A Study on Random and Prioritized Memory Buffer Techniques and Their Performance

Group 59: Navid Hassan Zadeh (261019828)
Elynna Liang (261052562), Nooshin Soheililangroodi (261041920)

Abstract

This project explores the efficiency and performance improvements of the Deep Q-network (DQN) algorithm through the implementation and comparison of Random and Prioritized Experience Replay techniques. We aim to empirically assess the impact of both strategies on algorithm performance in various environments, including a a stochastic and a non-stationary gridworld environment. Overall, our findings indicate that Prioritized Experience Replay (PER) can outperform Random Experience Replay (RER) in certain scenarios, suggesting its potential as a valuable strategy for enhancing the learning efficiency of DQN. As for the role of team members, Navid Hassan Zadeh worked on code implementation, project report, and designing experiments. Nooshin Soheililangroodi contributed to experiments and worked on the report. Elynna Liang contributed to experiments, worked on report and prepared the video representation.

1 Introduction

This project's idea was inspired by the material taught during lectures on deep Q-networks and the concept of experience replay associated to training these networks on batches of transitions. The Deep Q-network (DQN) algorithm is a powerful and fundamental idea in reinforcement learning, and thus it is worthwhile to analyze ways to improve its efficiency and performance. One underlying concept in this algorithm is the use of memory replay buffers to store (and later recount) experiences for training. The idea is that after an interaction with the environment, the agent would be trained not on that immediate interaction but on a batch of transitions sampled from replay memory. Different methods exist for structuring and sampling replay memory. For example, uniformly random sampling or some kind of prioritized sampling. The reasoning behind using a prioritized technique is that, often, some experiences are more important and impact than others, and this prioritization can speed up and improve the learning process. This project aims to study two particular strategies and compare their performance in various environments and settings. The first method is Prioritized Experience Replay (PER) technique inspired by the paper "Prioritized Experience Replay" by Tom Schaul et al. [3] The second method is the Random Experience Replay (RER) which is simply the uniform sampling method. Through three experiments, we examine their performance on CartPole-v1, a stochastic gridworld, and a non-stationary gridworld environment to draw conclusions about their performance on memory buffers with diverse and sometimes unaligning information.

2 Background

our implementation of the PER algorithm maintains a replay memory buffer to store past experiences, each associated with a priority determined by its temporal difference (TD) error. For a PER, sampling of transitions can be done greedily or stochastically. The greedy sampling can have issues such as sensitivity to noise, slow shrinkage of the error due to focus of the algorithm on a narrow subset of experiences, and consequently, over-fitting. However the stochastic version ensures a non-zero probability for all transitions. In this sampling strategy, the probability of a transition i in terms of its priority p_i is $P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$ This results in a sampling strategy between pure greedy and uniformly random where the exponent α determines how influential priorities of the transitions are. ($\alpha = 0$ corresponds to uniform random sampling) Moreover,

to avoid the bias from prioritized sampling, important sampling weights are used in the update step. $w_i = (\frac{1}{N} \cdot \frac{1}{P(i)})^{\beta}$ This introduces the second hyper-parameter β for the algorithm. The pseudocode of this algorithm can be found in the Appendix 7.3 Algorithm 1.

3 Related Work

The idea of prioritization which is used in this project is inspired by the paper "Prioritized Experience Replay" by Tom Schaul et al. [3] This paper introduces prioritized replay as a method to improve the efficiency of learning from experience replay by sampling transitions based on their importance determined via TD difference. Through experiments comparing prioritized replay with uniform experience replay, the paper demonstrates improvements in learning speed and performance across various Atari games. The study identifies two variants of prioritized replay, rank-based and proportional, and evaluates their impact on RL algorithms such as DQN and Double DQN. Results show that prioritized replay leads to better performance. Another closely-related paper which studies the effects of memory size and proposes an adaptive memory size algorithm is "The Effects of Memory Replay in Reinforcement Learning" by Ruishan Liu et al. [2]

4 Methodology

We began by implementing the DQN algorithm from scratch. We used the double implementation of target and main neural networks, which enhances the algorithm's learning stability. However, rather than periodically the weights of the target network, we adopted the soft update approach. We also implemented the Random Experience Replay (RER) algorithm to uniformly sample experiences without considering any other factors. During sampling, experiences are randomly selected from the buffer, which provides a diverse set of training data but potentially misses out on information regarding importance of each transition for learning. To deal with this issue, we implemented the Prioritized Experience Replay Memory algorithm which prioritizes experiences based on their TD errors and assigns a higher importance to those with larger errors. Each transition is associated with a priority value which we stored along with the transition.

