four DQN variants that were trained on batch data and without any access to the environment. In that regard, we used three different environments (two custom made) to collect fixed batches of data. We compared their total reward based on two specific reward types in the data, namely, immediate and delayed reward. Our finding shows that DQNs are somewhat effective in batch learning context when there are immediate rewards available in the environments but are substantially worse in the case of delayed rewards. We also investigate the effect of increased episode size on DQN performance and analyze the overall performance to understand which of the DQN variants perform well in an offline environment. All of our work is available at github at the following link: https://github.com/FahmidMorshed/EvalDQNsOffline.git

Background

Reinforcement learning is a branch of machine learning where an agent learns a decision policy in an environment in order to maximize some reward signal [4]. Typically, these problems are sequential learning problems in nature and often there are state space and action spaces to explore and exploit. The key difference between reinforcement learning and supervised learning is that, in supervised learning, the agent is given some target whereas in reinforcement learning, the agent explores the environment to learn by itself using reward.

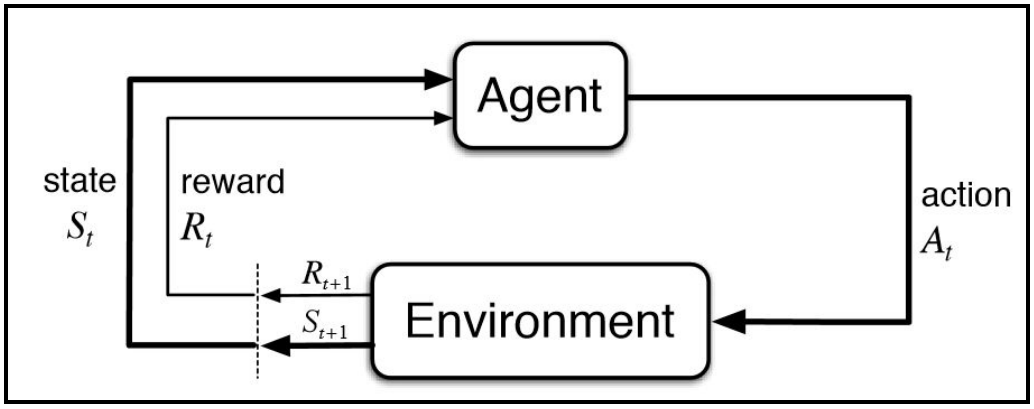


Figure 1: Reinforcement learning agent and environment

Reinforcement learning:
In typical reinforcement learning, an agent interacts with a given environment at timestep $t = 1, 2, 3, \ldots$. The agent is given a description of its current state, S_t, takes an action A_t, gets a reward R_{t+1} and moves to a new state S_{t+1} with transition probability $T(s, a, s') = P[S_{t+1} = s' | S_t = s, A_t = a]$. This representation is known as a Markov Decision Process (MDP). Formally, MDP is a discrete time stochastic control process where the assumption is that a single state is sufficient to represent everything about that state and no historical knowledge (i.e., previous states) are needed and is represented by a tuple $< S, A, T, r, \gamma >$ where S is a finite set of states. Here, $\gamma \in [0, 1]$ is a discount factor and is typically constant. A transition gives a reward $r(s, a) = E[R_{t+1} | S_t = s, A_t = a]$.

Q learning:
The agent learns a decision policy π that is a probability distribution over actions for each state, in deterministic case, a mapping function based on the state. At timestep t, on state S_t, the agent learns a policy that maximizes the discounted return or discounted sum of future rewards $G_t = \sum_{k=0}^{\inf} \gamma^k R_{t+k+1}$. To do so, there are many approaches in reinforcement learning such as value based learning and policy based learning. In value based RL, the agent learns an estimated expected discounted return when following a policy π of a state $V^{\pi}(s) = E_{\pi}[G_t | S_t = s]$. A similar, but slightly more explicit learning, is trying to estimate expected discounted return using a Q value which is a state-action pair value following policy π as $Q^{\pi}(s, a) = E_{\pi}[G_t | S_t = s, A_t = a]$. In Q learning, the agent estimates the Q value based on an experience tuple $< S_t = s, A_t = a, S_{t+1} = s', R_{t+1} = r >$ using the Bellman equation as follows:

$$Q^{\pi}(s, a) = r(s, a) + \gamma \; max_{a' \in A} Q(s', a')$$

Deep Q Network:

