AI 832: Project Goal Statement
A Consolidated Framework for Sampling from Experience Replay Buffer

Krutik Patel
IMT2021024

Ricky Ratnani
IMT2021030

Sai Madhavan G
IMT2021101

I. Problem Statement

Experience replay is a replay memory technique used in reinforcement learning where we store the agent's experiences at each time-step, $e_t = (s_t, a_t, r_t, s_{t+1})$ in a data-set $D = e_1, \cdots, e_N$, pooled over many episodes into a replay memory. Mini-batches of experiences are then sampled from this replay buffer to be applied on off-policy training algorithms. The experience replay mechanism helps in avoiding two key issues that could be caused by training on experiences in a sequential manner, namely, the updates being strongly correlated and thus breaking the i.i.d. assumption of stochastic gradient descent and rare experiences being forgotten rapidly.

Prioritizing few experiences while, so as to replay important transitions more frequently and learn more efficiently, has been proved to significantly improve performance when compared to uniform sampling as seen in [5]. The authors used the temporal difference error (TDE) computed while training their DQN agent, as a basis to sample experiences with a high TDE more often. Following their footsteps, we aim to explore what other metrics could be used for proportional sampling, which would enable important experiences to be sampled more often. We do this by systematically testing our proposed metrics on a variety of environments trained on a double DQN agent.

A. Definitions

- An experience or a transition (e_t) is a tuple containing the state, action, reward and next state ((s_t, a_t, r_t, s_{t+1}) obtained as a part of an agent's trajectory, at time t.

$$e_t = (s_t, a_t, r_t, s_{t+1})$$

- A chunk C is a collection of contiguous transitions of a fixed length.

$$C = \{e_t, e_{t+1}, \ldots, e_{t+k}\}$$

.

- The replay buffer B, is a collection of chunks consisting of transitions procured form trajectories of an agent over the training process. It is of a fixed length N.

$$B = \{C_1, C_2, \ldots, C_N\}$$

- The weight assigning function (f) maps a chunk to its weight which is a real number.

$$f(C_i) = w_i$$

- The weight distribution of a replay buffer ($W(B)$) is the collection of weights corresponding to the chunks in the buffer normalized such that their sum is 1.

$$W(B) = \{p_1, p_2, \ldots, p_N\}$$

where, $p_i = \frac{f(C_i)}{\Sigma_{j=1}^{N} f(C_j)}$

- The DQN agent ($Q(S, A)$) is a feed-forward neural network that, when trained till convergence, gives the value corresponding to the given state and action.
- A mini-batch or batch is a set of chunks that is sampled from B based on the distribution $W(B)$. This batch is used for the gradient update while training Q

B. Objectives

- Devise a flexible framework for f such that, for any given environment, f can be changed to the optimal function which minimizes the number of step updates required for the agent to reach convergence.
- Experiment and find overarching patterns on the configurations of f that work best for particular environments.

II. Literature survey

In recent years, the development of reinforcement learning (RL) techniques has led to a growing focus on optimizing the use of experience replay buffers. Various methods have been proposed to address different aspects of this challenge, aiming to improve learning efficiency and stability in dynamic environments.

Adjusting Replay Buffer Parameters: Zhang and Sutton (2017) [1] emphasize the significance of adjusting parameters such as the size of the memory buffer. They found that a very large replay buffer can significantly impair performance, as many transitions are rarely sampled. To mitigate this issue, they proposed strategies like Combined Experience Replay (CER), where alongside sampled transitions, some of the most recent transitions are included in the batch.

Improving Efficiency of Experience Replay: Methods like Experience Replay Optimization (ERO), Synthetic

Experience Replay (SYNTHER), and the ALAP algorithm have been proposed to optimize the selection of experiences or generate synthetic data to enhance learning performance, sample efficiency, and stability in RL tasks. [2]–[4]

ERO attempts to learn how to sample experiences from the replay buffer itself by constructing a new Markov Decision Process (MDP) and a reward function for each batch sampled from the replay buffer, and then training the sampling policy accordingly.

Prioritizing and Optimizing Experience Replay: Prioritizing and optimizing the replay of experiences is another key area of research. Methods such as Prioritized Experience Replay (PER) aim to accelerate learning speed and improve performance by strategically selecting and replaying experiences based on their significance or temporal-difference error. [5], [6].

