Coreset-Augmented Experience Replay

Anonymous Author(s)
Affiliation
Address
email

Abstract

Experience replay (ER) is a widely used method in reinforcement learning to decorrelate sequential data and to improve sample efficiency by reusing past transitions during learning. The most widely used way to select transitions is to keep the agent's most recent k experiences. This limits the agent's ability to hold onto important transitions, and can have a large memory footprint. To address this limitation, we develop a coreset-based replay strategy designed to keep a compact yet diverse set of experiences. By employing coreset-augmented experience replay, we show that in some environments, it is possible to closely match the performance of a life-time sized recency buffer with 0.2% of the memory footprint, and storing transitions from arbitrary points of the agent's lifetime.

1 Introduction

Experience Replay (ER) [10] has been widely adopted in Reinforcement Learning (RL) to stabilize and enhance training across various applications [12, 17]. Typically, ER is implemented using a first-in-first-out buffer that stores recent transitions. Mini-batches are then sampled independently and uniformly, giving each transition an equal chance of being selected. While this uniform strategy is simple and effective, several alternatives have been proposed to improve the effectiveness and efficiency of ER [15, 1]. Among them, Prioritized Experience Replay (PER) [13, 15] has received particular attention. PER assigns higher sampling probabilities to transitions with larger temporal difference (TD) errors [18], aiming to focus learning on samples for which the agent has higher TD-error. However, recent studies [13] suggest that, when combined with function approximation, PER may lead to overestimation, and poor generalization in early learning. This invites further investigation into methods that can effectively surpass the performance of recency based Replay buffers.

One approach to increase the sample efficiency of ER is to effectively reduce its size. The literature presents mixed findings regarding the optimal size of experience replay buffers [4] [23]. Ideally, we would prefer a solution that mitigates the need to carefully tune buffer size altogether. In this work, we propose a new perspective on experience replay. Rather than restricting the agent to learning only from its most recent experiences, we explore retaining experiences from arbitrary points throughout the agent's lifetime. While it is theoretically possible to store all past experiences, doing so is impractical: it may require unbounded memory, which might lead to a degradation in learning performance [23], and this approach fails to account for the varying importance of different experiences in shaping the agent's behavior [8]. What is needed is a method that chooses a subset of the agent's experience to store and relearn from, to reduce the memory footprint of the agent.

Coreset construction methods are one approach to try to reduce the size of an experience replay buffer. They have been effectively used to reduce the data needs in machine learning [21, 11]. We introduce a novel experience replay mechanism called coreset-augmented replay, which aims to maintain a limited set of carefully selected experiences from across the agent's lifetime, according to a selection criterion. Additionally, to preserve the benefits of recency, we complement the coreset buffer with a

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

similarly sized recency buffer that captures the most recent experiences, ensuring the agent remains responsive to new and relevant information.

We investigate the effect of using this experience replay method in a variety of benchmark environments. To begin with, we test the method in the environment of Open Room, on variations of our method to test the usefulness of our method. While promising, our experiments showcase a few of the challenges of adapting prototype selection methods to this setting: choosing the right criteria such that stored transitions remain useful to the agent, and we showcase this by ejecting the oldest samples in the buffer every k steps. We also show promising results of the algorithm working in well known benchmark environments, Mountain Car [14] and Acrobot [19], while using a learned notion of similarity/distance between samples. In general, our method showcases a promising direction in using more effective sampling selection methods for experience replay.

2 Background, Problem Setting and Notation

We formalize our setting as a finite, discrete-time Markov Decision Process (MDP) [2], described by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$. In this framework, at each time step t, the agent first observes a state $S_t \in \mathcal{S}$. Based on this observation, it selects an action $A_t \in \mathcal{A}$, after which the environment transitions to a next state S_{t+1} drawn according to the transition probability distribution $P(\cdot|S_t, A_t)$. Concurrently, the agent receives a scalar reward signal $R_{t+1} = R(S_t, A_t)$ as feedback. Ultimately, the objective of the agent is to learn a policy π that maximizes the expected cumulative reward, defined as $\mathbb{E}[G_t \mid S_t = s] = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s]$, for each initial state S_0, where γ is the discount factor $\gamma \in [0, 1]$.