The limitation of Q learning is that Q learning is not scalable. This is especially true in large state spaces where it is not feasible to estimate each state action pair independently. To workaround this limitation, a more suitable function approximation approach is often used. Neural networks are a fairly good approximator and in theory can approximate any function. In 2015, Mnih et al. [3] showed that neural networks can be integrated into the Q learning framework to achieve significant success in large state and action space. In their work, Mnih et al. added a convolutional neural network in Q learning to learn 47 Atari 2600 games and found it at least as good as expert human level performances. The core idea of DQN is to use a ε-greedy exploration and save recent experiences of $< S_t, A_t, R_{t+1}, S_{t+1} >$ in a memory buffer. The parameter of the neural network is optimized at timestep t using a stochastic gradient descent loss minimization as follows:

$$(R_{t+1} + \gamma \; max_{a' \in A} Q_{\theta}(S_{t+1}, a') - Q_{\theta}(S_t, A_t))^2$$

The neural network is defined as θ. The network learns the Q value by sampling batches from the experience reply buffer.

Related Work

Many deep RL work has shown tremendous success in recent years. DRL has been successfully applied to such varied domains as Go, Chess and Shogi, robotic hand dexterity, and physics simulators [2, 7, 8]. While most DRL algorithms have been mainly applied in an online context, very little work has been done on offline RL. This is because of the limitation of the algorithms themselves as they are not suitable for an offline context. Only a handful of algorithms, like Q learning, can be used without any model or environment. Deep learning based Q learning, or DQN [3] has seen significant success in online learning, and can be used in offline learning as well.

Recent work has shown that offline learning can be effectively used in educational context using deep RL. Ausin et al. (2019) [5], showed that DQNs are useful in the case of pedagogical policy learning for undergrad students and showed that higher post test scores can be achieved with AI driven learning policy. Shen et al. [9] applied value iterations on a pre-collected training corpus and found a pedagogical policy that improves student's learning performance. Domains like medical science have also seen significant uses in offline learning using DQNs [10].

Many variants of DQN such as Double DQN [12], prioritized experience replay [13], and dueling DQN [11] have been thoroughly investigated to measure their effectiveness in online learning. Studies have shown that variants as such Dueling DQN are not very effective across different environments whereas prioritized experience replay is very crucial in learning more quickly [6]. But these studies are solely focused on using online learning and nothing much has been done to investigate their efficacy in an offline context. Even though offline learning is getting better in recent times, most of the work only focuses on using vanilla DQN for their study. To our knowledge, no comparative study of different variants of DQNs has been done in an offline context. We believe such a study will be helpful for future researchers, especially in the domains of education or medical science.

Method

9					-9				🟩 6	-3
8			-4							
7	2						9	-5		
6				7						
5										
4										
3						-4			🟩 27	
2			🟩 5							
1										
0	Start				-3			4		
y\x	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

Figure 2: Gridworld. Agent starts at (0,0) and can only move right or up. If the agent moves out of the grid, the simulation ends. The green cells are the best move and a maximum of 5+27+6-1=37 reward can be collected. In non-deterministic cases, there is a 30% chance of performing a random action (right or up).

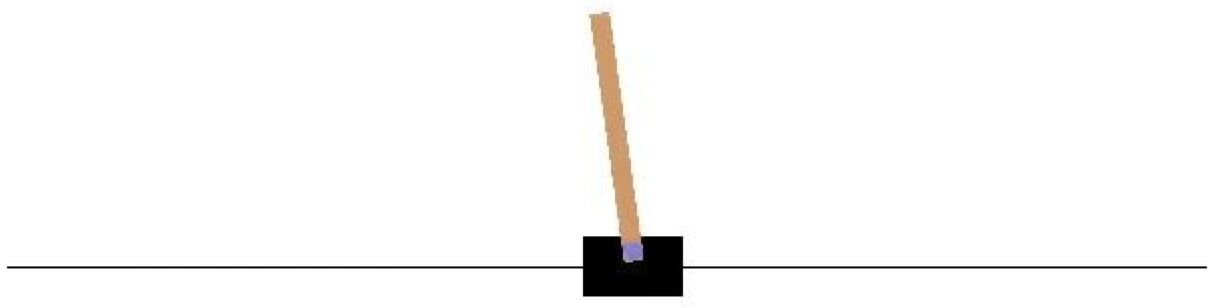


Figure 3: Cartpole environment. At any state, an agent can perform 2 actions, left and right. The goal is to keep the pole balanced. Total reward is the number of seconds the pole is balanced.