The basic intuition behind prioritized experience replay is to improve the efficiency of learning in reinforcement learning algorithms by focusing more on experiences that are likely to lead to significant learning updates. Additionally, importance sampling is often used to mitigate the bias introduced by prioritized sampling. However, a major challenge remains in optimizing the sampling of transitions from the replay buffer. Some transitions may be rarely sampled initially due to low temporal difference error, but they could become important later during training.

Future Directions: A major challenge is optimizing the sampling of transitions from the replay buffer. Our goal is to refine this sampling process to ensure that experiences are effectively and thoroughly sampled. We believe that more attention should be directed towards various parameters of the transitions (s, a, r, s') during the sampling from the replay buffer.

Thus, our intention is to advance upon existing methodologies, such as the Prioritized Experience Replay buffer, by incorporating the nuanced aspects of transition parameters, such as transition frequency, through the development of a comprehensive framework. In our effort, we aim to address this problem by conducting a thorough analysis and implementing a series of experiments on different environments targeting different parameters during the sampling process from the experience replay buffer. Through these efforts, our goal is to enhance and optimize the efficiency and performance of reinforcement learning algorithms.

III. Our Solution

A. Utilization of Chunks

Traditionally, transitions (State, Action, Reward, State) have been treated as the fundamental unit within the replay buffer. However, we propose a paradigm shift towards utilizing ”chunks,” which consist of sequentially contiguous blocks of transitions of a constant size. These chunks will be assigned weights and sampled to form batches for training the agent. Our framework operates in two stages:

1) **Calculation of Weights for Each Chunk:** We will incorporate various factors into the weight assignment process, as detailed in the subsequent section.
2) **Sampling Using These Weights to Form a Batch:** Once the weights are computed, our framework will utilize them to sample a batch from the entire replay buffer. Additionally, each weight update will be multiplied by an importance sampling term to mitigate bias.

B. Computation of Weights

Our framework considers several factors when assigning weights to each chunk, offering configurability to accommodate diverse training scenarios. The following factors are explored:

1) **Temporal Difference Error (TDE):** Inspired by prior research, we propose increasing the weights for transitions associated with high TDEs. High TDEs indicate instances where the agent experiences surprises, facilitating accelerated training.

$$y = R_{t+1} + \gamma * max_a Q(S_t + 1, a)$$

$$\Delta(\text{TD Error}) = (y - Q(S_t, A_t))^2$$

2) **Immediate Rewards:** Incorporating immediate rewards R_{t+1} into weight calculations is hypothesized to enhance the training process, especially in environments with sparse rewards.
3) **Immediate Estimated Returns:** We plan to include estimates of immediate returns in weight computations, favoring chunks with higher expected returns for the agent's learning process.

$$Expected\ Returns = R_t + 1 + *max_a Q(S_t + 1, a)$$

4) **Rarity:** Frequency of occurrence in the replay buffer will be factored into weight calculations. Chunks with lower frequency may be assigned higher weights, particularly in sparse reward scenarios.

$$Rarirty(i) = \frac{\sum_r freq_r}{freq_i}$$

where $freq_i$ is the frequency of the i^{th} transition
5) **Trace Bonus:** Exponentially decaying weights will be introduced for preceding chunks to emphasize the importance of understanding the sequential nature of experiences leading to valuable transitions.

$$Trace\ Value = \alpha * e^{(-\beta * t)}$$

where α, β are scaling factors, and t is the number of episodes completed after sampling.
6) **Staleness Bonus:** To address underrepresented experiences, a staleness bonus will be assigned to transitions that have not been sampled for an extended period, effectively increasing their weights within our framework.

$$Staleness\ Value = \eta * t$$

where η is the staleness factor and t is the number of episodes completed after sampling.

7) **Importance Sampling:** Drawing from previous studies, we advocate for adjusting sampling weights to prioritize transitions that contribute significantly to the learning process. By assigning higher weights to transitions with substantial impact, we enhance the efficiency of learning algorithms.

$$1/(N * p)^b$$

C. Approach and Technologies

In implementing our framework and conducting experiments, we will leverage the following technologies:

1) **PyTorch:** PyTorch provides an efficient platform for implementing and training complex neural networks, serving as the foundation for our learnable function within DQNs.
2) **OpenAI Gym:** Utilizing Gym's diverse environments, such as Atari games and robotics tasks, facilitates rigorous experimentation and benchmarking of our algorithms, ensuring robustness and generalization.