We also decided to implement a customized gridworld environment with adjustable non-stationarity and stochasticity in order to perform some of our experiments in this project. This part of our code was based on Gym's tutorial on creating custom environments. [1] In customizing our environment, we introduced a centralized vertical wall (barrier) separating the agent's starting point (top left) from its target location (top right). (Please refer to Figure 4a in Appendix 7.2) This barrier contains two openings, both of which allow the agent to pass through. The top opening offers a shorter route, while the bottom one requires a longer path for the agent to get to the target location. These two openings in the barrier can be arbitrarily opened or closed, which enables us to introduce a sudden change (non-stationarity) to the environment. Additionally, the agent's movements within the environment can be stochastic, implying that there is a small probability of deviation from its intended route. (Figure 4b)

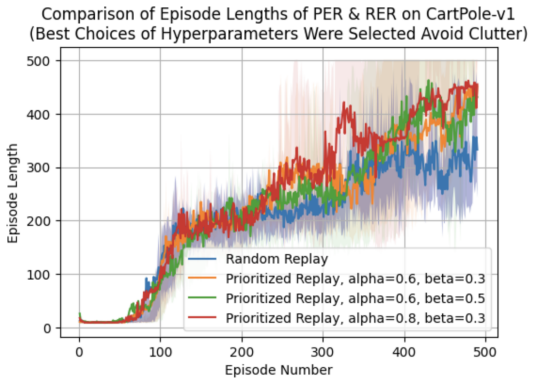


Figure 1: Experiment 1: Tesing on Cartpole-v1

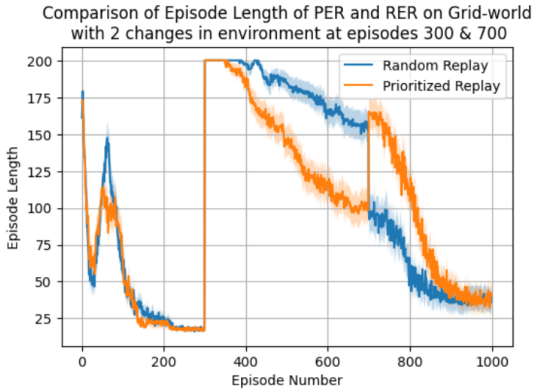
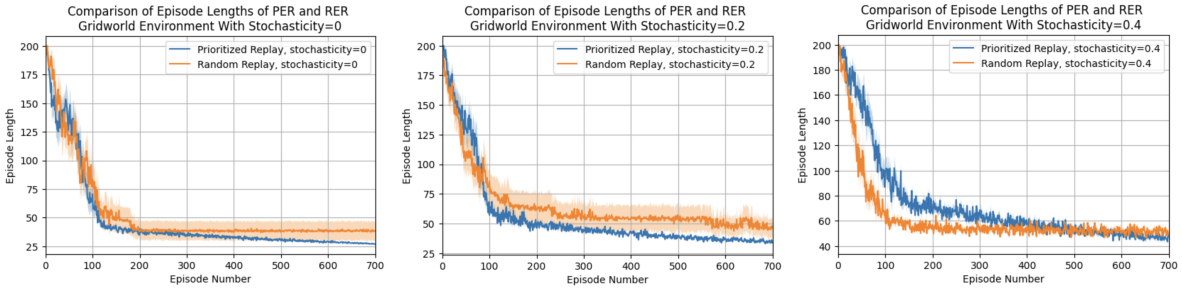


Figure 2: Experiment 3: Non-stationarity at episode 300 and 700.