Dataset

To investigate the applicability of DQN variants to offline learning tasks, we use data extracted from 3 different environments. These environments were implemented in OpenAI's Gym reinforcement learning environment [14], which is a popular reinforcement learning library. The library provides several classical control problems (e.g., balancing a pole on a movable cart, driving a car up a hill using momentum) and many Atari games (e.g., Asteroids, Centipede, Pong) which are often used as common environments for training and evaluating RL agents. We use one of the most popular environments, "CartPole-v1" (Figure 3) and implement two additional GridWorld environments (Figure 2) of our own using the Gym framework.

In the CartPole environment, the agent's task is to balance a pole on top of a cart for as long as possible. The set of possible actions available to the agent are {*Left, Right*}. That is, at each timestep, the agent decides to either move the cart left or right. For each timestep that the pole remains balanced on top of the cart, the agent receives a reward of +1. The simulation ends when the pole is more than 15 degrees from vertical in either direction, or if the cart moves more than 2.4 units from the center. As a result, a trivial policy where the agent simply moves in one direction is not viable.

The GridWorld environments, which we authored ourselves using the Gym framework, have an agent navigate a simple grid-based environment where certain tiles contain rewards, others contain penalties, and most tiles are empty (Figure 2). The goal is to collect the largest reward. The agent starts in the bottom left corner of the grid, position (0, 0), and is able to move up or to the right (i.e., the set of available actions is $\{$*Up, Right*$\}$). The agent only knows where they are on the grid and does not have any knowledge of the tiles around them. At each timestep, the agent chooses which direction to move in. If the agent moves to a tile with a reward or a penalty, they will collect that value. The simulation ends when the agent moves beyond the bounds of the grid. For the third and final environment, we implemented a nondeterministic version of the GridWorld. In this environment, everything is the same as with the original GridWorld, but there is a 30 percent chance that the agent's action will be randomly selected for them.

We extracted immediate reward as well as delayed reward. In the immediate reward case, the agent gets the reward immediately after taking an action. Whereas, in case of delayed reward, the agent does not get a reward after each step, but gets a cumulative reward at the end of the episode. This type of reward is difficult to learn for an agent because the agent does not know exactly which state action pair is responsible for immediate reward. This is also known as the credit assignment problem.

To use these three environments to explore offline learning with different DQN variants, we extracted batch datasets from each environment. This was done to simulate having access to some data for a reinforcement learning task, but not having direct access to the environment. For each environment, we extracted batch data of size 10k episodes, 5k episodes, and 1k episodes. This data was collected by having an agent act in the environment by randomly selecting from the set of available actions at each timestep. This random agent completed 10k episodes, and this large batch was randomly sampled to create the 5k and 1k datasets. As a result, we had datasets of varying sizes that represented limited amounts of information for each environment that our DQN-based agents would learn from. These datasets captured both immediate and delayed rewards.

DQN Algorithms

We investigated the performance of vanilla DQNs as well as double DQNs, prioritized experience replay DQNs, and dueling DQNs. Each of these DQN variants address some shortcomings of the original DQN framework. Double DQNs address overestimation bias that can be detrimental to learning with DQNs by decoupling the selection of an action from the evaluation of that action [12]. This is done by changing the loss function to the following:

$$\left(R_{t+1} + \gamma_{t+1}\, q_{\bar{\theta}}\left(S_{t+1},\ \underset{a'}{\operatorname{argmax}}\ q_{\theta}\left(S_{t+1},\ a'\right)\right) - q_{\theta}\left(S_t,\ A_t\right)\right)^2$$

Prioritized experience replay DQNs sample more frequently from transitions that have high learning potential rather than sampling from the entire replay buffer uniformly [13]. Instead, transitions are samples with probability p_t, defined below. With this sampling strategy, there is a bias toward recent transitions.

$$p_t \propto \left| R_{t+1} + \gamma_{t+1} \max_{a'} q_{\bar{\theta}}\left(S_{t+1}, a'\right) - q_{\theta}\left(S_t, A_t\right) \right|^{\omega},$$

where ω is a hyperparameter that determines the shape of the distribution.

Dueling DQNs use two streams of computation, value and advantage, share a convolutional encoder, and are merged by a special aggregator [11]. Under the dueling DQN framework, action values are factored as follows:

$$q_{\theta}(s, a) = v_{\eta}\left(f_{\xi}(s)\right) + a_{\psi}\left(f_{\xi}(s), a\right) - \frac{\sum_{a'} a_{\psi}\left(f_{\xi}(s), a'\right)}{N_{actions}},$$

where ξ, η, and ψ are the parameters of the shared encoder f_{ξ}, of the value stream v_{η}, and of the advantage stream a_{ψ}; and $\theta = \{\xi, \eta, \psi\}$ is their concatenation.

