Frugal Actor-Critic: Sample Efficient Off-Policy Deep Reinforcement Learning Using Unique Experiences

Nikhil Kumar Singh
Indian Institute of Technology, Kanpur
UP, India
nksingh@cse.iitk.ac.in

Indranil Saha
Indian Institute of Technology, Kanpur
UP, India
isaha@cse.iitk.ac.in

ABSTRACT

Efficient utilization of the replay buffer plays a significant role in the off-policy actor-critic reinforcement learning (RL) algorithms used for model-free control policy synthesis for complex dynamical systems. We propose a method for achieving sample efficiency, which focuses on selecting unique samples and adding them to the replay buffer during the exploration with the goal of reducing the buffer size and maintaining the independent and identically distributed (IID) nature of the samples. Our method is based on selecting an important subset of the set of state variables from the experiences encountered during the initial phase of random exploration, partitioning the state space into a set of abstract states based on the selected important state variables, and finally selecting the experiences with unique state-reward combination by using a kernel density estimator. We formally prove that the off-policy actor-critic algorithm incorporating the proposed method for unique experience accumulation converges faster than the vanilla off-policy actor-critic algorithm. Furthermore, we evaluate our method by comparing it with two state-of-the-art actor-critic RL algorithms on several continuous control benchmarks available in the Gym environment. Experimental results demonstrate that our method achieves a significant reduction in the size of the replay buffer for all the benchmarks while achieving either faster convergent or better reward accumulation compared to the baseline algorithms.

KEYWORDS

Sample efficiency, Reinforcement learning, Control policy synthesis, Off-policy actor-critic algorithm

ACM Reference Format:
Nikhil Kumar Singh and Indranil Saha. 2024. Frugal Actor-Critic: Sample Efficient Off-Policy Deep Reinforcement Learning Using Unique Experiences. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 9 pages.

1 INTRODUCTION

The off-policy actor-critic reinforcement learning (RL) framework has proven to be a powerful mechanism for solving the model-free control policy synthesis problem for complex dynamical systems with continuous state and action space. Experience replay plays a crucial role in the off-policy actor-critic RL algorithms. The experiences in the form of a tuple containing the current state, the applied action, the next state, and the acquired reward are collected through off-policy exploration and stored in the replay buffer. These experiences are replayed to train the critic using optimization algorithms such as stochastic gradient descent (SGD) [1, 2], which in turn guides the training of the actor to improve the target policy. The performance of the target policy thus depends significantly on the quality of the experiences available in the replay buffer.

To ensure the high quality of the experiences based on which the critic is trained, researchers have proposed several methodologies [16, 27, 31] that focus on prioritizing the experiences based on various heuristics. However, as the replay buffer size is generally very large to ensure that the samples are independent and identically distributed (IID), prioritizing the samples incurs a significant computational overhead. In this paper, we rather explore an alternative strategy to ensure the quality of the experiences on which the critic is trained. We aim to devise a mechanism that can identify unique experiences and store them in the replay buffer when they provide knowledge different from what is already available with the experiences present in the replay buffer. Maintaining unique experiences in the replay buffer has two major advantages. First, having unique experiences in the replay buffer ensures that the samples are IID, which is essential for the stochastic gradient descent (SGD) optimization algorithm used in training the critic to converge faster by not suffering from instability caused by high estimation variance [21, 37] and without the necessity of employing expensive methods for prioritizing the samples within the reply buffer. Second, it helps in reducing the memory requirement, which is crucial for deploying actor-critic RL algorithms in memory-constrained embedded systems [26, 32].

In this paper, we present an algorithm named Frugal Actor-Critic (FAC) that strives to maintain unique experiences in the replay buffer. Ideally, the replay buffer should contain those experiences that provide unique learning to the agent. From a state, it may be possible to reach some other state and acquire the same reward by applying different actions. However, as the goal of the agent is to maximize the accumulated reward, it does not need to learn all the actions. Rather, it is enough to learn one of those actions that lead to a specific reward from a state while moving to another state. Thus, FAC attempts to identify experiences based on unique state-reward combinations.

FAC employs a state-space partitioning technique based on the discretization of the state variables to maintain unique experiences in the replay buffer. However, a complex dynamical system may have a high-dimensional state vector with correlated components.

Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand. © 2024 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

Thus, state-space partitioning based on all the state variables becomes computationally expensive. FAC employs a matrix decomposition based technique that uses the experiences encountered during the initial phase of random exploration to identify the most important and independent state variables that characterize the behavior of the dynamical system. Subsequently, the state space is partitioned based on the equivalence of the values assumed by the selected state variables. The goal of the FAC algorithm is to store experiences that are from different partitions of the state space. However, naively discarding samples whose state values are similar to those present in the replay buffer might lead to degradation in learning (since samples with similar states can have different rewards) and hence synthesis of sub-optimal control policy. Since learning depends on reward maximization, the control policy needs the knowledge of different rewards achieved for a given state (by taking different actions). To accommodate different rewards possible for similar states, we introduce a mechanism for estimating the density of a reward based on a kernel density estimator [33]. This mechanism aims to identify those experiences for a given state partition whose corresponding rewards are under-represented in the replay buffer.

The computational overhead of FAC is negligible, and it obviates the necessity of employing computationally demanding sample prioritization methods to improve the performance of the replay buffer based RL algorithms. We theoretically prove that the FAC algorithm is superior to the general off-policy actor-critic algorithms in storing IID samples in the replay buffer as well as in convergence. Moreover, we evaluate FAC experimentally by comparing it with two state-of-the-art actor-critic RL algorithms SAC [11] and TD3 [9] on several continuous control benchmarks available in the Gym environment [3]. Experimental results demonstrate that our method achieves a significant reduction in the size of the replay buffer for all the benchmarks while achieving either faster convergent or superior accumulated reward compared to the baseline algorithms. We also compare FAC with the state-of-the-art sample prioritization technique LABER [16], demonstrating that FAC consistently outperforms LABER in terms of convergence and accumulated reward. To the best of our knowledge, this is the first work that attempts to control the entries in the replay buffer to improve the performance of the off-policy actor-critic RL algorithms.

