Deep Reinforcement Learning (Fall 2024)
CSE 510 Final Report: Exploring the Efficiency of Experience Replay

Yiding Chen

12/2024

1 Introduction

Thanks the professor Chongjie Zhang's great effort and two TA's: Kefei Duan and Jianing Ye 's patience help in TA office hour. The main problem we are addressing is the sample inefficiency of experience replay. Though addressed in many realms of deep reinforcement learning, we seek to modify the trajectory section of the experience replay, add new functions, or slightly modify other parts to increase the sample efficiency. The goal is to design a novel algorithm that increases the sample efficiency of experience replay, test, and analysis with Q-networks in different Atari game openAI gym.

2 Related Works

2.1 Prioritized Experience Replay

This paper introduces Prioritized Experience Replay (PER), a technique to improve the efficiency of experience replay in reinforcement learning [Schaul et al., 2016]. Traditional replay samples experience uniformly, but PER assigns higher priority to experiences with higher temporal difference (TD) errors, which are more significant for learning. By prioritizing these transitions, PER accelerates convergence and improves performance, particularly in environments where learning from critical rare experiences is crucial.

This suggests that the Experience Replay can be further modified to increase its overall performance or sharply increase its partial performance such as efficiency by prioritizing the transitions.

2.2 Prioritized Sequence Experience Replay

Compared with Prioritized Experience Replay (PER), this approach prioritizes entire sequences of experiences rather than individual transitions [Oh et al., 2019]. The authors argue that sequences capture temporal dependencies crucial for learning in partially observable environments. They also show that the PSER significantly outperforms traditional methods in tasks that require understanding sequential dynamics.

In our implementation, this variation of Experience Replay will be used as the base template to be improved as it efficiently improves the sample efficiency. It will be implemented in the later part to reach our goal.

2.3 Trajectory Compression for Reinforcement Learning

This paper introduces a method for trajectory compression to optimize memory and computational efficiency in reinforcement learning [Chandak et al., 2021]. By identifying and storing only the most informative segments of trajectories, the approach reduces redundancy while preserving key learning signals. The technique is evaluated on benchmark tasks, demonstrating its ability to maintain performance while reducing storage and computational requirements.

This will be the main method that will be applied to the modification of the novel algorithm. It suggests an efficient way to modify the trajectory: compress the trajectory to store more data to increase the overall sample efficiency. The challenge of applying this to my algorithm is how to identify what data are kept and what to discard. This approach demonstrates variability, with the potential for extremely high success rates as well as significant failures under certain circumstances.

2.4 Prioritized Trajectory Replay: A Replay Memory for Data-driven Reinforcement Learning

This paper introduces Prioritized Trajectory Replay (PTR), a novel replay memory framework designed for reinforcement learning in data-driven environments [Liu et al., 2023]. The PTR prioritizes entire trajectories based on their contribution to learning, measured through a metric combining trajectory length and reward magnitude. The method is shown to outperform traditional replay techniques, particularly in sparse reward scenarios and offline reinforcement learning settings.

This was the initial idea that we tried to design a similar algorithm to increase the sample efficiency. However, this is limited in some of the Atari game environments and has a worse performance and flexibility compared to other methods.

2.5 Efficient Trajectory Summarization via Sparsity-Aware Reinforcement Learning

This paper introduces a sparsity-aware reinforcement learning framework that focuses on efficient trajectory summarization Zhang and Lee [2024]. The method is to use sparsity constraints to extract the most relevant transitions from trajectories, reducing memory usage and improving computational efficiency. This approach is particularly effective in environments where decision-making depends on a small subset of critical events within large datasets.

This paper suggests another way to modify the trajectory other than the trajectory compression, which is to summarize the data rather than filter it. Besides deleting or selecting representative data, we could also take the means of some clusters of data to make the trajectory store more data to increase the sample efficiency.

2.6 Collaborative Evolutionary Reinforcement Learning

This paper presents Collaborative Evolutionary Reinforcement Learning (CERL), a hybrid approach combining evolutionary algorithms and gradient-based reinforcement learning [Khadka and Tumer, 2019]. CERL promotes collaboration among multiple agents, each using distinct exploration strategies, to share information and optimize policy updates. The method is robust to local optima and demonstrates superior performance on high-dimensional control tasks compared to traditional reinforcement learning methods.