Q-learning [20] is one way of achieving the above goal, which we focus on in this paper. Q-learning estimates the action-value function for the optimal policy, via the off-policy temporal difference learning update rule:

$$\hat{q}(S_t, A_t) \leftarrow \hat{q}(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, a) - \hat{q}(S_t, A_t)] \tag{1}$$

for each state-action pair (S_t, A_t). The optimal policy π^* is derived from the action-value function $\hat{q}(s, \cdot)$, by choosing the action with the highest value, via $\arg\max_{a \in \mathcal{A}} \hat{q}(s, a)$. In environments with large or infinite state spaces, explicitly storing action-values $\hat{q}(s, a)$ for every state-action pair becomes infeasible. Therefore, one could use function approximation with neural networks to generalize action-values across states.

DQN [12] is a reinforcement learning algorithm that incorporates function approximation using deep neural networks. To address issues of sample correlation and policy non-stationarity—caused by global changes in the network parameters θ during stochastic gradient descent—DQN relies on experience replay (ER) to stabilize training.

Experience replay in its simplest form, operates by storing each new incoming transition in a queue of the most k recent samples, discarding the oldest sample to keep a constant queue size k. In this work, we refer to this type of experience replay buffer as a *recency* replay buffer or *recency* buffer. For the purpose of reducing the memory footprint of the agent, we would like to keep a subset of the agent's interactions with the environment, and sample transitions uniformly from that subset. We call this subset a *coreset* buffer.

In machine learning, a coreset is a small subset of a dataset that approximately preserves key properties of the original data, enabling more efficient computation of optimization problems. To construct such a subset, especially in settings like reinforcement learning where data arrives continuously and online, we must first define a way to measure similarity/distance between samples. In this work, we use a *distance representation* $\phi : (\mathcal{S} \times \mathcal{A} \times \mathcal{R} \times \mathcal{S}) \to X$, where X denotes the image of the representation, mapping samples into a space where similarity and diversity can be meaningfully quantified. This representation forms the basis for selecting diverse samples during coreset construction.

The choice of the distance representation ϕ determines the measure with which we determine how diverse or similar samples are to one another. We discuss how this representation can be chosen based on domain knowledge or learned automatically in subsequent sections. After the choice of the distance representation ϕ is made, The distance between each pair of samples x_i, x_j is computed using their Euclidean distance $\|x_i - x_j\|_2$.

Here, We specifically focus on online coreset construction [22], where the goal is to incrementally select diverse transitions. We adapt the Online Greedy algorithm [16] to be the reinforcement learning setting. Online Greedy uses the similarity between the points to construct a matrix of pair wise similarities $K_{ij} = \kappa(x_i, x_j)$. The method uses the gaussian Radial Basis Function (RBF) Kernel and by taking the Euclidean distance of each of the samples' distance representation ϕ, calculates their similarities:

$$\kappa(x_i, x_j) = \exp(-\eta||\phi(x_i) - \phi(x_j)||_2) \tag{2}$$

where η is the shape parameter. The matrix K is called the Gram matrix [7]. To maximize the diversity of samples in the coreset, for each incoming sample, Online Greedy maximizes the logarithm of the absolute value of the determinant of the Gram matrix K (See 3), This objective, captures the diversity of the coreset:

$$\mathrm{D}(C) = \log|\det(K)| \tag{3}$$

3 Coreset-Augmented Experience Replay

3.1 Online Greedy

In order to investigate the effectiveness of diversity-based selection in experience replay, we adapt Online Greedy to the reinforcement learning setting. The Online Greedy algorithm (See Algorithm 1) maintains a fixed-size coreset by greedily inserting transitions that maximize the logarithm of the absolute value of the Gram matrix K, constructed from the pairwise similarities from current samples in the coreset buffer, to ensure that samples stored in the coreset buffer remain as diverse as possible.