2 PROBLEM

2.1 Preliminaries

Reinforcement Learning. To solve a control policy synthesis problem using RL, we represent the model-free environment $\mathcal{M}$ as a tuple $\langle S, A, \mathcal{T}, R\rangle$, where S denotes the set of continuous states $s \in \mathbb{R}^p$ of the system and A denotes the set of continuous control actions $a \in \mathbb{R}^q$. The function $\mathcal{T} : \mathbb{R}^{p\times q\times p} \longrightarrow [0, \infty)$ represents the unknown dynamics of the system, i.e., the probability density of transitioning to the next state given the current state and action. A transition (or sample) x at time t is denoted by (s_t, a_t, r_t, s_{t+1}), where $s_t \in S$ is a state, $a_t \in A$ is an action, $r_t \in \mathbb{R}$ is the reward obtained by the agent by performing action a_t at state s_t, and $s_{t+1} \in S$ is the next state. We denote by R the reward function for $\mathcal{M}$ such that $R(\omega) = \sum_{t=0}^{T} \gamma^t \cdot r_t$, where $\omega = (x_0, \ldots, x_T)$ is the trace generated by a policy, and γ is the discount factor. To find the optimal policy $\pi^* : S \longrightarrow A$, we define a parameterized policy $\pi_\theta(a|s)$ as the probability of choosing an action a given state s corresponding to parameter θ, i.e.,

$$\pi_\theta(a|s) = \mathbb{P}[a|s; \theta]. \tag{1}$$

We define the cost function (reward) associated with the policy parameter θ as follows:

$$J(\theta) = \mathop{\mathbb{E}}_{\omega\sim\pi_\theta} [R(\omega)], \tag{2}$$

where $\mathop{\mathbb{E}}_{\omega\sim\pi_\theta}$ denotes the expectation operation over all traces ω generated by using the policy π_θ.

Our goal is to learn a policy that maximizes the reward. Mathematically,

$$\theta^* = \arg\max_\theta \ J(\theta). \tag{3}$$

Hence, our optimal policy is the one corresponding to θ^*, i.e., π_{θ^*}.

Deep RL Algorithms with Experience Replay. The direct way to find the optimal policy is via policy gradient (PG) algorithms. However, the PG algorithms are not sample-efficient and are unstable for continuous control problems. Actor-critic algorithms [15], on the other hand, are more sample-efficient as they use Q-function approximation (critic) instead of sample reward return.

The gradient for the Actor (policy) can be approximated as follows:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \ [\sum_{t=0}^{T} \nabla_\theta \ log \ \pi_\theta(a_t|s_t) \cdot Q(s_t, a_t)]. \tag{4}$$

The function $Q(s_t, a_t)$ is given as:

$$Q(s_t, u_t) = \mathbb{E}_{s_{t+1}\sim\mathbb{P}(.|s_t,u_t)} [r_t + \gamma * Q(s_{t+1}, u_{t+1})]. \tag{5}$$

The policy gradient, given by Equation (4), is used to find the optimal policy. The critic, on the other hand, uses an approximation to the action-value function to estimate the function $Q(s_t, a_t)$. This approximation function is learned using transitions stored in a *replay buffer*, which are collected under the current policy. Experience replay, i.e., storing the transitions in a replay buffer $\mathcal{R}$, plays a vital role in actor-critic based algorithms. It enables efficient learning by reusing the experiences (transitions) multiple times. Moreover, as the replay buffer samples are more IID compared to that in the case of on-policy, the learning is more stable due to better convergence.

We use the term general actor-critic (GAC) algorithms to denote the general class of off-policy actor-critic algorithms that use samples from the replay buffer for learning.

2.2 Problem Definition

Before we present the problem addressed in the paper formally, let us motivate the need to maintain unique samples in the replay buffer while synthesizing a control policy using the classical example of controlling an inverted pendulum to maintain its upright position. By default, a pendulum freely hangs in the downward position, and the goal is to balance it vertically upward. During the exploration phase, it spends much time swinging around the downward position towards the left and right. Consequently, the replay buffer is populated with samples mostly from a small region of the state space (around the downward position). In the learning phase, this behavior leads to oversampling from a specific subset of the experiences and under-sampling from the rest. This phenomenon leads to the less frequent replay of critical and rare experiences (for example, those corresponding to the pendulum's upright position), leading to the delay in learning the control policy.

The above discussion emphasizes the importance of collecting the experience from all regions of the state space uniformly. In general, maintaining unique experiences in the replay buffer is essential to maintain the IID characteristics of the samples, which is crucial for the superior performance of the optimization algorithms used for learning. Moreover, it also helps in optimizing the size of the replay buffer, which, though may not pose any challenge for a simple system like the pendulum, may be a bottleneck for large and complex systems such as a humanoid robot. Keeping this in mind, we now define the problem formally.

Definition 1 (Equivalent Experience)**.** Let us consider a transition $x = \langle s_t, a_t, r_t, s_{t+1} \rangle$. We define a transition x as an (approximate) *equivalent* of x' if following condition hold:

- `state`(x) and `state`(x') are in close proximity in some metric space, where `state`$(x) = s_t$.
- `reward`(x) and `reward`(x') are in close proximity in some metric space, where `reward`$(x) = r_t$.

Note that the two metric spaces for state and reward can be different. A transition x is *unique* w.r.t. a replay buffer $\mathcal{R}$ if $\mathcal{R}$ does not contain any transition x' which is equivalent to x.

Problem 1 (Sample Efficiency). *Let $\mathcal{M}$ be an open-loop dynamical system with unknown dynamics. Synthesize a control policy π^* with replay buffer $\mathcal{R}$ only containing unique samples , so that the expectation of rewards generated by the policy gets maximized. Mathematically,*

$$\pi^* = \arg\max_{\pi} \underset{\substack{\omega \sim \mathcal{R}^\pi \\ \mathcal{R}^\pi \sim \mathtt{sim}(\pi, \mathcal{M})}}{\mathbb{E}} [R(\omega)]$$
$$\text{s.t. } \mathcal{R}^\pi \text{ contains unique samples.} \quad (6)$$