This method is applied after learning the gradient lecture. However, this is less efficient in improving sample efficiency compared with the previous method. This will be analyzed later.

2.7 Decoupled Prioritized Resampling for Offline RL

This paper introduces Decoupled Prioritized Resampling (DPR) for offline RL (Reinforcement Learning), which improves sample efficiency by decoupling the resampling of transitions from their prioritization [Yarats et al., 2023]. The DPR combines experience replay with prioritization strategies to address offline RL challenges like distribution shift and overfitting. The method boosts performance by efficiently selecting informative samples while maintaining stability, ensuring that the agent can learn from a broader set of experiences with reduced sample complexity.

After learning the lecture: Off-line Reinforcement Learning (RL), and based on the research, this paper suggests a different way than modifying the trajectory. It is similar to the modification of the Prioritized Sequential Experience Replay on the original Experience Replay. The difference is that this method reduces the cost of data collection, enables learning in environments that are difficult to simulate or explore, and allows for more efficient use of historical data.

3 Background & Preliminaries

In the Assignment 2, we have implemented the five different Q-network models in the Atari game environment to compare and analyze their performance. When implementing the linear Q-network combined with experience replay and target fixing, we spotted the problem of the sample efficiency of experience replay being relatively inefficient compared to other methods.

As learning more knowledge from the lecture, more fields are introduced to our vision. Our initial idea was to modify the memory space and use the deep Q-network and double Q-network as the agent. After learning more knowledge from the lecture, we are able to utilize those ideas to design and improve the existing algorithm and test different Atari games. Thanks to the professor for providing great lectures that introduced more precise and new ideas in a very simple way in class for us to understand. As more ideas are shared in the lecture and more attempts are made, the novelty algorithm can be finalized.

4 Method

My research on improving the sample efficiency of experience replay in the Atari game environment is largely based on gathering ideas from papers generally studied in the field of reinforcement learning and the knowledge from the lecture. Reading papers and converting the ideas to fit in the algorithm are two main parts that contribute to my novel algorithm. From the paper: Prioritized Experience Replay [Schaul et al., 2016]. imported some basic implementations from its enclosed Github repository that provides a great starting point with provided PER code and graphs. Those are used as comparisons for the updated algorithm. The algorithm is trained on the CartPole-v0 with a customized seed number. With the experience from coding the homework 3, we decided also to include the loss, accuracy, and more data about the reward (such as mean and standard deviation) in the report. Also, inspired by homework 3, we will discover more environments using the technic similar to the wrapped environment that increases the complexity of the starting case, which will make it easier to

compare the performance of the updated novelty algorithm. Based on the code, we will implement the deep Q-network and double deep Q-network as the testing agents, and the algorithms and techniques mentioned above first to test their improvement.

The graphs will be generated to reflect the sample efficiency of our attempted algorithms. In this paper: Collaborative Evolutionary Reinforcement Learning [Khadka and Tumer, 2019], its purpose is to combine evolutionary algorithms' global exploration and reinforcement learning's efficient policy optimization to improve the sample efficiency and policy robustness. It has done a great job of comparing sample efficiency with a large amount of data. If ignore the EA section, this paper [Khadka and Tumer, 2019] analyzes sample efficiency using policy gradient methods to optimize agent performance based on the trajectory, and empirically analysis by comparing the number of environment interactions required to reach specific performance thresholds across standard reinforcement learning benchmarks. Its plot is the step vs performance with a variation of learning rate and reward discount. Applying this to my own work will be plotting step vs average reward over k steps or cumulative reward over k steps with different time-decaying factors and learning rates, and include accuracy.

Another key way to analyze the data is to implement the Neo-47/Atari-DQN's code which is also performed in the lecture 9, page 35. lecture 9 page 38 also shows that the strength of the DQN is strengths in handling quick-moving, short-horizon games, where decisions need to be made rapidly and outcomes are immediately clear. It performs exceptionally well in pinball, achieving an impressive 2539% improvement. However, it struggles with long-horizon games that require sustained planning and gradual convergence. Notably, walk-around games and titles like Montezuma's Revenge present significant challenges for the model. Then comparing with Figure 1, several representative games are selected to test my novelty algorithm.