Algorithm 1 Online Greedy with coreset buffer

Require: Similarity function $\kappa(\cdot,\cdot)$, threshold for coreset size N, function $\log|\det(\cdot)|$, candidacy rate η

Initialize coreset $C \leftarrow \emptyset$
while $|C| < N$ **do**
    Receive transition t
    $C \leftarrow C \cup \{t\}$
end while
for all incoming transition t **do**
    $\lambda \sim U(0,1)$
    if $\lambda > \eta$ **then**
        break
    end if
    $G_0 \leftarrow \log|\det(K(C))|$ ▷ $K(C)$: Gram matrix of C
    Initialize best_gain $\leftarrow 0$, best_swap_index $\leftarrow$ `None`
    for $i = 1$ to N **do**
        $C' \leftarrow C$ with $C[i]$ replaced by t
        $G_i \leftarrow \log|\det(K(C'))|$
        if $G_i - G_0 >$ best_gain **then**
            best_gain $\leftarrow G_i - G_0$
            best_swap_index $\leftarrow i$
        end if
    end for
    if best_gain > 0 **then**
        Replace C[best_swap_index] with t
    end if
end for

We adapt the Online Greedy algorithm to our setting as follows. Initially, the buffer is pre-filled with the most recent transitions. For each new incoming transition, we consider replacing it with each existing transition in the buffer individually. For each potential replacement, we recompute

the diversity metric D over the resulting buffer. The transition is only retained if one of these replacements yields an increase in the overall diversity; in that case, we apply the swap that results in the greatest increase. If no such improvement is observed, the incoming transition is discarded.

In its current form, the algorithm will not be able to react to changes in the environment sufficiently quickly. Samples selected for inclusion in the coreset buffer do not necessarily represent the agent's most recent interactions with the environment, which may hinder the agent's ability to adapt online and learn from its latest experiences. In order to mitigate this, we also include a similarly sized recency buffer. We then sample from the coreset and recency buffers in proportion to their respective sizes, combine the selected transitions into a mini-batch, and use this batch for training with learning algorithms such as DQN.

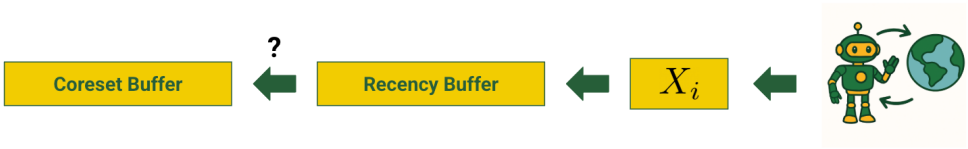


Figure 1: The composite replay buffer. As the agent interacts with its environment at each time step, new samples arrive and are first put in the recency buffer. Once they exit the recency buffer, they are checked as candidates to be added to the coreset buffer.

3.1.1 Distance Diversity Criterion

It is difficult to scale the size of the coreset in the prior method, as the number of operations scales with the order of $O(n^4)$ where n is the size of the coreset. To mitigate this, we adapt a dictionary sparsification procedure [6] to diversify our coreset buffer. This approach evaluates the similarity between the incoming sample and each sample already in the replay buffer, and admits (rejects) the new sample only if its similarity scores is smaller (exceeds) a predefined threshold, for every sample in the coreset buffer.

$$\kappa(x_i, x_i) - \frac{\kappa(x_i, x_j)^2}{\kappa(x_j, x_j)} \geq \delta^2 \tag{4}$$

We simplify this criterion by setting $\kappa(x_t, x_t) = 1$, reflecting the assumption that a sample is maximally similar to itself. When the buffer reaches capacity, we apply a sparsification procedure (see Algorithm 3) to free up space for new samples.

The diversity criterion coreset then works by evaluating every incoming sample, and using equation 5 to decide whether to reject or accept incoming samples.

$$1 - \kappa(x_i, x_j)^2 < \delta^2 \tag{5}$$

if the condition holds for all samples x_i in the coreset, that means that the sample is sufficiently dissimilar to all other samples and can be accepted to the coreset (See algorithm 2).

4 Experiments

How can these methods be adapted to experience replay to reduce its memory footprint and improve sample efficiency? Can they be adapted to a fully specified agent without modification? This paper investigates these questions by running experiments with different coreset strategies in standard reinforcement learning tasks.