IV. Analysis

We conducted extensive experiments with different configurations of our framework spanning around 40 hours. Although our framework is in many ways agnostic to the training algorithm in use, we used a double DQN in out experiments. We performed experiments on three environments from the OpenAI gym [7] - Cartpole, lunar lander and Acrobot.

A. Optimization

The optimizations involved in designing and implementing such a framework proved to be non-trivial. This was primarily because updating weights of chunks belonging to the replay buffer could be only done in linear time. And the large sizes of replay buffers made even linear sweeps to cause large overheads. Scope for parallelization also remained scarce, as some of these update operations had dependencies to other weights of other chunks neighbouring it. We managed to optimize it by exploiting the efficient compute provided by the numpy [8] library. We managed to optimize most functionalities in our framework to take only around 50% more time when compared execution time without our framework (Table I). However, even with these optimizations, implementation of chunks proved to be quite costly and implementation of rarity was very slow in spite of using hashable data structures internally.

B. Effect of different weight assigning strategies

We tried to perform ablation studies to observe the effect of the different weight computing strategies, both independent and along with each other. The metrics we focused on were:

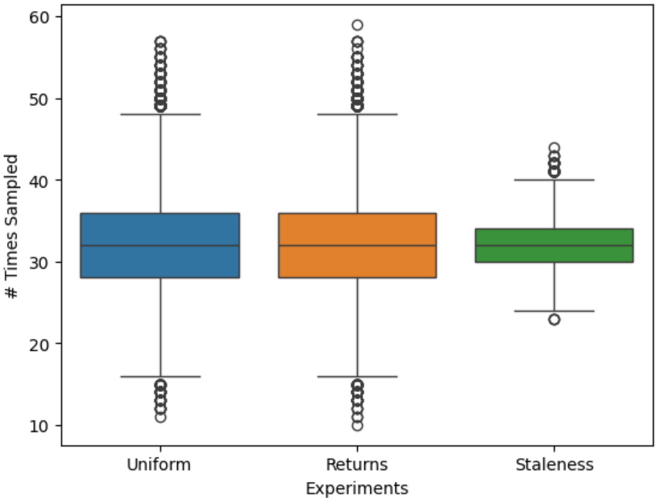


Fig. 1: A box plot showing the distribution of number of times each chunk was sampled in a particular experiment in the Cartpole environment.

- **Candid inference performance**, which involved running inference on checkpoints obtained while training fifty times and calculating the average score of these runs
- **Time taken to reach best performance**, in terms of both number of episodes as well as number of gradient updates.

1) Uniform sampling: We used uniform sampling as a baseline for performance. In this configuration, every chunk was equally likely to be sampled.

$$f(C_i) = \frac{1}{N}$$

Using uniform sampling on cartpole produced decent results (Table II). The agent reached the best possible performance score of 1600 in 500 episodes or around 38,000 steps. On an average, each chunk was sampled around 32 times before it was removed from the buffer (See Fig 1).

On applying uniform sampling on the Acrobat environment descent outcomes were received.The agent achieved its highest performance score of 35 at 244 episodes, and the inferences got stable from around 250 episodes roughly equivalent to 160,000 steps. On average, each data chunk underwent approximately 33 samplings before being removed from the buffer.

Utilizing uniform sampling within the Lunar Lander Environment yielded promising outcomes, as depicted in Table **??**. The agent achieved its optimal performance score of 300 within 1200 episodes, equivalent to approximately 100,000 steps. On average, each chunk was sampled approximately 35 times before being deleted from the buffer, as illustrated in Fig **??**.