Models

We trained several neural networks according to the DQN parameters shown in Table 1. Each DQN variant (1 per row of the table) was used to train 6 neural networks - that it, for each Gym environment (i.e., CartPole, GridWorld, and GridWorld-nondeterministic), one network was trained with immediate reward and one was trained with delayed reward for each episode size (10k, 5k and 1k). In total, we trained 92 neural networks.

Each neural network was trained in Python with the Keras library using TensorFlow backend. Networks all had a single feedforward network with 128 fully-connected units. They were trained with mini batches of size 64, a learning rate of 0.0001, and ReLU activations. With computational complexity being a limiting factor in this work, there was not sufficient time to perform hyperparameter tuning on these networks to improve performance. Additionally, it would be preferable to report results across several different runs of each network to account for differences caused by the random seed used. We leave this as future work.

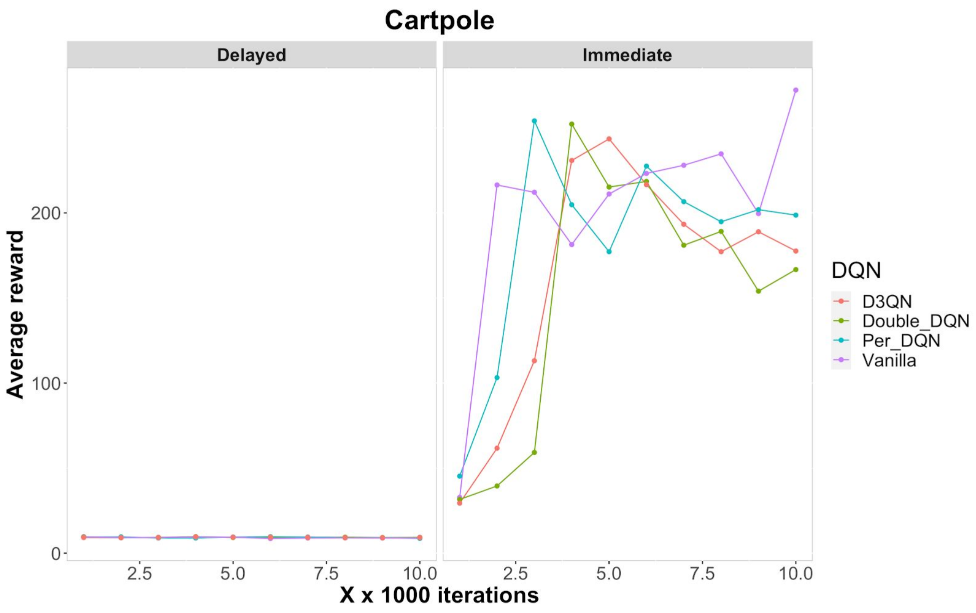
DQN Variant	Dueling	Double Param	Priority Alpha
Vanilla	FALSE	0	0
Prioritized experience replay	FALSE	0	0.05
Double	FALSE	0.5	0.05
Dueling	TRUE	0.5	0.05

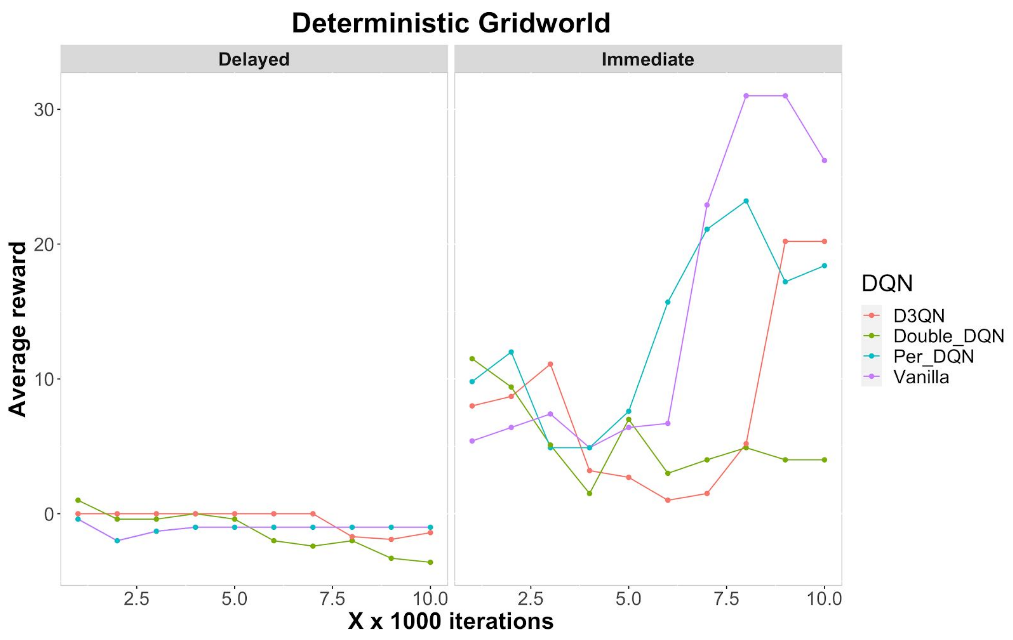
Table 1: The different DQN settings explored in this work