Algorithm 2 Transition Acceptance with Periodic Coreset Sparsification

Require: Incoming transition t, similarity function $\text{sim}(p, t)$, threshold δ, interval k

Initialize coreset $C \leftarrow \emptyset$
Initialize step counter $i \leftarrow 0$
function PROCESSTRANSITION(t)
    $i \leftarrow i + 1$
    if $i \bmod k = 0$ **then**
        Remove the oldest item from C
    end if
    if $C = \emptyset$ **then**
        Accept t and add to C
        return
    end if
    if coreset is full **then**
        Sparsify C
    end if
    for all $p \in C$ **do**
        if $1 - \kappa(p, t)^2 < \delta^2$ **then**
            Reject t
            return
        end if
    end for
    Accept t and add to C
end function

Algorithm 3 Coreset Sparsification

Require: Current coreset C, similarity function $\text{sim}(\cdot, \cdot)$, threshold δ

Initialize sparsed_coreset $\leftarrow \emptyset$
$\Delta \leftarrow \eta \cdot \delta$ ▷ Increase threshold by factor of $\eta > 1$
for all $p \in \text{reverse}(C)$ **do** ▷ From most recent to oldest
    for all $t \in$ sparsed_coreset **do**
        if $1 - \kappa(p, t)^2 < \delta^2$ **then**
            continue to next p
        end if
    end for
    Add p to sparsed_coreset
end for
$C \leftarrow$ sparsed_coreset
$\delta \leftarrow \Delta$ ▷ Update threshold

4.1 The Open Room Environment

We begin this section with experiments in a simple gridworld environment, called Open Room.

The environment consists of an empty room, and a square region in the bottom right corner as the goal state. The observation given to the agent is are the (x, y) coordinates in the room. The agent takes action a of **left**: $[-1, 0]$, **right**: $[1, 0]$, **up**: $[0, 1]$, and **down**: $[0, -1]$ directions, and some uniform noise $\epsilon \sim U(0, 1)$ is added to the action to make the environment stochastic. The episode terminates when the agent reaches the goal region, where the agent receives its only non-zero reward of +1.

For these experiments, we compare four variants of experience replay: (1) a recency buffer containing all experiences over the agent's lifetime, (2) a recency buffer with the 200 most recent experiences, (3) the Online Greedy coreset construction method with a fixed buffer size of 100, and (4) the composite buffer defined in the methodology section.

We use DQN as the policy optimization algorithm, with a neural network consisting of two hidden layers, each with 32 units. The target network is updated every 128 time steps. The distance representation ϕ is chosen as the (x, y) coordinates of the agent in the environment. For the input of

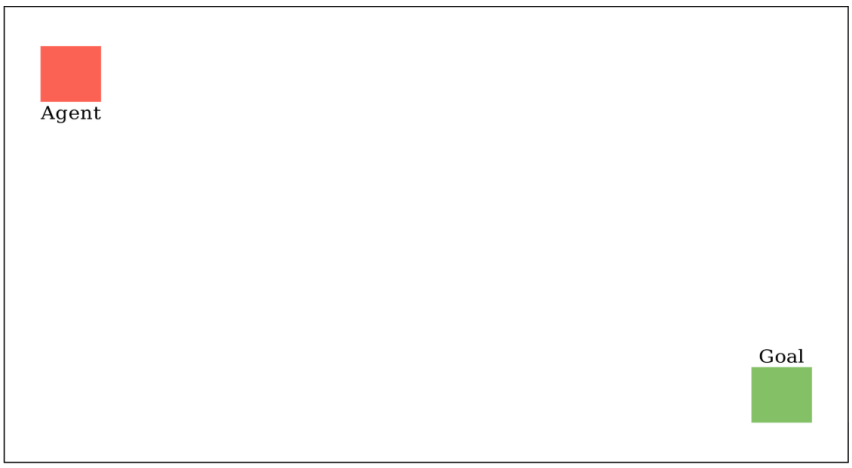


Figure 2: The Open Room Enviornment. The goal of the agent is to reach the goal state.

the network, we represent each continuous (x, y) coordinate as a one-hot vector by discretizing each axis into 20 uniform buckets. The resulting feature is a 40-dimensional binary vector with one active entry per axis. The η in the RBF kernel is set to 5. To avoid over-computation, we only accept 10% of the seen samples by the agent. The intuition is that transitions close to each other are also similar.