2) Temporal Difference Error: In an attempt to replicate the results of [5], this configuration involved using temporal difference error calculated while training to be the sole factor

	Uniform Sampling	Our Framework	Vanilla Implementation
Cartpole	30.34	21.23	1.58
Acrobat	31.2	20.73	2.01
Lunar Lander	21.22	13.39	0.29

TABLE I: Performance of our framework compared to vanilla implementation in terms of number of gradient steps taken per second

Experiment	**TDE α**	**Rewards**	**Returns**	**IS**	**Trace**	**Staleness**	**Chunk Size**	**Best score**	**# Episodes**	**# Steps**
Baseline	✗	✗	✗	✗	✗	✗	1	1600	500	38829
TDE # 1	1	✗	✗	✗	✗	✗	1	714.52	150	11288
TDE # 2	0.6	✗	✗	✗	✗	✗	1	861.6	300	28884
TDE # 3	0.6	✗	✗	✓	✗	✗	1	1600	450	35504
Rewards	✗	✓	✗	✓	✗	✗	1	1600	550	85519
Returns	✗	✗	✓	✓	✗	✗	1	1600	600	82092
TDE # 4	0.6	✗	✗	✓	✓	✗	1	1600	500	65859
Staleness	✗	✗	✗	✓	✓	1	1	563.86	300	37588
TDE # 5	0.6	✗	✗	✓	✓	0.1	1	**1600**	**200**	**22628**
TDE # 6	0.6	✗	✗	✓	✓	0.1	2	1600	550	40296
TDE # 7	0.6	✗	✗	✓	✓	0.1	4	271.4	250	13983

TABLE II: Performance across experiments in the Cartpole environment. The best achievable score in this environment is 1600.

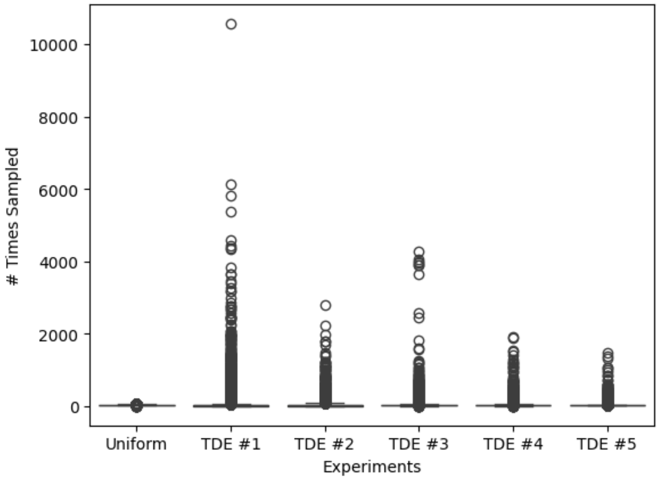


Fig. 2: Another box plot comparing distributions of different TDE experiments in Cartpole environment.

contributing to the weight of a chunk. A hyper parameter, $0 \leq \alpha \leq 1$, was also used to control the extent to which this would contribute to calculating weights.

$$f(C_i) = \Delta^\alpha$$

Two experiments were carried out using TDE on the cartpole environment. The first one used $\alpha = 1$, and managed to reach it's best score in merely 150 episodes or 11,000 steps (See Table II). However, neither was it able to reach it's maximum score, nor did it increase or even maintain it's score throughout the rest of the experiment. This is probably a result of the heavy bias induced by using TDE alone, which is also visually apparent from the distribution of how many times each chunk was sampled (Fig 2). In an attempt to dampen this bias, the second experiment used $\alpha = 0.6$, however, it produced only marginally better results.

TDE experiment was carried out on Arcobat enviroment using $\alpha = 0.6$ The inference breached the return threshold of 0 at around 160 episodes or 98856 steps but it kept on fluctuating or oscillating and later stabilized and started giving consistent returns after episode 475 or 143190 steps. However after another 200 episodes the inference started oscillating again and it never stabilized again. This might have been caused because of the bias introduced by giving priority to TDE.

The TDE experiment was conducted within the Lunar Lander environment, employing $\alpha = 0.6$. Inference results demonstrated a peak score of 260, observed around the 591st episode. During the training phase, scores stabilized from approximately the 120th to the 700th episode. However, thereafter, scores began to fluctuate significantly, leading to a decline in the moving average of scores. This fluctuation may be attributed to the potential bias introduced by prioritizing transitions based on TDE values (see Table **??**).

3) Importance Sampling: In order to mitigate the heavy bias induced by the weights, we multiplied each gradient step a weighted importance sampling term, as done in [5].