Algorithm 1: Frugal Actor-Critic

1 **procedure** `frugal_actor_critic`$(\mathcal{M}, T)$
2 $\pi_\theta \leftarrow$ `initialize_policy()`
3 $\Omega \leftarrow$ `initial_rollout`$(\mathcal{M}, \pi_\theta)$
4 $\kappa \leftarrow$ `find_important_dimensions`(Ω)
5 $\alpha \leftarrow$ `partition_state_space`$(\mathcal{M}, \kappa, \mu)$
6 $\pi^* \leftarrow$ `learn`$(\mathcal{M}, \pi_\theta, \alpha, T)$
7 **return** π^*
8 **procedure** `find_important_dimensions`(Ω)
9 $\langle \mathsf{Q}, \mathsf{R}, \mathsf{E} \rangle \leftarrow$ `matrix_decomposition`(Ω)
10 $idx \leftarrow \mathtt{index}(|\mathtt{diag}(R)| \geq \nu \cdot |\mathtt{diag}(R)[0]|)$
11 $\kappa \leftarrow E[\mathtt{diag}(R)[idx]]$
12 **return** κ
13 **procedure** `learn` $(\mathcal{M}, \pi_\theta, \alpha, T)$
14 $\mathcal{R} \leftarrow \emptyset$; $\phi \leftarrow \{\}$; $s_0 \sim \rho^\pi$
15 **while** $t < T$ **do**
16 $a_t \leftarrow \pi_\theta(s_t)$
17 $\langle s_{t+1}, r_t \rangle \leftarrow \mathtt{sim}(\mathcal{M}, s_t, a_t)$
18 $\mathcal{R} \leftarrow$ `select_samples_for_insertion`(ϕ, α, s_t, r_t)
19 `Sample b transitions` $\langle s_i, a_i, r_i, s_{i+1} \rangle$ `from` $\mathcal{R}$
20 $\delta_i \leftarrow r_i + \gamma \cdot \max_a Q(s_{i+1}, a; \theta) - Q(s_i, a_i; \theta)$
21 $\theta_{k+1} \leftarrow \theta_k + lr * \sum_{i=1}^{b} \delta_i \cdot \nabla_\theta Q(s_i, a_i; \theta)$
22 $t \leftarrow t + 1$
23 **return** π^*_θ
24 **procedure** `select_samples_for_insertion`(ϕ, α, s_t, r_t)
25 $\mathfrak{R}(r_t, s_t) \leftarrow \int_{r_t - \beta}^{r_t + \beta} \rho_K(y, s_t)\, dy$
26 $\epsilon^s = \frac{\epsilon}{\exp(\frac{|\phi[\alpha(s)]|}{\eta})}$
27 **if** $\mathfrak{R}(r_t, s_t) < \epsilon^s$ **then**
28 $\mathcal{R} \leftarrow \mathcal{R} \cup \langle s_t, a_t, s_{t+1}, r_t \rangle$
29 $\phi[\alpha(s_t)] \leftarrow \phi[\alpha(s_t)] \cup \{r_t\}$
30 **else**
31 `discard`$(\langle s_t, a_t, s_{t+1}, r_t \rangle)$
32 **return** $\mathcal{R}$

3 THE FAC ALGORITHM

The FAC Algorithm is presented in Algorithm 1. FAC takes the model of the environment $\mathcal{M}$ and the number of time-steps T as input and returns the optimal control policy π^*. For this purpose, we start with a random policy (line 2) and collect transitions (rollouts) using it (line 3). Then, we find the important dimensions of the state-space from the collected rollouts using `find_important_dimensions` method (line 4). Along these important dimensions, the state-space is partitioned using `partition_state_space` method (line 5) to create a set of abstract states. Then, these abstract states are used to identify unique samples using `select_samples_for_insertion` function to find the optimal policy (line 6). We now present each of these functions in detail.

3.1 Finding Significant State Dimensions

The FAC algorithm identifies unique experiences based on the abstract states created through the partition of the state space. However, for high-dimensional complex nonlinear systems, naive partitioning of the state space leads to a large number of abstract states, hindering the scalability of the algorithm. Several states of a high-dimensional system have insignificant impacts on the overall behavior of the system. Moreover, as some of the state dimensions might be correlated, considering all the state dimensions unnecessarily introduce several abstract states that are not reachable by the system. Thus, as the first step (Lines 2-4), FAC finds a subset of independent state variables that captures the unique behaviors of the system.

FAC collects the trace $\Omega = (s_0, s_1, \ldots, s_{\mathcal{T}-1})$, s_i denoting the state at the i-th step, based on the random policy π_θ used in the initial rollout process before the learning starts at the $\mathcal{T}$-th step. These traces are now used to extract the subset of significant state dimensions κ. This is reasonable as the initial policy is completely random, and hence the data collected is sufficiently exhaustive to approximate the overall behavior. One could imagine that Principal Component Analysis (PCA) [35] is suitable for implementing this step. However, though PCA is capable of providing a low-dimensional representation of the original high-dimensional data, it is incapable of identifying the subset of state dimensions that contributes the most to the low-dimensional representation. Thus, we rather employ the rank-revealing QR decomposition [4], which gives us the state dimensions (i.e., features) sorted according to their singular values. This enables us to discard the less critical state dimensions containing minor information (i.e., low variance) or contain information that is already present in some other dimension (i.e., high correlation).

To apply QR decomposition on the initially collected system trace Ω, the function `find_important_dimensions` creates a matrix $[s_0^\top\, s_1^\top \ldots s_{\mathcal{T}-1}^\top]^\top$ with time-step as rows and state dimension as columns. The function `matrix_decomposition` refers to the permuted-QR decomposition such that $\Omega[:, E] = Q \cdot R$, where E is the permutation matrix. The function `index` fetches the index of columns (state dimensions) that satisfies the input constraints, i.e., features whose singular value is larger than ν times the largest singular value among all features. Here $\nu \in \mathbb{R}$ is a hyperparameter having a value between 0 and 1.

3.2 State-Space Partitioning

Once the important state dimensions κ are decided, FAC partitions the state space along those dimensions using the `partition_state_space` function (line 5). It uses a vector μ of length $|\kappa|$ whose element μ_i decides the granularity of the i-th important state, i.e., the range of values of the i-th important state is divided into μ_i partitions. Overall, the state space partitioning results in a non-uniform multi-dimensional grid structure, leading to an abstract state-space as $\mathcal{A} = \langle a_1, \ldots, a_v \rangle$ where $a_i \in \mathbb{R}^\kappa$. The hyperparameter μ controls the coarseness of the partition of the abstract state-space — smaller values for the elements in μ lead to a larger number of abstract states, i.e., the value of v. Note that although we have infinite states in the original state-space, we have a finite (v) number of abstract states.

We use $\alpha : S \longrightarrow \mathcal{A}$ to denote a state partition function. For a state $s \in S$, we denote the vector containing the values in κ dimensions, preserving the order as s^κ. The corresponding mapped abstract state a for state s is given as

$$a = \underset{\hat{a} \in \mathcal{A}}{\mathtt{arg\,min}}\, \|s^\kappa - \hat{a}\|. \tag{7}$$

3.3 Density Estimation

Once the abstract states are identified, FAC requires a mechanism to identify experiences providing unique rewards for an abstract state. We create a function $\phi : \mathcal{A} \to \mathcal{P}(\hat{R})$, where $\mathcal{P}(\hat{R})$ represents the power set of the set $\hat{R}$ having all the distinct reward values present in the replay buffer. Here, ϕ is a mapping from each abstract state to a set of dissimilar (unique) reward values. We use density estimation to control the insertion of a sample $\langle s, a, r, s' \rangle$ into the replay buffer, i.e., we estimate the density of the reward corresponding to the new sample using the set of rewards corresponding to an abstract state. If the new sample's reward density is low (i.e., distinct), then we insert it, else we reject it.