Optimization is performed using Adam [9] with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. To select the learning rate for each variant, we conduct a grid search over values $[10^{-7}, 5 \times 10^{-7}, 10^{-6}, 5 \times 10^{-6}, 10^{-5}, 5 \times 10^{-5}, 10^{-4}, 5 \times 10^{-4}, 10^{-3}, 5 \times 10^{-3}, 10^{-2}]$ using 30 random seeds. We then evaluate the best configuration for each variant over 100 independent seeds to generate the final performance plots.

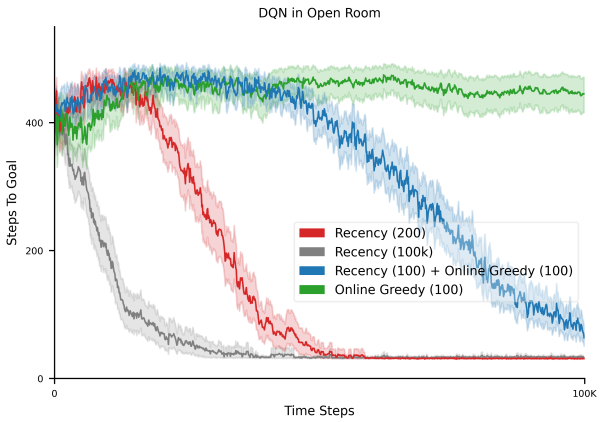


Figure 3: The use of a coreset replay buffer in the Open Room Environment, with DQN. The y-axis represents the number of steps it takes the agent to reach the goal. As one can see, A coreset buffer made using Online Greedy does not learn to reach the goal,

Using ejection improves the performance the and the algorithm can recover the performance of the coreset buffer.

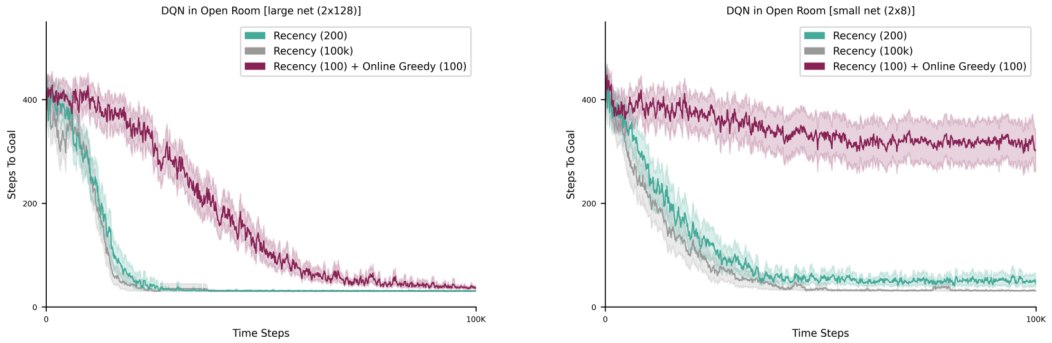
The performance of the algorithm is highly degraded in the absence of a recency buffer. This is expected since the agent is unable to react to changes or keep track of new experiences, as samples can be from arbitrary points of the agents experience.

4.1.1 The Effect of Network Size

To better understand the impact of buffer composition and network capacity on learning performance, we begin by examining a key question: *Why might the network perform poorly during training?* One possible explanation is that inaccurate updates—arising from suboptimal samples—propagate

through the network and consume a significant portion of its capacity, hindering generalization and learning.

To investigate this hypothesis, we analyze the effect of increasing the size of the neural network. In Figure 4b, subfigure (a) shows performance when using a network with 128 units per layer, while subfigure (b) depicts results with only 8 units per layer. In both configurations, agents trained using large and small recency buffers (of size 10^5 and 200, respectively) are still able to solve the task effectively, however, the coreset buffer method suffers in the case with a smaller network, suggesting that the network's capacity plays a critical role in mitigating the impact of harmful updates.



(a) Coreset Augmented Replay with a Larger Network size 2 layers, each with 128 units.

(b) Coreset Augmented replay with a smaller network size of 2 layers, each with 8 units. The performance of the network degrades significantly with this smaller size.

Figure 4: The effect of network size on the performance of coreset Augmented replay. The performance degrades when using smaller network sizes.