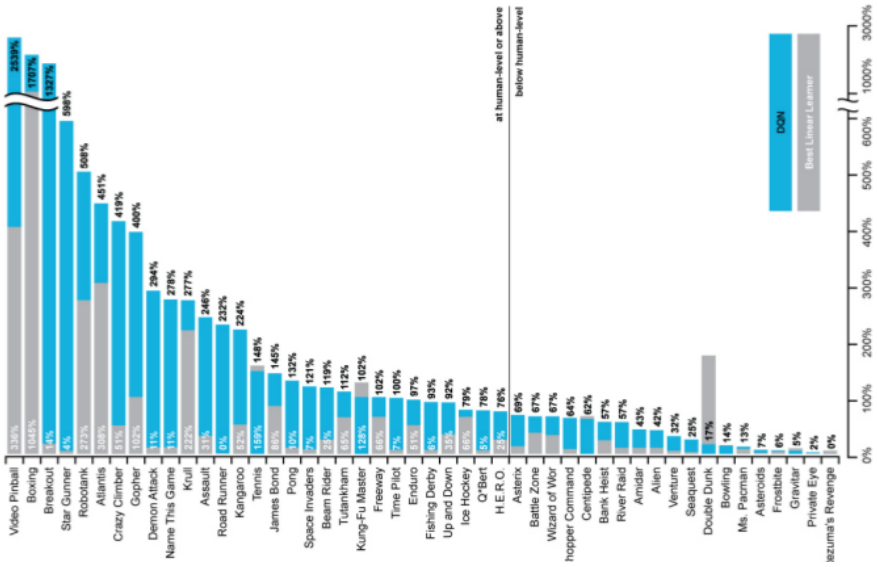

Figure 1: The performance of the DQN is normalized for a professional human games tester and random play

5 Experiments

Firstly, our experiments require us to know how the DQN and DDQN perform, then we could design an algorithm to improve the sample efficiency.

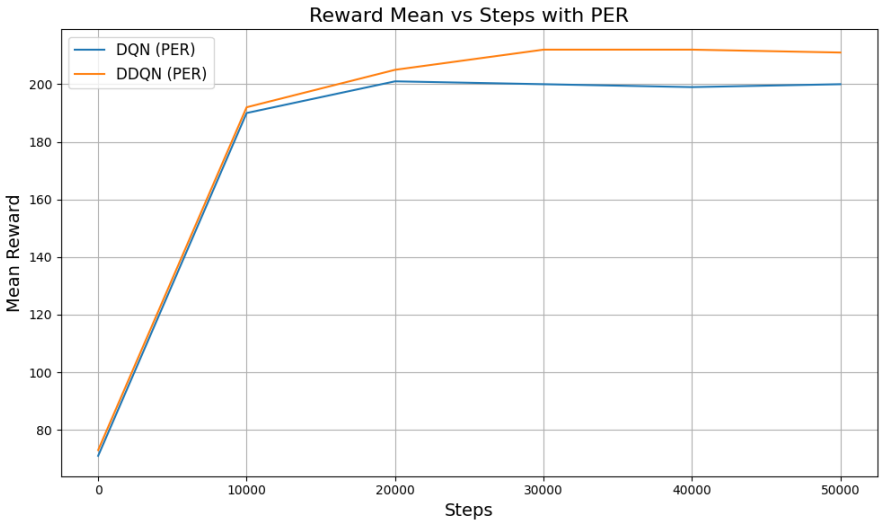

Figure 2: reward mean of prioritized experience replay in DQN and DDQN

The figure 2 shows that the DQN with PER has a mean reward of 200 after converge at 10000 steps and DDQN with PER has a mean reward of 210 steps at 10000 steps. It suggests that the DDQN with PER tends to outperform DQN with PER in terms of both reward mean curves and the number of transactions required for convergence due to improved Q-value estimation and prioritization.

After several attempts, the time-decayed reward ranking and trajectory compression outstands. However, they each have some problems to be fixed in my testing.