Results

Performance of DQNs on delayed reward versus immediate reward:

The first hypothesis that we test through our experimentation is that all the variants of DQN perform sufficiently well for offline learning only in presence of immediate reward. This hypothesis is supported. An analysis of variance on deterministic gridworld dataset shows that the average rewards collected by all the variants of DQNs in presence of immediate reward (M =10.482 (8.302)) is significantly better (F(1, 75) = 81.500, p = 1.32e-13) than the average reward collected by all the variants in presence of delayed reward (M = -0.998 (0.926)). The similar result is evident in the cartpole dataset (F(1, 75) = 224.995, p = <2e-16) and in non-deterministic gridworld dataset (F(1, 75) = 49.272, p = 8.54e-10). Figure 4 shows the average reward collected by the variants of DQNs after each 1000 iterations in cartpole, deterministic and non-deterministic gridoworld dataset respectively. The graphs also portrait that all the variants perform well in presence of immediate reward. On a deterministic gridworld dataset, the highest reward collected by Vanilla DQN given immediate reward is 26.2, whereas, given delayed reward it is -1. On the cartpole dataset, reward collected by Vanilla DQN is 272.2 and 8.9 given immediate and delayed reward respectively. On the non-deterministic gridworld dataset, reward collected by Vanilla DQN is 10.5 and 4.96 given immediate and delayed reward respectively. All the graphs were generated keeping the episode size constant at 10,000.