The use of IS drastically improved results on cartpole when using TDE (See Table II). The resulting performance surpassed the baseline by a reasonable margin. Due to it's effective impedance of bias, IS was used in all of the following experiments on the cartpole environment.

In the Lunar Lander Environment, employing Importance Sampling (IS) led to an earlier peak performance, observed at approximately the 100th episode. Furthermore, IS contributed to a reduction in the variances of rewards obtained from the environment. Although there was marginal disparity in the stabilized values between runs utilizing Temporal Difference Error Importance Sampling (TDE-IS) and those without IS, the moving average of TDE-IS peaked earlier compared to the baseline. However, regrettably, shortly after stabilizing, around the 460th episode, a decline in performance was observed. We speculate that this decline may be attributed to the diminishing training efficacy of TDE beyond a certain point.

4) Immediate Rewards: We attempted to use the immediate rewards obtained in a transition as a criterion for assigning weight.

Due to the nature of the environment of cartpole, wherein every step garnered an equal reward of 1, the results of using rewards based sampling on it was unremarkably similar to the baseline of uniform sampling.

The Acrobot environment features sparse rewards, leading to a scenario where transitions associated with high rewards consistently received elevated priority and weights. We can observe that by the fact that after some time in training the max amount of time a chunk was sampled was more than 70 .Consequently, this impeded the training process, as only those transitions with significant rewards present in the buffer were sampled. Consequently, it required more episodes (345 or 93,000 steps) to surpass the environment's return threshold. However, even after reaching this threshold, stability was elusive, with inference scores continually oscillating.

In the Lunar Lander environment, the scores exhibited high oscillation of scores. Initially, learning progress was gradual when considering rewards exclusively. However, in subsequent stages, learning accelerated significantly, nearly stabilizing to levels comparable to uniform sampling. This phenomenon may be attributed to the dense rewards characteristic of the environment. Notably, a similar pattern was observed in the inference results.

5) Estimated Returns: In this approach, we attempted to use the estimated return computed by the DQN agent while training to be the basis for assigning weights to chunks.

From Table II, we see that such a strategy is in fact, worse than our baseline for cartpole. Surprisingly, the distribution of number of times each chunk was sampled is very close to the uniform case (Fig 1). When we look at the course of estimated return values over the training process (Fig 3), it is clear that it is following a somewhat linear trend of increasing monotonically. This results in the weight distribution W, always being linearly skewed towards the most recent chunks added to the buffer. This effectively reduces the size of the buffer as older chunks have a very low probability of being sampled when the weights are normalized. Therefore, estimated returns based weight assignment wouldn't be a good idea for environments such as Cartpole where rewards are dense and uniform, which ensures that the Q value always increases.

Sampling using expected returns for the Acrobat environment showed similar results as baseline and reaching the stability at around 250 episodes. This is true for inference as well. After passing through the return threshold of the environment, consistently good inference returns were achieved.

Sampling based on expected returns in the Lunar Lander environment yielded outcomes comparable to the baseline. This similarity extended to the inference results as well. However, following the completion of 680 episodes, fluctuations in scores greater than those of the baseline were observed.

6) Rarity: For assigning weights based on the rarity of occurrence of a particular state, we discretized the state space into bins. This made it easy to accumulate occurrences in a neighbourhood of state space.

Due to challenges in optimizing and practical time constraints, experiments with rarity were not carried out in Cartpole and Acrobot environments.

Employing rarity-based sampling in the Lunar Lander Environment resulted in a decrease in performance compared to the baseline. Nonetheless, both approaches exhibited nearly identical curves on the plots, with the baseline (uniform sampling) showing slightly superior performance compared to rarity-based sampling.

7) Trace: In addition to the weights assigned using the above strategies, a fixed-length ”trace” window of exponentially decaying weights was applied to preceding chunks to emphasize the importance of understanding the sequential nature of experiences leading to valuable transitions.

In the Cartpole environment, a trace window of size 20, with a decay factor of 0.2 was applied on top of the TDE based weight calculation. The resulting performance was slightly worse than baseline (See table II). The decrease in performance is probably due to the overall increase in bias (See outliers in fig 2) when it is propagated by chunks to their preceding neighbours.