Our density estimation procedure is motivated by the kernel density estimator (KDE) [33]. We define the density estimation function for a reward r w.r.t a given state s as follows:

$$\rho_K(r, s) = \sum_i K(r - \phi[\alpha(s)]_i; h), \tag{8}$$

where i refers to the indexes of the different rewards corresponding to $\phi[\alpha(s)]$. Also, K refers to the kernel function, and h refers to the bandwidth, which acts as a smoothing parameter.

We define the reward density estimate (RDE) $\mathcal{R}$ of a reward sample r as follows:

$$\mathcal{R}(r, s) = \int_{r-\beta}^{r+\beta} \rho_K(y, s) \cdot dy, \tag{9}$$

where β controls the space width around the reward r used to compute reward density. An experience with state s and reward r is stored in the replay buffer if $\mathcal{R}(r, s) \le \epsilon$, where $\epsilon \in (0, 1)$ is a hyperparameter.

Generally, for a given abstract state, if we have a high RDE for any reward value (or range), this indicates that we have enough samples in that reward region. Consequently, more samples are not needed in that region. However, we might encounter a scenario where an abstract state has a large number of samples for those many different rewards. Consequently, each sample's probability mass contribution would be much less for the RDE of any reward region, ultimately leading to low RDE for all the reward regions of that abstract state. A low RDE by default would encourage new samples to be added to the replay buffer (since $\mathcal{R}(r, s) < \epsilon$), which we want to avoid as we already have enough samples. To address this issue, we use a dynamic value of ϵ by scaling it $1/\mathtt{exp}(\frac{|\phi[\alpha(s)]|}{\eta})$ times as follows:

$$\epsilon^s = \frac{\epsilon}{\mathtt{exp}(\frac{|\phi[\alpha(s)]|}{\eta})}. \tag{10}$$

Here η is a hyperparameter that represents a large number to keep the value of the exponential term closer to 1 in general and to make it attain a higher value only for an extremely large number of samples.

In Algorithm 1, RDE is computed at line 25. In case of low RDE (line 27), the transition tuple is inserted into replay buffer $\mathcal{R}$ (line 28) and ϕ is updated (line 29). In case the reward density is high, i.e., there are already a sufficient number of samples in the region, the transition is discarded (line 31).

3.4 Learning from Replay Buffers.

In Algorithm 1, the learning continues for T time-steps (line 15). Each simulation step of the policy under the environment at time t generates the transition tuple $\langle s_t, a_t, r_t, s_{t+1} \rangle$ (line 17). In line 18, the function `select_samples_for_insertion` to compute the RDE of the sample at time t is invoked, which in turn adds the sample to the replay buffer or rejects it. Subsequently, a mini-batch of samples is uniformly sampled from the replay buffer (line 19) and is used to update the control policy parameters θ (line 21).

FAC controls the insertion of samples into the replay buffer, thereby ensuring the uniqueness of samples present there. Practically, this is analogous to a sampling strategy where we pick unique samples. Controlling insertions into the replay buffer rather than sampling from the replay buffer is primarily driven by computational efficiency. Due to this, FAC incurs very little overhead vis-à-vis the GAC algorithms. The uniform sampling strategy (UER), which is computationally the least expensive [34], is used in FAC.

3.5 Computation Overhead

We now present an analysis of the complexity of the operations used in the FAC algorithm. In Line 3-5, we perform a few additional computations before starting the control policy synthesis. All these operations are performed before the training starts and involve negligible overhead. While inserting samples into the replay buffer, we perform some additional computations in Line 25 for density estimation for a given sample using the samples in its close vicinity (the number of such samples is fewer due to our insertion strategy). Hence, the time complexity of the insertion operation of FAC remains $O(1)$, i.e., constant in terms of the number of samples. Although negligible cost, the additional computations that we perform help us achieve the sampling from the replay buffer in $O(1)$ time due to uniform sampling strategy. Prioritization sampling, on the other hand, is computationally expensive with $O(\log n)$ complexity due to the updation of priorities of the batch samples (via priority queues). We circumvent this issue by controlling the insertion into the replay buffer (using operations having negligible cost) rather than performing prioritized sampling from the replay buffer.

4 THEORETICAL ANALYSIS

We use the notion of regret bound [10, 28] to compare the convergence of FAC algorithms vis-a-vis the GAC algorithm. The standard definition of cost function in reinforcement learning (refer Equation 2) uses a function of policy parameter θ. In this case, the traces/transitions are implicit as they are generated from a policy with parameter θ (i.e., π_θ). However, in the off-policy setting, the transitions used for learning are not directly generated by the policy but sampled from the replay buffer. To take this into account, we denote the cost function as $J(\theta, x)$ instead of $J(\theta)$ where x represents the transitions from replay buffer $\mathcal{R}$.

Theorem 1 (Convergence)**.** *Let $J(\theta, x)$ be an L-smooth convex cost function in policy parameter θ and $x \in \mathcal{R}$. Assume that the gradient $\nabla_\theta J(\theta, x)$ has σ^2-bounded variance for all θ. Let ζ represent the number of equivalent samples in a mini-batch of size b ($\zeta \leq b$) for GAC algorithms. Then, FAC would converge $\frac{b+\zeta^2+\zeta}{b}$ times faster than the GAC algorithms.*

Proof. Refer [30]. □

As mentioned earlier, a good mini-batch of data samples is one which has independent and identically distributed data samples. In statistics, this means that the samples should be as random as possible. One way of quantifying randomness is using entropy [29] in data. In the following theorem, we prove that the FAC algorithm improves the IID-ness of the samples stored in the replay buffer using the notion of entropy.

Theorem 2. *[Improving the IID characteristic of replay buffer] Let the replay buffer at a given instant consist of samples $(x_1, \ldots, x_m)$ with λ equivalent elements for a GAC algorithm. Then the increase in the entropy of the replay buffer samples populated via the FAC algorithm with respect to the GAC algorithm is given by*

$$\Delta\mathcal{H} = \frac{(\lambda + 1) \cdot \mathtt{log}(\lambda + 1)}{m}.$$

Proof. Refer [30]. □

In Theorem 1, we define convergence using a mini-batch of samples (size b) that is sampled from the replay buffer at any time step, whereas in Theorem 2, we define IID characteristics over the whole replay buffer (size m).

Table 1: Details of the Benchmarks

Environment	#Observations	#Actions	κ
Pendulum	3	1	1
MCC	2	1	1
LLC	8	2	2
Swimmer	8	2	4
Reacher	11	2	5
Hopper	11	3	4
Walker	17	6	5
Ant	27	8	8
Humanoid	376	17	16

5 EVALUATION

This section presents our experiments on synthesizing control policy using our proposed FAC algorithm.