(a) Stochasticity = 0 (b) Stochasticity = 0.2 (c) Stochasticity = 0.4

Figure 3: Experiment 2 Results

5 Experiments

We conducted three different experiments as follows:

5.1 Experiments 1: Classical Gym environment Cartpole-v1

The first experiment consisted of running the DQN algorithm with both Random and Prioritized Experience Replay on CartPole-v1. Our goal was to evaluate the effectiveness of Prioritized Experienced Replay and understand if there is any advantage in using it over Random Experience Replay in a non-stochastic and stationary environment such as Cartpole-v1. The prioritized algorithm has two important hyper-parameters (namely α and β) which we needed to optimize. We performed ten runs of all possible combinations of $\alpha \in \{0.6, 0.7, 0.8\}$ and $\beta \in \{0.3, 0.5, 0.7\}$. The average value and std error of the three best performing hyper parameters of PER as well as RER are drawn in Figure 1. Due to our limited time and computation power, we couldn't run the configurations more than 10 times. However, with a higher number of runs, the results would be more reliable. (10 runs of each configuration with parallel processing took us more than 18 hrs for this experiment)

We observed based on the results that the performance of both PER and RER were similar in the first 200 episodes. However, PER was able to outperform RER slowly after episode 200. The graph showing all of the tested hyperparameters can be found in appendix 7.5. GRAPH 5. Also, for more details on the average episode length of each configuration of this experiment, please refer to Appendix 7.2 Table 2

5.2 Experiment 2: Stochastic Gridworld environment

In this experiment, we used a stochastic gridworld environment with different levels of stochasticity to evaluate and compare the two memory replay algorithms within such a setting. This environment has a vertical wall separating the agent and its target, and there is an opening in this wall that the agent needs to learn to go through. The stochasticity is introduced in the following way: Suppose the agent takes the 'right' or 'left' actions. Sometimes, it could end up in the upper or lower blocks with a certain (often small) probability p which we call stochasticity value. But often (with probability $1 - p$) it will go to its intended direction. Similarly, this is implemented for 'up', 'down' movements. For more details on this and an image of the environment please refer to appendix 7.4 Figure 4b. We ran each algorithm 20 times for three different values of stochasticity, namely 0, 0.2, and 0.4. The results of this experiment are drawn in Figure 3 for each of the stochasticity values. For stochasticity=0.4, the results show that in the first 600 episodes RER does better than PER. However, after episode 600, PER manages to catch up and slightly outperform RER. For stochasticity=0.2, RER does better for approximately the first 90 episodes, while during the rest of the training, PER performs significantly better. In the case of no stochasticity, the PER outperforms RER after episode 100. We hypothesize based on these graphs that the higher stochasticity values result in a better performance of Random Experience Replay at the beginning. However, PER outperforms RER eventually given adequate time. This hypothesis can be explained by the nature of prioritized replay in being more prone to rare transitions and giving them a higher weight since it operates based on temporal difference. However, this is only a hypothesis for now and we would require more extensive experiments to confirm it, which we are unable to do within the scope of this final project.

5.3 Experiment 3: Non-stationary Gridworld environment

Using a customized non-satationary gridworld environment 4a, we constructed an experiment to investigate the performance of these two algorithms on a non-stationary environment. By non-stationarity, we mean two small but consequential changes in the layout of the environment during the training such that the agent needs to re-learn its path to target. These changes are as follows: initially, the top opening is open and the bottom opening is close. First change (occurring at episode 300) involves switching both doors such that the agent is forced to take the path through the bottom door. Second change (occurring at episode 700) switches both doors again, making it the same as the starting state. Our goal was to observe how fast each replay algorithm enables DQN to adapt to the each change and discover the new route to target location. We used infinite memory in this experiment.[1] We performed 50 runs of each algorithm. The results are drawn in Figure 2. Also, for more detailed information about the average episode length of each algorithm at different points during training please refer to Table 4 in Appendix 7.2. An interesting observation from the results is that after the first change, the random replay method takes longer than prioritized replay to recover. The second change, reverting the layout to the initial layout, occurs while random replay has less effectively learned the new path compared to prioritized replay. This could explain why random replay starts out performing better than prioritized replay after episode 700. However, eventually prioritized replay catches up to Random Replay in performance.

6 Conclusion

In conclusion, this project compared Prioritized Replay (PER) with Random Experience Replay (RER) in the Deep Q-network (DQN) algorithm across various environments. While both initially performed similarly in Experiment 1 (CartPole), PER eventually outperformed RER with optimized parameters. In Experiment 2, RER showed initial superiority in high-stochasticity settings, but PER adapted better over time. In Experiment 3, PER showed faster recovery after the first environmental shift, but a slower one in the second change. Overall, while both PER and RER have their merits, our experiments suggest that PER can offer certain advantages or disadvantages based on the dynamics of the environment.