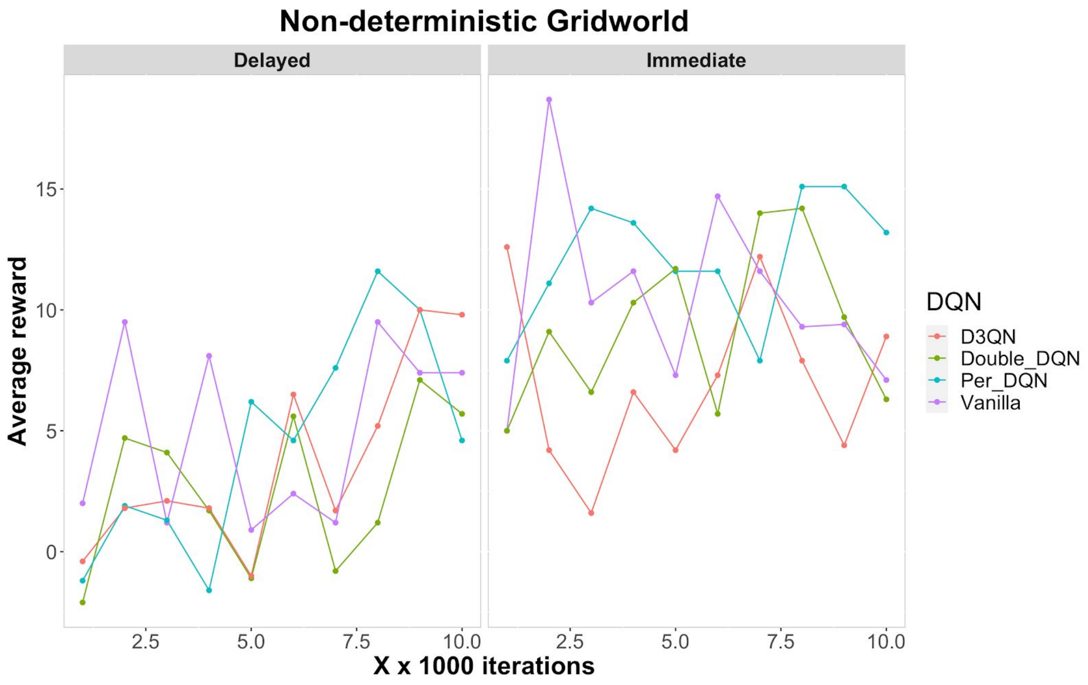
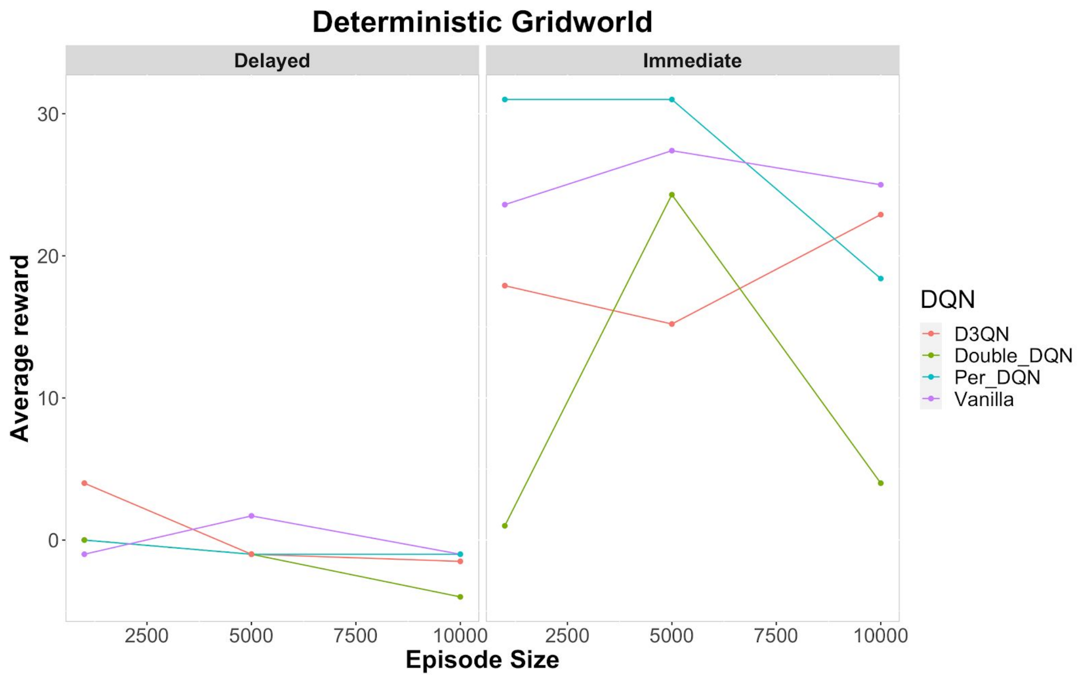
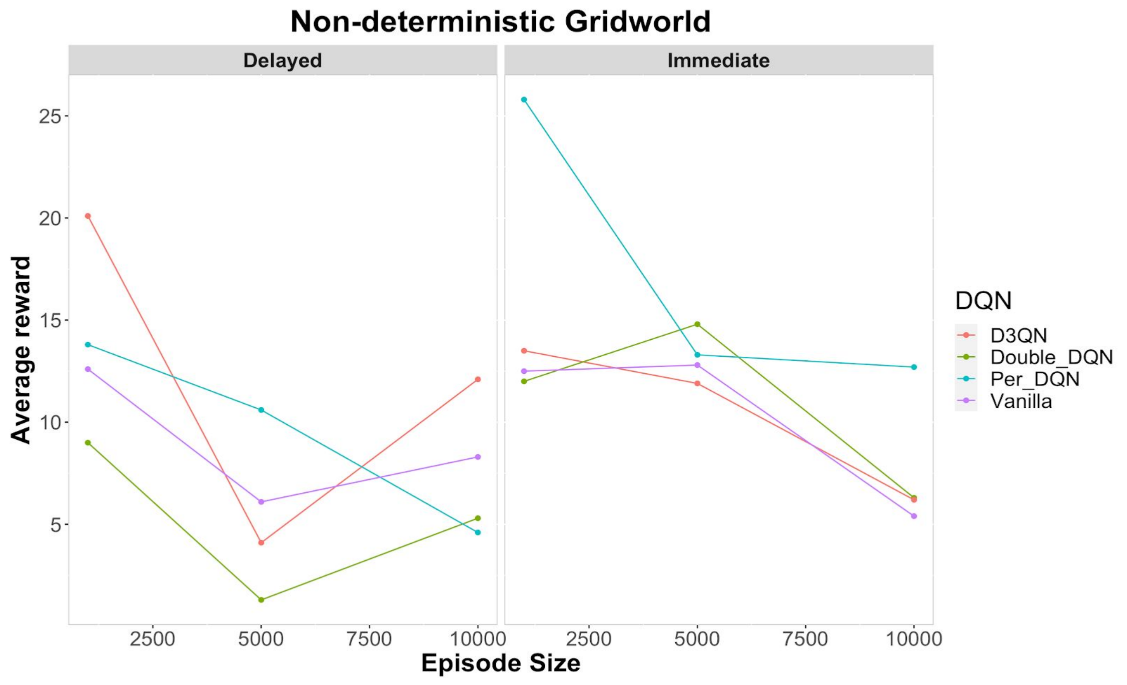