5.1 Experimental Setup

We implement our FAC algorithm as a generic algorithm that can be used on top of any off-policy actor-critic algorithm. To simulate the environment, we use the `gym` simulator [3]. We run all the synthesized control policies on 100 random seeds to measure the reward accumulation. All experiments are carried out on an Ubuntu22.04 machine with Intel(R) Xeon(R) Gold 6226R @2.90 GHz×16 CPU, NVIDIA RTX A4000 32GB Graphics card, and 48 GB RAM. The source code of our implementation is available at https://github.com/iitkcpslab/FAC.

5.1.1 Baselines. We use two state-of-the-art deep RL algorithms, SAC [11] and TD3 [9], developed by introducing specific enhancements to the GAC algorithm, as the baseline for comparison with FAC algorithm as they are the most widely used model-free off-policy actor-critic algorithms for synthesizing controllers for "continuous" environments. While SAC uses stochastic policy gradient, TD3 uses deterministic policy gradient. Additionally, we compare FAC with state-of-the-art prioritization based algorithm LABER [16].

5.1.2 Benchmarks. We use 9 benchmarks based on Gym (Classic Control, Box2d, and Mujoco) in our experiments. The details of the benchmarks are provided in Table 1.

5.2 Performance Metrics

Here, we present the performance metrics we use to evaluate FAC in comparison to the baseline algorithms.

5.2.1 Convergence. We define convergence point (CP) as the timestep at which reward improvement settles down, i.e., it enters a band of reward range $[0.9 \cdot r_{max}, r_{max}]$ and remains within it thereafter

Table 2: Comparison of Convergence (CP), Replay buffer size ($\mathcal{R}$) and Reward (R) of FAC with baselines SAC and TD3

Model	CP_B	CP_F	Δ^{CP}	$\|\mathcal{R}_B\|$	$\|\mathcal{R}_F\|$	$\Delta^{\mathcal{R}}$	R_B	R_F	Δ^R	P
Pendulum (SAC)	17600	17600	–	20000	11412	42.94	$-143.97_{\pm 87.80}$	$-144.66_{\pm 88.53}$	−0.47	1.75
Pendulum (TD3)	18400	18400	–	20000	14137	29.31	$-138.80_{\pm 88.56}$	$-136.89_{\pm 85.41}$	+1.37	1.44
MC (SAC)	30696	23089	24.78	50000	10001	80.00	$-7.01_{\pm 0.09}$	$-7.09_{\pm 0.41}$	−1.14	4.94
MC (TD3)	38419	33590	12.56	100000	10004	90.00	$-6.13_{\pm 0.22}$	$-5.97_{\pm 0.07}$	+2.61	10.27
LLC (SAC)	315118	215108	31.73	500000	422642	15.47	$182.28_{\pm 17.50}$	$179.72_{\pm 16.05}$	−1.40	1.17
LLC (TD3)	290846	263927	9.25	200000	70026	65.00	$137.90_{\pm 51.81}$	$146.43_{\pm 26.90}$	+6.18	3.03
Swimmer (SAC)	232000	188000	18.96	500000	83922	83.21	$338.11_{\pm 1.95}$	$328.11_{\pm 1.98}$	−2.95	5.82
Swimmer (TD3)	216000	928000	–	1000000	61289	93.87	$47.45_{\pm 1.37}$	$114.86_{\pm 1.91}$	+142.06	46.75
Reacher (SAC)	183600	171600	6.53	300000	151743	49.41	$19.18_{\pm 10.70}$	$19.96_{\pm 10.23}$	+4.06	2.17
Reacher (TD3)	249600	243600	2.40	20000	20000	0	$13.45_{\pm 12.10}$	$17.10_{\pm 10.28}$	+27.13	1.40
Hopper (SAC)	725191	577319	20.39	1000000	290327	70.96	$3273.10_{\pm 5.98}$	$3529.13_{\pm 6.49}$	+7.82	3.72
Hopper (TD3)	998851	848361	15.06	1000000	320182	67.98	$3232.16_{\pm 246.55}$	$3269.26_{\pm 562.92}$	+1.14	3.16
Walker (SAC)	986412	779446	20.98	1000000	696091	30.39	$4332.53_{\pm 546.30}$	$5386.14_{\pm 24.75}$	+24.33	1.79
Walker (TD3)	784014	855112	–	1000000	755460	24.45	$4517.75_{\pm 9.86}$	$4875.14_{\pm 57.47}$	+7.91	1.43
Ant (SAC)	909250	782363	13.95	1000000	872183	12.78	$3800.34_{\pm 1289.42}$	$5057.42_{\pm 1022.72}$	+33.08	1.52
Ant (TD3)	784967	676999	13.75	1000000	754430	24.55	$5397.40_{\pm 400.47}$	$5416.09_{\pm 139.93}$	+0.34	1.33
Humanoid (SAC)	1819768	1079479	40.68	1000000	807923	19.20	$5706.61_{\pm 639.32}$	$5479.23_{\pm 786.85}$	−3.98	1.19
Humanoid (TD3)	1841538	1320449	28.29	1000000	436749	56.32	$5231.64_{\pm 695.39}$	$5217.76_{\pm 396.35}$	−0.26	2.29

till the end of learning. Here, r_{max} refers to the maximum reward attained during the training.

We define the percentage improvement in convergence (Δ^{CP}) achieved by FAC (CP_F) w.r.t. baselines (CP_B) as:

$$\Delta^{CP} = \frac{CP_B - CP_F}{CP_B} \times 100. \quad (11)$$

5.2.2 Size of Replay Buffer. The size of a replay buffer $|\mathcal{R}|$ is the number of transitions stored in it at the completion of learning. The replay buffer in the baseline setting is denoted by $\mathcal{R}_B$ and w.r.t the FAC Algorithm by $\mathcal{R}_F$. We measure the reduction in size of replay buffer $\Delta^{\mathcal{R}}$ achieved via FAC w.r.t. a baseline as:

$$\Delta^{\mathcal{R}} = \frac{|\mathcal{R}_B| - |\mathcal{R}_F|}{|\mathcal{R}_B|} \times 100. \quad (12)$$

5.2.3 Reward. The total reward accumulated by the synthesized control policy for the baseline is denoted by R_B and for the FAC Algorithm by R_F. The percentage change in total reward accumulated Δ^R by FAC algorithm w.r.t. baseline algorithm is defined as

$$\Delta^R = \frac{R_F - R_B}{R_B} \times 100. \quad (13)$$

5.2.4 Sample Efficiency. This metric captures the combined effect of reward improvement and the reduction in the size of the replay buffer. We denote the per-sample efficiency for FAC using P_F and that of baseline using P_B as $P_F = R_F/|\mathcal{R}_F|$ and $P_B = R_B/|\mathcal{R}_B|$.