Network size affects the performance of a Composite Buffer. Specifically, increasing the size of the network (its capacity) seems to help the composite buffer learn better, but learning is still slower than both a small and large recency buffers.

4.1.2 Composite coreset with Ejection

Having established that both small and large networks can learn effectively with recency-based buffers, we next ask: *What accounts for the performance degradation observed when using the coreset buffer?* By design, the coreset construction procedure favors diversity, selecting samples that are dissimilar from one another. While this promotes broad coverage of the state space, it may inadvertently introduce challenges.

One plausible explanation is that the resulting buffer becomes highly off-policy, containing older samples that no longer reflect the agent's current behavior, thereby reducing their usefulness for learning [4]. Figure 5 supports this hypothesis by illustrating a modification to the composite buffer in which the oldest sample in the coreset are periodically ejected. Specifically, we remove the oldest sample every k steps, evaluating values of $k = 10, 20, 50$. We find that more frequent ejection (i.e., lower values of k) yields better performance, effectively making the coreset buffer behave more like a recency buffer—an effect clearly reflected in the improved learning outcomes.

4.2 Value-Based Distance Representation

In general, it is not obvious how one can choose a distance representation to adequately address the question of sample similarity in an environment. Can we use what the model has learned as a way to represent samples? Specifically, can we use the current learned action-value function $\hat{q}(s, \cdot)$ as a distance representation? Does the vector $\hat{q}(s, \cdot)$ serve as a good distance representation for coreset construction?

In order to answer the previous question, we test this in two different control environments, namely Mountain Car [14] and Acrobot [19]. For comparison, we use more than one Q-Learning method, by using Expected QRC (EQRC) [5] (See Figure 6b and 6d). In order to show the utility of our method

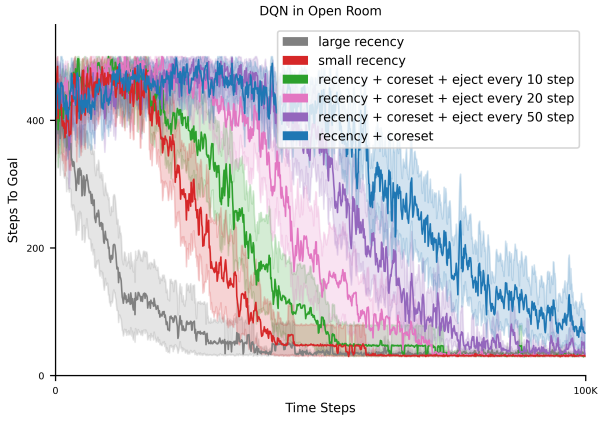
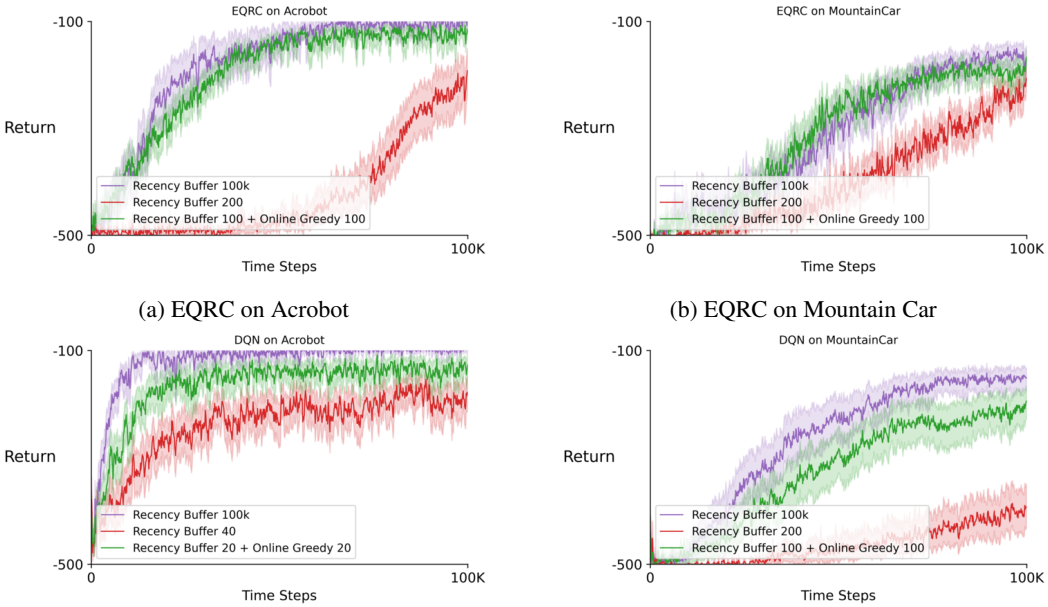