Figure 4: Average reward collected by the variants of DQNs after each 1000 iterations in cartpole, deterministic and non-deterministic gridworld dataset respectively. Y-axis represents the mean of rewards in each 1000 iterations.

Performance as a function of the amount of data available:

The next hypothesis that we are interested in is all the variants of DQN work better with more data, in both delayed and immediate reward. To test this hypothesis, we increased the episode size and calculated the average best reward taken from the last 10 iterations for each of the variants. Figure 5 shows the performance of variants of DQNs with increased episode size in a form of reward curve. We cannot see any evidence of increased performance with increased episode size for any of the environments. The reason behind this may be the environments being less complex to be able to learn by these variants with increased episode size.





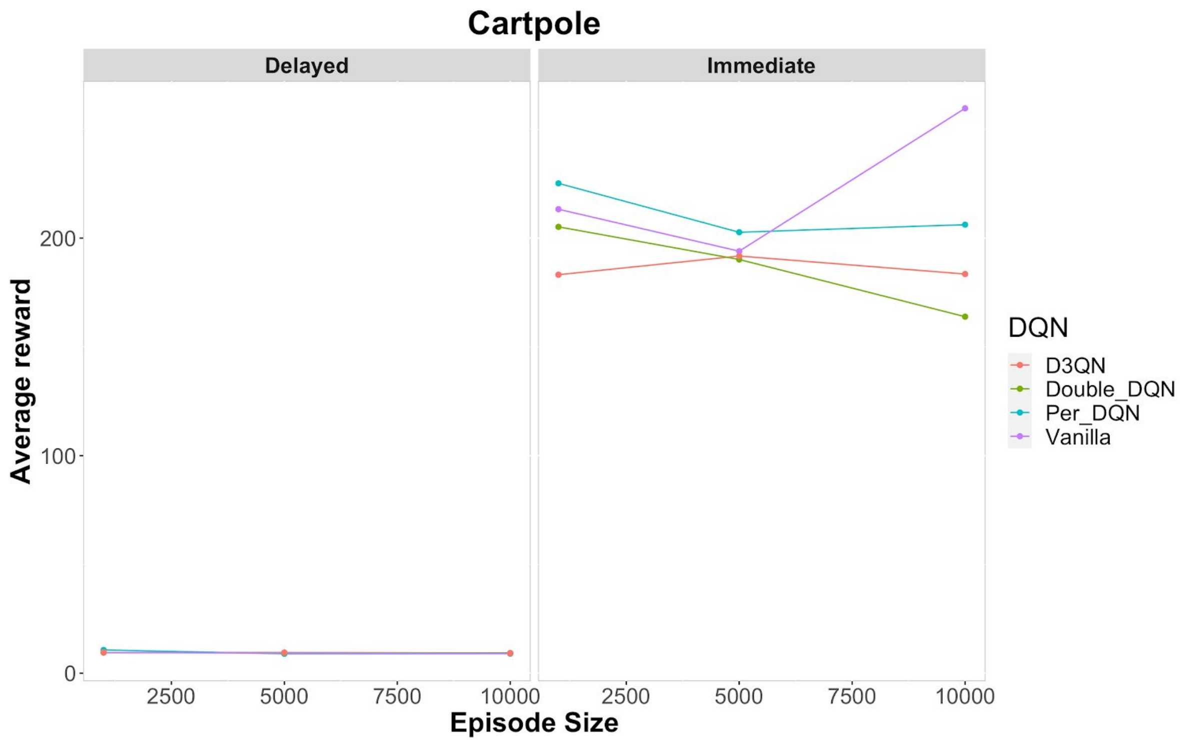


Figure 5: Performance of variants of DQNs with increased episode size. Average reward of the last 10 iterations is shown in Y-axis and episode size in X-axis.