Now we define a metric P to capture the relative per-sample efficiency achieved by FAC w.r.t baseline algorithm as

$$P = \frac{P_F}{P_B} = \frac{R_F \cdot |\mathcal{R}_B|}{|\mathcal{R}_F| \cdot R_B}. \quad (14)$$

Note that the above equation is meaningful if the rewards are positive. In the case of negative rewards, we first translate them into positive values by adding to them the sum of the absolute values of the reward in both cases. For instance, in the case of pendulum w.r.t. SAC (see Table 2), the addition term is abs(−143.97) + abs(−144.66) = 288.63. So, the translated reward for baseline (R_B) is 144.66, and for FAC is 143.97.

Table 3: Hyper-parameters used in the experiments

ν	ϵ	η	β	μ
0.5	0.2	$1e5$	0.2	50

5.3 Hyper-parameters

We run the experiments using the same configuration as the baselines [25], i.e., the state-of-the-art GAC algorithms SAC and TD3. In Equation 8, we use the Epanechnikov kernel [14] function. The values of other hyper-parameters used in experiments are shown in Table 3.

We consider three kernels for our density estimation, namely Gaussian, Tophat, and Epanechnikov. In general, the curve corresponding to the Gaussian kernel is too smooth, while that of the Tophat kernel is quite coarse. Thus, we select the Epanechnikov kernel as the curve is well balanced w.r.t. smoothness and coarseness. We perform experiments on three benchmark systems to judge the efficacy of the kernels. We observe that the Epanechnikov kernel indeed provides the best results consistently among these three kernels in terms of the accumulated rewards (refer [30] for the details).

The results of experiments for deciding the other hyper-parameters values are available in [30].

5.4 Results

In this subsection, we present our experimental results.

Convergence, Replay Buffer Size, and Accumulate Reward. The training for control policy synthesis (refer to the plots in [30]) is done using the same configuration as used in the stable-baselines3 [25]. Table 2 compares the convergence, size of the replay buffer, total accumulated reward, and sample efficiency of FAC with those of SAC and TD3 on nine benchmarks. From the Table, we can see the following trends:

- FAC converges faster than both SAC and TD3 on most of the benchmarks. In comparison to SAC and TD3, it converges up to 40% and 28% faster, respectively.
- FAC consistently requires a significantly smaller replay buffer compared to both SAC and TD3. It achieves up to 83% reduction in the size of the replay buffer in comparison to SAC and up to 94% reduction in comparison to TD3. For instance, the memory requirement of the replay buffer in the case of Humanoid is `5.7GB` for TD3 and `2.48GB` for FAC.
- For several benchmarks, FAC achieves significantly better accumulated reward. In the remaining cases, the degradation is marginal (up to only 4%).
- For all the benchmarks, FAC convincingly outperforms both SAC and TD3 in terms of sample efficiency.

There are a few exceptions to the general trends. For instance, for the simplest case of Pendulum in which the duration of control policy synthesis is small, we do not observe any improvement in convergence or the accumulated reward with FAC, but we observe a significant reduction in replay buffer size (upto 43%). In the case of Swimmer, the FAC requires significantly more steps to converge than TD3. However, no useful control policy could be synthesized for baseline TD3, as evidenced by low accumulated reward. On the other hand, the FAC extension of TD3 could synthesize a significantly superior controller with an improvement of 142% in the accumulated reward and almost 94% smaller replay buffer. Similarly, in the case of Walker w.r.t. baseline TD3, convergence was slightly delayed on account of improvement in the size of the replay buffer and the reward accumulation. In the case of Reacher, w.r.t. baseline TD3, there was no reduction in replay buffer size as the baseline replay buffer size was already very small. However, we get 27% improvement in reward accumulation and improved convergence.

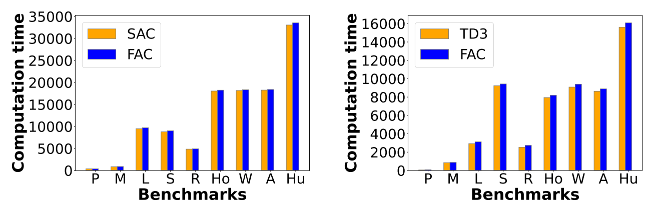
Control Cost. In general, we observe a decrease in control cost for FAC w.r.t. the baselines (refer Table 4). In some cases, there has been an increase in the control cost (shown in red), but it led to the synthesis of a superior control policy with more reward accumulation (see Table 2).

Computation Time. The computation time overhead during training due to the FAC algorithm is negligible compared to overall time (refer Figure 1).

Comparison with prioritization-based methods. A major approach to achieving sample efficiency in RL algorithms is sample prioritization. Thus, we compare our work with the most recently published prioritization algorithm LABER[17]. PER [28] is the most well-cited algorithm that introduced sample-prioritization. We did not include comparison results with PER as LABER has already been shown to be superior to PER.

Table 4: Comparison of control cost of the policy synthesized via FAC w.r.t SAC and TD3

Model	Baseline	FAC
Pendulum (SAC)	180.24 ± 32.33	72.49 ± 42.16
Pendulum (TD3)	135.46 ± 52.47	119.08 ± 48.45
MCC(SAC)	76.06 ± 17.51	70.99 ± 4.19
MCC(TD3)	61.32 ± 2.21	59.78 ± 0.79
LLC (SAC)	141.68 ± 105.20	113.64 ± 59.35
LLC (TD3)	257.93 ± 295.78	182.72 ± 110.07
Swimmer(SAC)	1394.31 ± 8.86	1264.12 ± 4.99
Swimmer(TD3)	1135.26 ± 224.99	1340.31 ± 22.94
Reacher(SAC)	13.20 ± 8.86	13.30 ± 8.88
Reacher(TD3)	69.51 ± 56.77	40.18 ± 32.89
Hopper(SAC)	969.93 ± 11.94	1025.52 ± 12.55
Hopper(TD3)	1390.49 ± 119.07	1325.62 ± 221.50
Walker(SAC)	2992.32 ± 328.23	3169.07 ± 16.37
Walker(TD3)	4901.46 ± 14.49	4592.26 ± 476.37
Ant(SAC)	2166.00 ± 602.51	2205.39 ± 401.49
Ant(TD3)	2327.36 ± 156.66	2186.27 ± 42.31
Humanoid(SAC)	1415.79 ± 146.36	1065.29 ± 121.39
Humanoid(TD3)	1953.06 ± 246.69	1807.19 ± 131.33



(a) FAC v/s SAC (b) FAC v/s TD3

Figure 1: Comparison of computation time (in seconds) required for training using FAC w.r.t. SAC and TD3.