Figure 5: Performance of DQN in the Open Room domain using various configurations of the composite replay buffer. Each line color represents a different buffer variant. The key comparison is between the blue and green curves, which illustrate the performance difference between a composite buffer with recency and one without.

for small buffer sizes, we identify the smallest buffer size at which the recency-based method's performance begins to deteriorate. In such a setting, we can see that using the composite replay buffer using a value-based distance representation ϕ recovers performance. The experiment setup is the same as in 3.



(a) EQRC on Acrobot

(b) EQRC on Mountain Car

(c) DQN on Acrobot

(d) DQN on Mountain Car

Figure 6: Performance comparison on Acrobot and Mountain Car using a value-based distance representation for coreset construction. Subfigures (a) and (b) show results with Expected QRC (EQRC), while (c) and (d) show results with DQN. In each case, we compare a small and large recency buffer against a composite buffer using the learned action-value function $\hat{q}(s, \cdot)$ as a distance representation. The choice of the size of the small and composite buffers are based on when the small buffer fails. The composite buffer improves performance, in this setting where a equally-sized recency buffer alone is not insufficient due to limited capacity.

4.3 Distance Criterion

The computational cost of Online Greedy prohibits the feasibility of experiments with larger coreset buffer sizes. To address this, we adapt a coreset construction method [6], that scales $O(n^2)$ with the size of the coreset buffer.

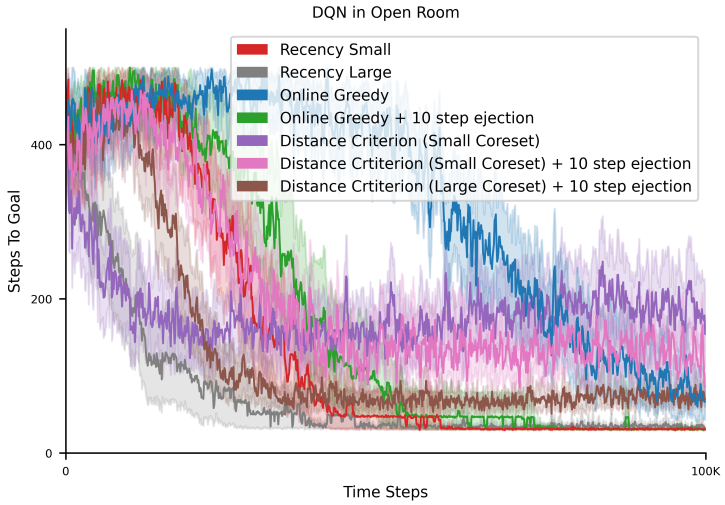


Figure 7: Performance of DQN in the Open Room environment using various replay strategies. The y-axis shows the number of steps required to reach the goal (lower is better), and the x-axis shows the number of training time steps. We compare recency-based buffers (small and large), Online Greedy (with and without 10-step ejection), and distance-based coreset buffers of varying sizes, with and without an ejection mechanism. Ejection improves performance for Online Greedy. Notably, recency-based replay consistently achieves the best performance, while distance-based coresets without ejection struggle to match their effectiveness, especially later in learning.

For the distance-based coreset criterion, we use the following hyperparameter values: coreset size of 100, an initial diversity threshold $\delta = 0.05$, a scaling factor $\eta = 1.2$, and an ejection rate $k = 10$, which determines how frequently the oldest transition is removed. We observe that, under both the ejection and non-ejection variants, the agent fails to learn the optimal policy. To ensure a fair comparison, we adopt the same configuration used in Figure 3.

The distance-based criterion offers a useful way to improve learning. It helps the agent learn faster with a smaller buffer, especially early in training. However, performance tends to drop as training continues. This may be because the coreset stops changing once it is full. To test this idea, we added a simple ejection mechanism. This change slightly hurts early performance but improves learning later on.