Performance of different DQN variants:
Let us now focus on the performance differences among the DQN variants. Many related papers that work in an offline environment use Vanilla DQN. The reason behind choosing Vanilla DQN over other variants is not covered in those works. We aim to analyze how all other variants perform in an offline environment in order to understand the rationale behind using Vanilla DQN in an offline environment. To understand the differences, we analyze the time to converge and the final reward gained by all the variants in Figure 4. In both deterministic gridworld and cartpole dataset, we can observe that the Vanilla DQN tends to converge faster than all other variants and it also acquires the highest rewards with increased training epoch. However, for non-deterministic dataset, we can see PER_DQN to converge faster and collect more rewards than Vanilla DQN. It would be interesting to investigate if PER_DQN performs consistently better than other variants in the face of nondeterminism or the results are just by chance. One implicit reason behind PER_DQN performing better than other DQNs in non-deterministic environments could be its prioritized sampling method.

Discussion

Our study analyzes the performance of variants of DQN in offline environments. We found that all the variants of DQN perform sufficiently well for offline learning only in presence of immediate reward and in both deterministic and non-deterministic environments. Our results also show that variants of DQNs perform better with increased episode size only when the environment is complex and large enough. They reach an optimum maximum reward and further increasing episode size has no considerable effect on the performance. Finally, in offline environments, Vanilla DQN performs better than other variants in deterministic environments both in terms of time to converge and reward collected which justifies the reason behind picking Vanilla DQN in case of offline environments. However, PER_DQN has shown promising performance over other variants in the non-deterministic gridworld environment.

There are various limitations of our study. We measured all the differences in discrete offline environments only. Continuous state and action space are out of scope of this study. We could not conduct statistical significant tests for differences in performance of all the DQN variants in terms of time to converge and reward collection. All the runs take a considerable amount of time. The multiple runs to do the significance tests using different random seeds are time consuming. We opt to investigate this in future.

Conclusion

In this project, we conducted empirical analysis of the variants of DQNs in an offline environment. Vanilla DQN proved to be better than all others in terms of time to converge and reward collection in presence of immediate reward and in a deterministic environment. We could not draw any conclusion for delayed reward for the environments being too straightforward and also due to limited numbers of runs. In the future, we would focus on delayed reward performance and how that can be increased. There are many other variants of DQNs other than the four DQNs that we have not tested in this paper. Other than Q learning, there are AC3 approaches that have been proved useful for offline learning. We can see how its performance differs with other DQNs in an offline environment. Finally, extending our work to an online learning environment would be another interesting addition to our work.

References

[1] Andrychowicz, OpenAI: Marcin, et al. "Learning dexterous in-hand manipulation." *The International Journal of Robotics Research* 39.1 (2020): 3-20.

[2] Silver, David, et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play." *Science* 362.6419 (2018): 1140-1144.

[3] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." *nature* 518.7540 (2015): 529-533.

[4] Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[5] Ausin, Markel Sanz. "Leveraging Deep Reinforcement Learning for Pedagogical Policy Induction in an Intelligent Tutoring System." *In: Proceedings of the 12th International Conference on Educational Data Mining (EDM 2019),*. 2019.

[6] Hessel, Matteo, et al. "Rainbow: Combining improvements in deep reinforcement learning." *arXiv preprint arXiv:1710.02298*(2017).

[7] Rajeswaran, Aravind, et al. "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations." *arXiv preprint arXiv:1709.10087* (2017).

[8] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." *arXiv preprint arXiv:1509.02971*(2015).

[9] Shen, Shitian, and Min Chi. "Reinforcement Learning: the Sooner the Better, or the Later the Better?." *Proceedings of the 2016 conference on user modeling adaptation and personalization*. 2016.

[10] Liu, Zhuo, et al. "Deep reinforcement learning with its application for lung cancer detection in medical Internet of Things." *Future Generation Computer Systems* 97 (2019): 1-9.

[11] Wang, Ziyu, et al. "Dueling network architectures for deep reinforcement learning." *International conference on machine learning*. 2016.

[12] van Hasselt, Hado P., et al. "Learning values across many orders of magnitude." *Advances in Neural Information Processing Systems*. 2016.

[13] Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. 2015. Prioritized experience replay. In Proc. ofICLR.

[14] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. 2016. OpenAI Gym. *arXiv preprint arXiv:1606.01540*.