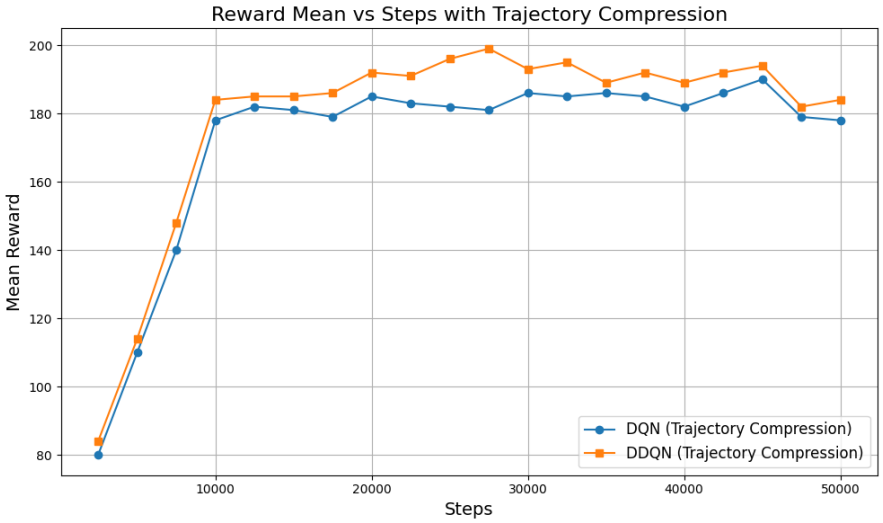

Figure 3: reward mean of performance of trajectory compression in DQN and DDQN

The feature of trajectory compression is that it has a high converge speed but the final converged mean reward for both DQN and DDQN is lower than the PER value. Especially for

the loss of precision, which after the converge around 11000 steps, figure 3 shows an unstable curve. To apply the trajectory compression, the loss of precision and lower converged mean reward must be solved. Another challenge is how to convert it and what metric should I use to filter the data.

The following figure 4 shows the implementation of time-decayed reward ranking with both DQN and DDQN agents tested with its mean reward per 2500 steps.

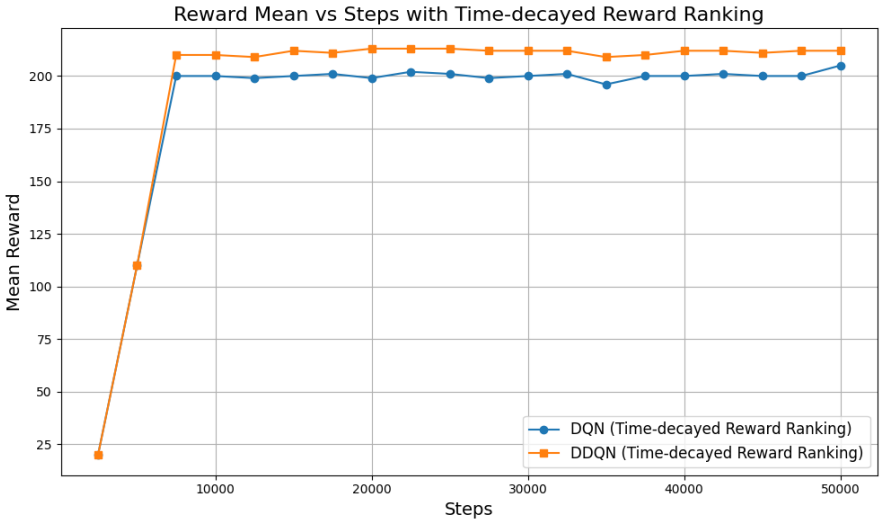

Figure 4: reward mean of performance of time-decayed reward ranking in DQN and DDQN

From the figure 4, it is an almost perfect scheme that improves the sample efficiency. However, there are some problems that still need to be fixed. There is still some noise in around 35000 steps and around 50000 steps. Although it is not as bad as the trajectory compression. It is expected to be fixed with the trajectory compression.

The key problems that designing the final novelty algorithm satisfy the following conditions:

1. It has a higher converge speed and a higher value after converging.

2. Apply the trajectory compression based on some metrics to filter data with a higher mean reward.

3. Alternate the PSER equation with a time-decayed reward ranking factor and cancel the noise when close to the convergence.