5 Limitations and Future Work

While coreset construction shows promise as a direction for improving experience replay mechanisms, there are several limitations that warrant further consideration. The assumption that diversity is a good criterion for learning remains unproven. There is a need for more theoretical work to determine whether diversity is an appropriate and effective objective, or whether alternative criteria could offer better performance. The relationship between the chosen criterion for coreset construction and the resulting agent performance is not yet fully understood. In particular, the extent to which diversity correlates with the utility of selected samples requires deeper investigation. This work assumes the ability to freely choose a distance representation for samples, but does not explore or recommend which types of representations are most beneficial.

There are several directions for future work that could build on this investigation. One promising direction is to evaluate the coreset construction approach in additional domains, such as the Atari Suite [3]. Understanding the impact of different distance representations on performance, and the types of samples prioritized through diversity maximization, requires more investigation. Moreover, a broader exploration of alternative diversity criteria and coreset construction methods is warranted. The current study does not exhaust the space of possible strategies, and other methods may yield better results or improved efficiency. Finally, the computational cost of online coreset construction is

non-trivial. Reducing this cost is an important area for future research, particularly for applications requiring real-time or resource-constrained execution.

6 Conclusion

In this paper, we explored the use of coreset-based construction strategies for experience replay in reinforcement learning. We examined two distinct coreset mechanisms and introduced a value-based distance representation, which enables compatibility with value-learning methods such as DQN. Our experiments show that coreset-augmented replay buffers can outperform simple recency-based baselines in terms of learning performance in some settings. Notably, our learned distance metric enables the replay buffer to retain diverse transitions, improving learning performance in constrained memory settings. These findings highlight the potential of coreset-augmented replay as a flexible and efficient alternative to traditional experience replay strategies.

References

[1] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

[2] Andrew Barto and Richard S. Sutton. Reinforcement learning: an introduction. 2018.

[3] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of artificial intelligence research*, 47:253–279, 2013.

[4] William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. Revisiting fundamentals of experience replay. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3061–3071. PMLR, 13–18 Jul 2020.

[5] Sina Ghiassian, Andrew Patterson, Shivam Garg, Dhawal Gupta, Adam White, and Martha White. Gradient temporal-difference learning with regularized corrections. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3524–3534. PMLR, 13–18 Jul 2020.

[6] Paul Honeine. Analyzing sparse dictionaries for online learning with kernels. *IEEE Transactions on Signal Processing*, 63(23):6343–6353, 2015.

[7] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 2nd edition, 2013.

[8] Angelos Katharopoulos and François Fleuret. Not all samples are created equal: Deep learning with importance sampling. In *International conference on machine learning*, pages 2525–2534. PMLR, 2018.

[9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8:293–321, 1992.

[11] Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for data-efficient training of machine learning models. In *International Conference on Machine Learning*, pages 6950–6960. PMLR, 2020.

[12] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[13] Parham Mohammad Panahi. Investigating the interplay of prioritized replay and generalization. *Proceedings of the 1st Reinforcement Learning Conference (RLC 2025)*, 2024.

[14] Andrew William Moore. Efficient memory-based learning for robot control. Technical report, University of Cambridge, Computer Laboratory, 1990.

[15] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[16] Matthew Schlegel, Yangchen Pan, Jiecao Chen, and Martha White. Adapting kernel representations online using submodular maximization. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3037–3046. PMLR, 06–11 Aug 2017.

[17] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[18] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44, 1988.

[19] Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, 8, 1995.

[20] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[21] Yu Yang, Hao Kang, and Baharan Mirzasoleiman. Towards sustainable learning: Coresets for data-efficient deep learning. In *International Conference on Machine Learning*, pages 39314–39330. PMLR, 2023.

[22] Jaehong Yoon, Divyam Madaan, Eunho Yang, and Sung Ju Hwang. Online coreset selection for rehearsal-based continual learning. In *Proceedings of the 10th International Conference on Learning Representations (ICLR)*, 2022.

[23] Shangtong Zhang and Richard S Sutton. A deeper look at experience replay. *NIPS 2017 Deep Reinforcement Learning Symposium*, 2017.