In the Acrobat environment, a trace window of size 15, with a decay factor of 0.5 was applied on top of the TDE based weight calculation. The results obtained were significantly better then TDE without trace. The max return along with the stability of inference returns was achieved in just 190 episodes or 73200 steps.Thus taking lesser gradient steps.The fluctuations in this experiment were significantly lesser than the uniform and just TDE experiments. Also we can observe that initially some chunks had much more sampling frequency but after the stability was achieved the sampling frequency got similar to the uniform case.

In the Lunar Lander environment, a trace window of size 20, coupled with a decay factor of 0.2, was implemented atop the TDE-based weight calculation method. The resultant performance closely mirrored that of the baseline. However, it's noteworthy that the average number of times a transition was sampled under this approach was approximately 22, which contrasts with the baseline's average of 35 samplings per transition.

8) Staleness: Staleness introduces a weight term based on when was the last time a chunk was sampled in a batch.

In the cartpole environment, we performed an experiment where we pushed it to the extreme and used staleness as the sole factor for assigning weights. The results (table II) were

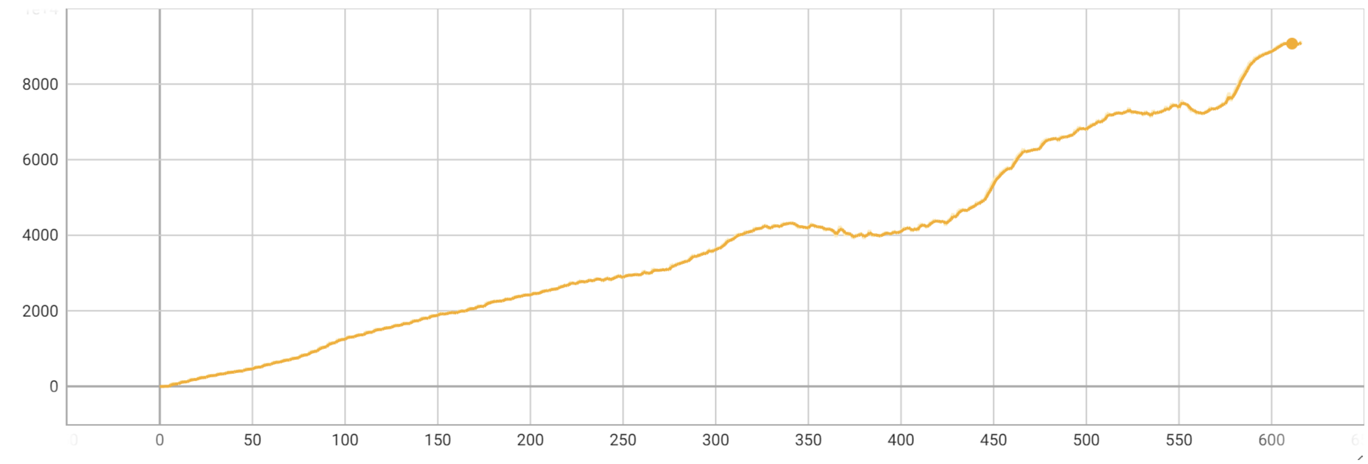


Fig. 3: Value of estimated return values over the course of training. The x-axis is the number of episodes.

significantly worse than the baseline. The distribution over number of times each chunk was narrower than the uniform case (1), implying using staleness alone squashes out a lot of the bias. However, when we use staleness in a controlled way, it manages to combat the additional bias induced by trace and is able to give the best performance in the environment.

Three experiments were conducted with staleness bonus provided along with tde, reward and immediate rewards respectively. The results were simmilar for each of them compared to those without staleness.Although More oscillations were seen in the immediate returns case. For all of them the stability was achieved at approximately 280 episodes. So we can say that the staleness did not impace the performance by much.

Our experiments in the Lunar Lander environment explored the impact of a staleness bonus. We investigated how this bonus interacts with rewards, returns, TD error (TDE), and its effect in isolation. Interestingly, the results across various metrics (scores, samples, inference values) showed little to no significant difference between these approaches. While staleness combined with other criteria didn't lead to substantial improvement, it also didn't significantly degrade performance compared to the baseline. It appears to converge to a value slightly below the baseline.

9) Chunk Size: Two experiments were carried out in the Cartpole environment with chunk size 2 and 4. Both of them gave significantly bad results when compared to the baseline (Table II).