6.1 Future Work

As mentioned in Experiment 2, doing more tests to confirm our hypothesis would be a good extension of this project. Also, since in this project we did not modify the memory capacity, it would be interesting to investigate different methods of adaptive memory buffer capacity and also methods of lowering the size of the memory such as cutting it by half. Additionally, experimenting with other memory replay methods could be another way to extend this project in future. Exploring hybrid approaches could also potentially be beneficial in non-stationary environments since looking at experiment 3, the performance of each techniques were favourable in a specific setting.

References

[1] Gym Documentation. *Make your own custom environment*. `https://www.gymlibrary.dev/content/environment_creation/#creating-a-package`. Accessed: April 16, 2024.

[2] Ruishan Liu and James Zou. *The Effects of Memory Replay in Reinforcement Learning*. 2017. arXiv: `1710.06574 [cs.AI]`.

[3] Tom Schaul et al. *Prioritized Experience Replay*. 2016. arXiv: `1511.05952 [cs.LG]`.

[1] This was done by assigning a capacity to memory buffer larger than the maximum number of times a transition can be added to it given the experiment at hand.

7 Appendix

7.1 Parameter settings of the Experiments

Table 1: Parameter settings of the Experiments

Environment	Experiment 1: CartPole-v1		Experiment 2: Stochastic Gridworld		Experiment 3: Non-stationary Gridworld	
Memory Buffer Type	Random	Prioritized	Random	Prioritized	Random	Prioritized
# of runs (averaged over)	10	10	20	20	50	50
Learning rate	5e-4	5e-4	5e-4	5e-4	5e-4	5e-4
Gamma	0.99	0.99	0.99	0.99	0.99	0.99
Episode number	490	490	700	700	1000	1000
Tau	0.01	0.01	0.01	0.01	0.01	0.01
Training batch size	64	64	64	64	64	64
Epsilon max (starting)	0.1	0.1	0.1	0.1	0.4	0.4
Epsilon min	0.02	0.02	0.02	0.02	0.2	0.2
Stochasticity	-	-	Hyperparameter: 0, 0.2, 0.4	Hyperparameter: 0, 0.2, 0.4	0	0
Memory buffer size	50,000	50,000	50,000	50,000	200,000	200,000
Memory buffer alpha	-	Hyperparameter: 0.6, 0.7, 0.8	-	0.6	-	0.6
Memory buffer beta	-	Hyperparameter: 0.3, 0.5, 0.7	-	0.5	-	0.5

7.2 More detailed Results

Table 2: Results of Experiments 1: Average length of last 10 episodes

Alg.	Avg. length of last 10 episodes
RER	323.35
PER, $\alpha = 0.6$ $\beta = 0.3$	**429.04**
PER, $\alpha = 0.6$ $\beta = 0.5$	**420.19**
PER, $\alpha = 0.6$ $\beta = 0.7$	222.25
PER, $\alpha = 0.7$ $\beta = 0.3$	342.04
PER, $\alpha = 0.7$ $\beta = 0.5$	350.04
PER, $\alpha = 0.7$ $\beta = 0.7$	297.9
PER, $\alpha = 0.8$ $\beta = 0.3$	**446.88**
PER, $\alpha = 0.8$ $\beta = 0.5$	303.37
PER, $\alpha = 0.8$ $\beta = 0.7$	277.25

Table 3: Results of Experiments 2: Average length of last 10 episodes

Stochasticity	Alg.	Avg. length of last 10 episodes
Stochasticity=0	RER	38.42
	PER	**27.24**
Stochasticity=0.2	RER	45.94
	PER	**34.64**
Stochasticity=0.4	RER	51.39
	PER	**46.16**

Table 4: Results of Experiments 3: Average length of last 10 episodes at different times during the training

Alg.	Average length of 10 last episode at:					
	ep=200	ep=299	ep=600	ep=699	ep=800	ep=999
RER	24.7	17.97	166.61	156.27	**62.38**	**37.37**
PER	**22.27**	**17.67**	**115.77**	**101.11**	115.73	38.99