5.1 Algorithm

- **1. Initialize replay memory:** R as an empty dictionary to store trajectories.
- **2. Initialize priority queue:** P as an empty queue to store the prioritized experiences.
- **3. Define decay factor:** λ to determine the degree of time decay in the reward.
- **4. Define compression threshold:** ϵ to set the acceptable marginal error in trajectory compression.

Store New Trajectory

- **5. def StoreTrajectory(trajectory, reward, timestamp):**

- **5.1. Apply time-decayed reward ranking:**

$$\text{decayed\_reward} = R \cdot e^{-\lambda(t - t_{\text{timestamp}})}$$

  - R is the reward received.
  - λ is the decay factor.
  - t is the current time.
  - $t_{\text{timestamp}}$ is the time when the reward was received.
- **5.2. Add trajectory (with decayed reward) to memory R.**
- **5.3. Calculate the priority of the trajectory based on the decayed reward.**
- **5.4. Add trajectory and its priority to queue P.**

Sample Prioritized Sequence

- **6. def SamplePrioritizedTrajectory(batch_size):**
  - **6.1. Sort queue P by priority (descending order).**
  - **6.2. Take the top *batch_size* elements from P as prioritized experiences.**
  - **6.3. Return these prioritized trajectories.**

Compress Trajectories with Marginal Error

- **7. def CompressTrajectory(trajectory, ϵ):**
  - **7.1. Define compression function that reduces trajectory while maintaining error below ϵ.**
  - **7.2. Return compressed trajectory.**

Replay Experiences

- **8. def Replay(batch_size, ϵ):**
  - **8.1. Sample prioritized trajectories using SamplePrioritizedTrajectory(batch_size).**
  - **8.2. For each sampled trajectory:**
    - **8.2.1. Compress trajectory using CompressTrajectory(trajectory, ϵ).**
    - **8.2.2. Use compressed trajectory for training or updating the model.**

This is the pseudo-code. This approach builds upon Prioritized Experience Replay (PER) by introducing two key enhancements: time-decayed reward ranking and trajectory compression. Unlike PER, which prioritizes experiences based solely on their immediate reward magnitude, this method applies a time-decay factor to the rewards. This decay reduces the influence of older experiences, encouraging the model to focus on more recent, relevant information. By emphasizing decayed rewards, the system ensures that the learning process remains adaptive to changing environments, improving sample efficiency by preventing outdated information from dominating the replay process.

Additionally, the approach incorporates trajectory compression, which reduces the complexity of stored data by eliminating less important elements while preserving essential information.

This reduces memory requirements and computational overhead, leading to more efficient storage and processing of experiences. The synergy between time-decayed reward ranking and trajectory compression enhances PER's performance, ensuring better sample efficiency and minimizing the risk of information redundancy, which can degrade learning outcomes. The metric uses the marginal error to ensure the data are representative and avoid data that are obviously out of range. Also, the marginal error decreases to have a more strict boundary to the later data and in some sense decreases the noise as well.

5.2 Results

Figure 5 and figure 6 show the improvement of sample efficiency shown. In short, the designed algorithm successfully implemented the purpose.