We present the comparison of FAC with LABER in Table 5. We observe that FAC outperforms LABER in terms of both convergence and accumulated reward for most benchmarks while maintaining a significantly smaller size for the replay buffer (The replay buffer size $|\mathcal{R}|$ for LABER is the same as the baseline SAC/TD3).

6 RELATED WORK

The first use of experience replay in machine learning was in [18], where training was improved by repeating rare experiences. Of late, replay buffers have played a significant role in the breakthrough success of deep RL [12, 17, 19, 20]. An ideal replay strategy is expected to sample transitions in a way that maximizes learning by the agent. Two different types of methods have been proposed in the literature to achieve this. The first focuses on which experiences to add and keep in the replay buffer, and the second focuses on how to sample important experiences from the replay buffer.

There has been very little research in developing strategies to control the entries into the replay buffer. The authors in [5, 6]

Table 5: Comparison of Convergence (CP) and Reward (R) of FAC with baseline LABER w.r.t. SAC and TD3

Model	CP_L	CP_F	Δ^{CP}	R_L	R_F	Δ^R	P
Pendulum (SAC)	13600	17600	–	$-180.17_{\pm 209.52}$	$-144.66_{\pm 88.53}$	+24.54	2.18
Pendulum (TD3)	18400	18400	–	$-142.29_{\pm 87.62}$	$-136.89_{\pm 85.41}$	+3.94	1.47
MC (SAC)	29635	23089	22.09	$-7.07_{\pm 1.37}$	$-7.09_{\pm 0.41}$	−0.28	4.99
MC (TD3)	71131	33590	52.77	$-9.75_{\pm 0.05}$	$-5.97_{\pm 0.07}$	+63.31	16.33
LLC (SAC)	423928	215108	49.25	$183.49_{\pm 18.10}$	$179.72_{\pm 16.05}$	−2.05	1.16
LLC (TD3)	299600	263927	11.90	$139.37_{\pm 47.06}$	$146.43_{\pm 26.90}$	+5.06	3.00
Swimmer (SAC)	468000	188000	59.83	$1.25_{\pm 6.00}$	$328.11_{\pm 1.98}$	+26148.8	1563.88
Swimmer (TD3)	796000	928000	–	$34.36_{\pm 3.11}$	$114.86_{\pm 1.91}$	+234.28	54.54
Reacher (SAC)	172800	171600	0.69	$19.32_{\pm 10.17}$	$19.96_{\pm 10.23}$	+3.31	2.04
Reacher (TD3)	249600	243600	2.40	$13.33_{\pm 12.07}$	$17.10_{\pm 10.28}$	+28.28	2.54
Hopper (SAC)	952414	577319	39.38	$1844.07_{\pm 256.41}$	$3529.13_{\pm 6.49}$	+91.37	6.59
Hopper (TD3)	877102	848361	3.27	$3315.74_{\pm 82.07}$	$3269.26_{\pm 562.92}$	−1.40	3.08
Walker (SAC)	863761	779446	9.76	$4321.31_{\pm 843.80}$	$5386.14_{\pm 24.75}$	+24.64	1.79
Walker (TD3)	648102	855112	–	$3809.35_{\pm 36.37}$	$4875.14_{\pm 57.47}$	+27.97	1.69
Ant (SAC)	885086	782363	11.61	$3443.06_{\pm 770.65}$	$5057.42_{\pm 1022.72}$	+46.88	1.68
Ant (TD3)	924276	676999	26.75	$3082.34_{\pm 1518.43}$	$5416.09_{\pm 139.93}$	+75.71	2.33
Humanoid (SAC)	1822733	1079479	40.77	$4838.66_{\pm 1294.53}$	$5479.23_{\pm 786.85}$	+13.23	1.40
Humanoid (TD3)	1999903	1320449	33.97	$412.38_{\pm 30.78}$	$5217.76_{\pm 396.35}$	+1165.27	28.97

suggest that for simple continuous control tasks, the replay buffer should contain transitions that are not close to current policy to prevent getting trapped in local minima, and also, the best replay distribution is in between on-policy distribution and uniform distribution. Consequently, they maintain two buffers, one for on-policy data and another for uniform data, but this involves a significant computational overhead. This approach works in cases where policy is updated fewer times and hence is not suitable for complex tasks where policy is updated for many iterations. In [13], the authors show that a good replay strategy should retain the earlier samples as well as maximize the state-space coverage for lifelong learning. Hence, for stable learning, it is extremely crucial to identify which experiences to keep or discard from the replay buffer, such that sufficient coverage of the state-action space is enforced.

Most of the recent works have focused on sampling important transitions from the replay buffers to achieve sample efficiency. By default, the transitions in a replay buffer are uniformly sampled. However, many other alternatives have been proposed to prioritize samples during the learning process, most notably PER [27], which assigns TD errors as priorities to samples that are selected in a mini-batch. However, PER is computationally expensive and does not perform well in continuous environments, and leads to the synthesis of sub-optimal control policies [7, 8, 23, 24]. The issue of outdated priorities and hyper-parameter sensitivity of PER have been addressed in [16], where the authors propose importance sampling for estimating gradients. In [31], the authors propose re-weighting of experiences based on their likelihood under the stationary distribution of the current policy to minimize the approximation errors of the value function. The work in [22] is based on the idea that transitions collected on policy are more useful to train the current policy and hence enforce similarity between policy and transitions in the replay buffer. In [36], a replay policy is learned in addition to the agent policy. A replay policy helps in sampling useful experiences. All these methods are orthogonal to our approach, though all of them can potentially get benefited by using FAC to maintain unique samples in a replay buffer and reduce its size. Moreover, the prioritization based methods use some approximation of TD-errors as priority, which is problematic as it is biased towards outliers. FAC, on the other hand, uniformly samples from replay buffer and hence is not biased towards outliers.

7 CONCLUSION

We have proposed a novel off-policy actor-critic RL algorithm for control policy synthesis for model-free complex dynamical systems with continuous state and action spaces. The major benefit of the proposed method is that it stores unique samples in the replay buffer, thus reducing the size of the buffer and improving the IID characteristics of the samples. We provide a theoretical guarantee for its superior convergence with respect to the general off-policy actor-critic RL algorithms. We also demonstrate experimentally that our technique, in general, converges faster, leads to a significant reduction in the size of the replay buffer and superior accumulated rewards, and is more sample efficient compared to the state-of-the-art algorithms.

ACKNOWLEDGMENTS

The first author acknowledges the Prime Minister's Research Fellowship from the Govt. of India and the second author acknowledges the research grant MTR/2019/000257 from Science and Engineering Research Board of Govt. of India for supporting this research.