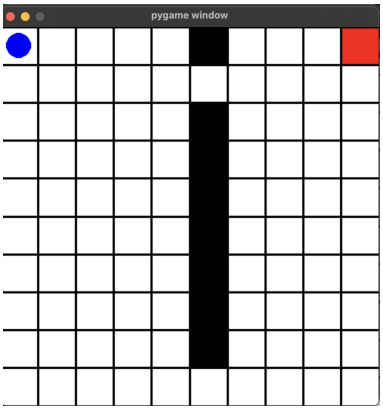
7.3 Pseudo Code of the Prioritized Replay

Algorithm 1 Prioritized Experience Replay

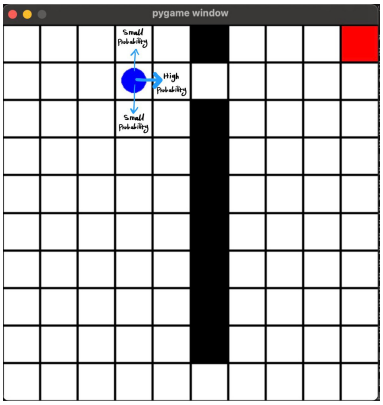
Require: minibatch k, step-size η, replay period K and size N, exponents α and β, budget T.
Ensure: Updated weights θ, and target network weights θ_{target}

Initialize replay memory $\mathscr{H} = \emptyset$, $\Delta = 0$, $p_1 = 1$
Observe S_0 and choose $A_0 \sim \pi_\theta(S_0)$
for $t = 1$ **to** T **do**
    Observe S_t, R_t, γ_t
    Store transition $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$ in $\mathscr{H}$ with maximal priority $p_t = \max_{i<t} p_i$
    if $t \equiv 0 \mod K$ **then**
        for $j = 1$ **to** k **do**
            Sample transitions and calculate $P(j) = p_j^\alpha / \sum_i p_i^\alpha$
            Compute importance-sampling weight $w_j = \left(N \cdot P(j)\right)^{-\beta} / \max_i w_i$
            Compute TD-error $\delta_j = R_j + \gamma_j Q_{\text{target}}\left(S_j, \arg\max_a Q(S_j, a)\right) - Q(S_{j-1}, A_{j-1})$
            Update transition priority $p_j \leftarrow |\delta_j|$
            Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
        end for
        Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$, reset $\Delta = 0$
        perform a soft update of the target network θ_{target} based on main network θ
    end if
    Choose action $A_t \sim \pi_\theta(S_t)$
end for

7.4 More Details on The Customized Gridworld Environment We Tailored for This Project



(a) The two openings are both open in the picture above. During our experiments, we can arbitrarily close either of them.



(b) This image shows the stochasitcity. Suppose the agent wants to go right. With high probability it goes to right cell. But with a small probability it ends up in top or bottom cells.

Figure 4: Images of The Gridworld Environment Customized For The Project

7.5 Additional Graphs From The Experiments

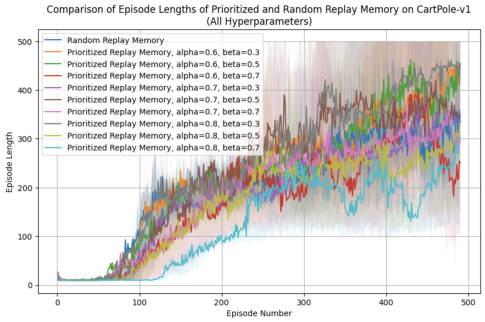


Figure 5: Experiment 1: Graph of all Hyperparameters

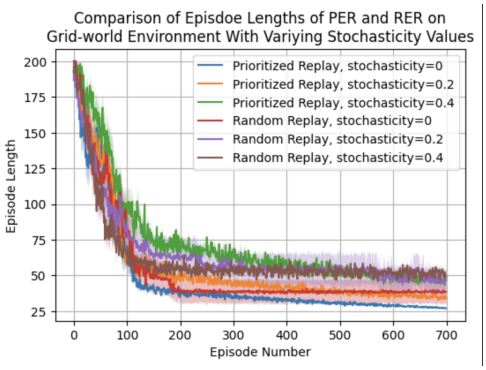


Figure 6: Experiment 2: Graph of all Hyperparameters