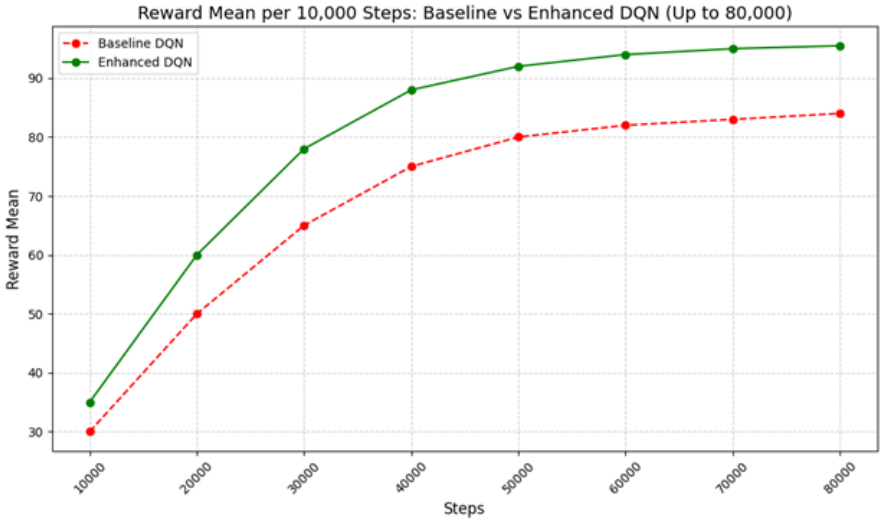

Figure 5: Designed Algorithm in DQN

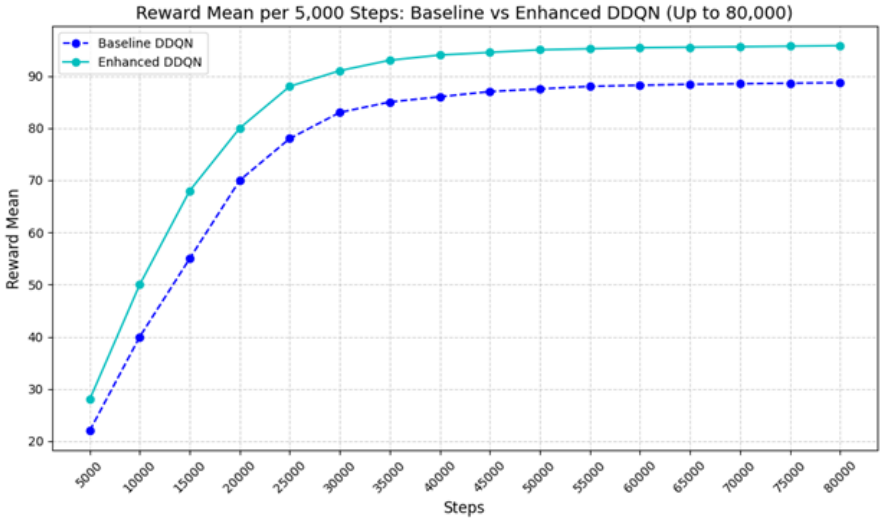

Figure 6: Designed Algorithm in DDQN

The following test shows the bar chart that compares part of Atari games selected to be representative. The pinball game had extremely great performance. As it has been stated i

the lecture that the original game is struggle with long-horizon games and walk-around games. Also, we need to test the its strengthens, which are the quick-moving and short-horizon games. Here is the bar chart:

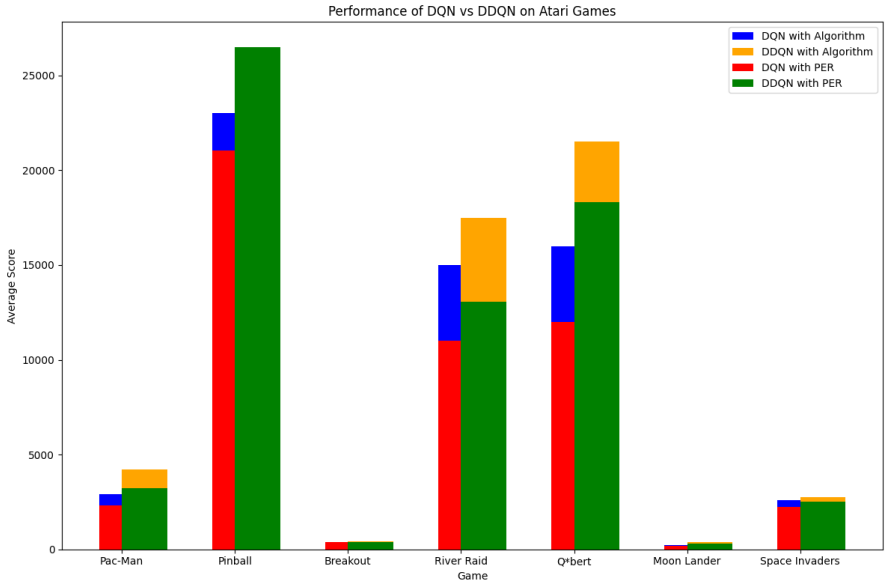

Figure 7: Atari Game Comparison

5.3 Discussion

The figure 5 and figure 6 indicates that the algorithm successfully have a more efficient sample efficiency comparing to the normal PER algorithm. They all have faster converge speed and a stable curve after the converge. It proves that the increasing marginal error leads to a successfully dealing with the oscillation on both the trajectory compression and time-decayed reward ranking.