REFERENCES

[1] Léon Bottou and Olivier Bousquet. 2007. The Tradeoffs of Large Scale Learning. In *Advances in Neural Information Processing Systems*, J. Platt, D. Koller, Y. Singer, and S. Roweis (Eds.), Vol. 20. Curran Associates, Inc.

[2] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. 2018. Optimization Methods for Large-Scale Machine Learning. *SIAM Rev.* 60, 2 (2018), 223–311.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. *CoRR* abs/1606.01540 (2016). arXiv:1606.01540 http://arxiv.org/abs/1606.01540

[4] Tony F. Chan. 1987. Rank revealing QR factorizations. *Linear Algebra Appl.* 88-89 (1987), 67–82.

[5] Tim de Bruin, Jens Kober, K.P. Tuyls, and Robert Babuska. 2015. The importance of experience replay database composition in deep reinforcement learning. In *Deep Reinforcement Learning Workshop, NIPS 2015.*

[6] Tim de Bruin, Jens Kober, Karl Tuyls, and Robert Babuska. 2016. Improved deep reinforcement learning for robotics through distribution-based experience retention. *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2016), 3947–3952.

[7] Justin Fu, Aviral Kumar, Matthew Soh, and Sergey Levine. 2019. Diagnosing Bottlenecks in Deep Q-learning Algorithms. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*. PMLR, 2021–2030.

[8] Scott Fujimoto, David Meger, and Doina Precup. 2020. An Equivalence between Loss Functions and Non-Uniform Sampling in Experience Replay. In *Advances in Neural Information Processing Systems*, Vol. 33. Curran Associates, Inc., 14219–14230.

[9] Scott Fujimoto, Herke van Hoof, and David Meger. 2018. Addressing Function Approximation Error in Actor-Critic Methods. In *Proceedings of the 35th International Conference on Machine Learning*, Vol. 80. 1587–1596.

[10] Geoffrey J. Gordon. 1999. Regret Bounds for Prediction Problems. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory* (Santa Cruz, California, USA) *(COLT '99)*. Association for Computing Machinery, New York, NY, USA, 29–40.

[11] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *ICML*. 1861–1870.

[12] Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-Learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence* (Phoenix, Arizona) *(AAAI'16)*. AAAI Press, 2094–2100.

[13] David Isele and Akansel Cosgun. 2018. Selective Experience Replay for Lifelong Learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence* (New Orleans, Louisiana, USA). AAAI Press.

[14] M. C. Jones. 1990. The performance of kernel density functions in kernel distribution function estimation. *Statistics and Probability Letters* 9, 2 (1990), 129–132.

[15] Vijay R. Konda and John N. Tsitsiklis. 2003. OnActor-Critic Algorithms. *SIAM Journal on Control and Optimization* 42, 4 (2003), 1143–1166.

[16] Thibault Lahire, Matthieu Geist, and Emmanuel Rachelson. 2022. Large Batch Experience Replay. In *Proceedings of the 39th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 162)*. 11790–11813.

[17] Timothy et. al. Lillicrap. 2016. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings.*

[18] Longxin Lin. 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning* 8 (1992), 293–321.

[19] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. (2013). http://arxiv.org/abs/1312.5602 arxiv:1312.5602 : NIPS Deep Learning Workshop 2013.

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Kirkeby Fidjeland, Georg Ostrovski, Stig Petersen, Charlie Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518 (2015), 529–533.

[21] Deanna Needell, Rachel Ward, and Nati Srebro. 2014. Stochastic Gradient Descent, Weighted Sampling, and the Randomized Kaczmarz algorithm. In *Advances in Neural Information Processing Systems*, Vol. 27. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2014/file/f29c21d4897f78948b91f03172341b7b-Paper.pdf

[22] Guido Novati and Petros Koumoutsakos. 2019. Remember and Forget for Experience Replay. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). 4851–4860.

[23] Youngmin Oh, Kimin Lee, Jinwoo Shin, Eunho Yang, and Sung Ju Hwang. 2021. Learning to Sample with Local and Global Contexts in Experience Replay Buffer. In *International Conference on Learning Representations.*

[24] Youngmin Oh, Jinwoo Shin, Eunho Yang, and Sung Ju Hwang. 2022. Model-augmented Prioritized Experience Replay. In *International Conference on Learning Representations.*

[25] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. 2021. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research* 22, 268 (2021), 1–8.

[26] Siva Satyendra Sahoo, Akhil Raj Baranwal, Salim Ullah, and Akash Kumar. 2021. MemOReL: A Memory-Oriented Optimization Approach to Reinforcement Learning on FPGA-Based Embedded Systems. In *Proceedings of the 2021 on Great Lakes Symposium on VLSI* (Virtual Event, USA) *(GLSVLSI '21)*. Association for Computing Machinery, New York, NY, USA, 339–346.

[27] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2016. Prioritized Experience Replay. In *4th International Conference on Learning Representations, ICLR, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings.*

[28] Shai Shalev-Shwartz. 2012. Online Learning and Online Convex Optimization. *Found. Trends Mach. Learn.* 4, 2 (feb 2012), 107–194.

[29] C. E. Shannon. 1948. A mathematical theory of communication. *The Bell System Technical Journal* 27, 3 (1948), 379–423.

[30] Nikhil Kumar Singh and Indranil Saha. 2024. Frugal Actor-Critic: Sample Efficient Off-Policy Deep Reinforcement Learning Using Unique Experiences. arXiv:2402.05963

[31] Samarth Sinha, Jiaming Song, Animesh Garg, and Stefano Ermon. 2022. Experience Replay with Likelihood-free Importance Weights. In *Proceedings of The 4th Annual Learning for Dynamics and Control Conference (Proceedings of Machine Learning Research, Vol. 168)*. 110–123.

[32] Camélia Slimani, Stéphane Rubini, and Jalil Boukhobza. 2019. K -MLIO: Enabling K -Means for Large Data-Sets and Memory Constrained Embedded Systems. In *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 262–268.

[33] George R. Terrell and David W. Scott. 1992. Variable Kernel Density Estimation. *The Annals of Statistics* 20, 3 (1992), 1236 – 1265.

[34] Mnih et. al. Volodymyr. 2015. Human-level control through deep reinforcement learning. *Nature* 518 (2015), 529–533.

[35] Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal component analysis. *Chemometrics and Intelligent Laboratory Systems* 2, 1 (1987), 37–52. Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists.

[36] Daochen Zha, Kwei-Herng Lai, Kaixiong Zhou, and Xia Hu. 2019. Experience Replay Optimization. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence* (Macao, China) *(IJCAI'19)*. AAAI Press, 4243–4249.

[37] Peilin Zhao and Tong Zhang. 2015. Stochastic Optimization with Importance Sampling for Regularized Loss Minimization. In *Proceedings of the 32nd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 37)*. Lille, France, 1–9.