In the acrobat environment experiment was conducted with Chunk size of 4. To be able to compare the results and to maintain the total number of transitions in the batch the batch size was reduced by 4 times. i.e from 32 to 8. It was observed that the episodes required to reach the return threshold of the environment was less at around 130 or 140 and also the big oscillations till reaching the threshold were also less compared to the chunk size 1 as the inference return increased steadily on average.This might be because of the fact that during training more similar transitions are coming in group and so the learning on them is more. Also the sampling frequency had similar trend as that of uniform but the entire trend was divided by 4 as expected. So the sampling frequency remained at 8 on average.

Again for Lunar Lander, two experiments were carried out with chunk size 2 and 4. Both of them produced unforgivably bad results compared to the baseline.

V. Results

Some overarching patterns from the experiments lead us to hypothesize the following:

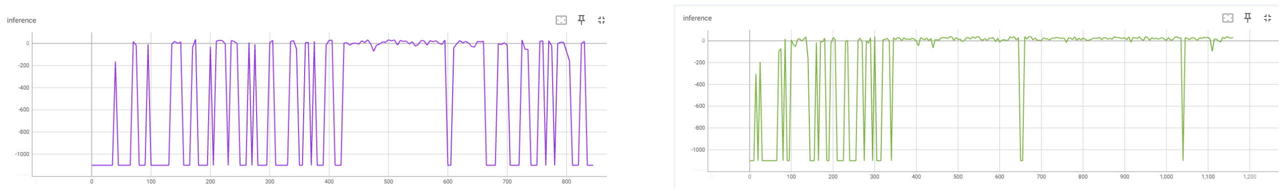
- Factors like TDE, rewards, returns, rarity and trace can induce heavy biases.
- Importance sampling factor and staleness weight can be used to reduce the effect of these biases.
- Estimated returns based weight assignment may not work well in environments where rewards are dense and uniform.

However, even though we managed to perform experiments on various configurations, we haven't performed enough experiments on a given configuration and environment to conclusively deduce results.

VI. Conclusion

In this project, we made a comprehensive framework for sampling from the replay buffer using some techniques that are novel to the best of our knowledge. We also observe some patterns on the framework's different configurations that could potentially be applied to different environments.

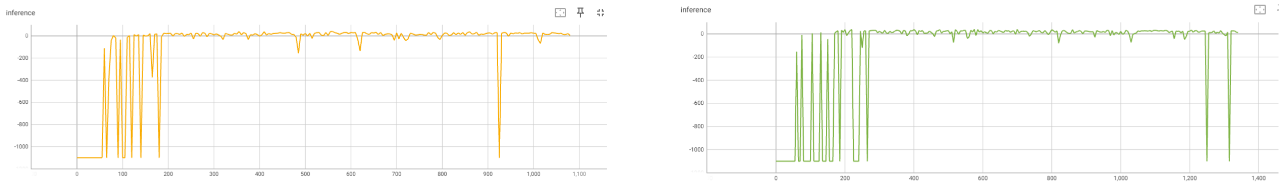
VII. Plots



(a) reward with trace

(b) return with trace

Fig. 4: Plots for acrobat environment



(a) tde with trace

(b) return with importance sampling

Fig. 5: Plots for acrobat enviroment

References

[1] Zhang, Shangtong, and Richard S. Sutton. ”A deeper look at experience replay.” arXiv preprint arXiv:1712.01275 (2017).

[2] Zha, Daochen, et al. ”Experience replay optimization.” arXiv preprint arXiv:1906.08387 (2019).

[3] Lu, Cong, et al. ”Synthetic experience replay.” Advances in Neural Information Processing Systems 36 (2024).

[4] Chen, Zhuoying, Huiping Li, and Rizhong Wang. ”Attention Loss Adjusted Prioritized Experience Replay.” arXiv preprint arXiv:2309.06684 (2023).

[5] Schaul, Tom, et al. ”Prioritized experience replay.” arXiv preprint arXiv:1511.05952 (2015).

[6] Fedus, William, et al. ”Revisiting fundamentals of experience replay.” International Conference on Machine Learning. PMLR, 2020.

[7] https://www.gymlibrary.dev/index.html

[8] https://numpy.org/doc/stable/index.html