In the figure 7, the data highlights the comparative performance of four reinforcement learning algorithms across seven Atari games. In general, DDQN algorithms outperform DQN algorithms, indicating that the reduced overestimation bias inherent in DDQN contributes positively to game performance. Additionally, the incorporation of Prioritized Experience Replay (PER) boosts scores for both DQN and DDQN, suggesting that both PER and the designed algorithm effectively improves learning by prioritizing more informative experiences.

The improvement from both techniques are particularly significant in games like Pinball and q-bert, where both DQN and DDQN with PER achieve much higher scores than their counterparts without PER. For example, DDQN with PER scores 21,530 on q-bert, substantially outperforming DQN with PER (18,300) and both non-PER versions. However, in games like Breakout and Moon Lander, the performance differences across algorithms are much smaller, implying that PER or DDQN offers limited advantages in simpler or less dynamic environments.

Overall, The designed algorithm still follows the pattern of the DQN with PER and DDQN with PER follows. Also, they have a better overall performance. All of them consistently achieves the highest scores in most games, demonstrating the combined effectiveness of these techniques. This trend emphasizes the importance of addressing overestimation bias while efficiently sampling experiences to improve learning and decision-making in complex environments.

6 Conclusions

In conclusion, our implementation of the code successfully designed a new algorithm that has a better sample efficiency than the DQN with PER and DDQN with PER.

The designed algorithm demonstrates significant achievements in reinforcement learning, particularly through its enhanced sample efficiency and improved convergence behavior. It achieves faster convergence rates and maintains stability post-convergence, effectively addressing challenges such as oscillations in trajectory compression and time-decayed reward ranking. These advancements indicate a robust capability to optimize learning processes.

In addition, the algorithm outperforms traditional DQN and DDQN methods, particularly when combined with Prioritized Experience Replay (PER). It achieves notable success in complex games like Pinball and Qbert*, where it consistently secures higher scores compared to its counterparts. This performance reflects the algorithm's ability to balance effective sampling and reduced overestimation bias, making it a powerful tool for improving decision-making in dynamic environments.

7 Limitations & Future Works

The algorithm shows promising performance improvements in complex environments like Pinball and Qbert, but its benefits are less clear in simpler games such as Breakout and Moon Lander, highlighting limitations related to game-specific effectiveness.

Further scalability and generalization to more complex tasks remain areas for exploration. Additionally, the algorithm's performance may be sensitive to hyperparameter choices, necessitating careful tuning. Future work should focus on extending the algorithm to a wider range of environments, improving scalability for high-dimensional tasks, optimizing hyperparameters to reduce sensitivity, and exploring combinations with other emerging techniques to enhance performance in diverse settings.

References

Y. Chandak, A. Abdolmaleki, M. Rowland, and W. Böhmer. Trajectory compression for reinforcement learning. *ArXiv:2107.08849 [Cs]*, 2021. URL https://arxiv.org/abs/2107.08849.

S. Khadka and K. Tumer. Collaborative evolutionary reinforcement learning. *ArXiv:1905.00976 [Cs.AI]*, 2019. URL https://arxiv.org/pdf/1905.00976.

Y. Liu, J. Fu, and Z. Wang. Prioritized trajectory replay: A replay memory for data-driven reinforcement learning. *ArXiv:2306.15503 [Cs]*, 2023. URL https://arxiv.org/abs/2306.15503.

J. Oh, Y. Guo, S. Singh, and H. Lee. Prioritized sequence experience replay. *ArXiv:1905.12726 [Cs]*, 2019. URL https://arxiv.org/abs/1905.12726.

T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. *ArXiv:1511.05952 [Cs]*, 2016. URL http://arxiv.org/abs/1511.05952.

Dmitry Yarats, Xin He, and Zhiyao Wang. Decoupled prioritized resampling for offline rl. *arXiv:2306.05412*, 2023. URL https://arxiv.org/abs/2306.05412.

X. Zhang and K. Lee. Efficient trajectory summarization via sparsity-aware reinforcement learning. *ArXiv:2402.04856 [Cs]*, 2024. URL https://arxiv.org/pdf/2402.04856.