The Rank and Gradient Lost in Non-stationarity: Sample Weight Decay for Mitigating Plasticity Loss in Reinforcement Learning

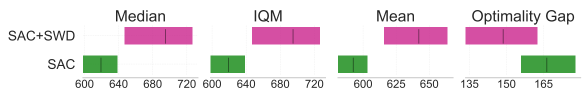
Anonymous authors
Paper under double-blind review

Abstract

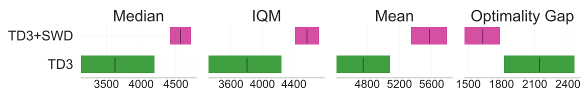
Deep reinforcement learning (RL) suffers from plasticity loss severely due to the nature of non-stationarity, which impairs the ability to adapt to new data and learn continually. Unfortunately, our understanding of how plasticity loss arises, dissipates, and can be dissolved remains limited to empirical findings, leaving the theoretical end underexplored. To address this gap, we study the plasticity loss problem from the theoretical perspective of network optimization. By formally characterizing the two culprit factors in online RL process: the non-stationarity of data distributions and the non-stationarity of targets induced by bootstrapping, our theory attributes the loss of plasticity to two mechanisms: the rank collapse of the Neural Tangent Kernel (NTK) Gram matrix and the $\Theta(\frac{1}{k})$ decay of gradient magnitude. The first mechanism echoes prior empirical findings from the theoretical perspective and sheds light on the effects of existing methods, e.g., network reset, neuron recycle, and noise injection. Against this backdrop, we focus primarily on the second mechanism and aim to alleviate plasticity loss by addressing the gradient attenuation issue, which is orthogonal to existing methods. We propose Sample Weight Decay (`SWD`) — a lightweight method to restore gradient magnitude, as a general remedy to plasticity loss for deep RL methods based on experience replay. In experiments, we evaluate the efficacy of `SWD` upon TD3, Double DQN and SAC with SimBa architecture in MuJoCo, ALE and DeepMind Control Suite tasks. The results demonstrate that `SWD` effectively alleviates plasticity loss and consistently improves learning performance across various configurations of deep RL algorithms, UTD, network architectures, and environments, achieving SOTA performance on challenging DMC Humanoid tasks.

1 Introduction

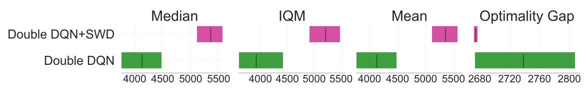
Deep reinforcement learning (RL) has achieved remarkable success across a variety of domains, including robotics (Akkaya et al., 2019), game playing (Berner et al., 2019) and LLM post-training that endows language models with the ability to generate human-like replies for breaking the Turing test (Biever, 2023). The core driver behind these advancements of deep RL lies in the combination of RL and deep neural networks. With the powerful expressive capacity and adaptive learning ability, the neural networks can effectively approximate and optimize value functions and policies under the RL training regime. However, recent studies have identified a



(a) SWD for SimBa-SAC in DMC tasks



(b) SWD for TD3 in MuJoCo tasks



(c) SWD for Double DQN in ALE tasks

Figure 1: Aggregate *Reliable* metrics (Agarwal et al., 2021) with 95% Stratified Bootstrap CIS.

critical yet often overlooked challenge — *Plasticity Loss*: as training progresses, the learning ability of neural networks gradually diminishes (Elsayed & Mahmood, 2024; Nikishin et al., 2022). To address this phenomenon, researchers in the RL community have proposed different metrics and remedies mainly from empirical perspectives, such as Network Reset (Nikishin et al., 2022), Neuron Recycling (Sokar et al., 2023), Noise Injection (Nikishin et al., 2023a). However, these existing works all rely on empirical intuitions and lack clear theoretical grounding, leaving a significant gap between empiricism and theory. Despite the significance of this issue, explaining plasticity from the theoretical perspective and developing principled algorithms remain highly challenging due to the complexity of the underlying mechanisms of plasticity loss in the context of deep RL.

To analyze the optimization dynamics of Reinforcement Learning (RL) agents, we develop a structured theoretical framework rooted in a core insight: due to the dynamic nature of the optimization process in RL, the loss function evolves with each optimization iteration—effectively initiating a new optimization ”task” in each round. Critically, the initial optimization point for the updated loss function in the current round is exactly the terminal point from optimizing the previous round’s loss function. This sequential initialization mechanism raises fundamental questions about its potential adverse impacts on optimization performance, and this line of inquiry underpins the entire logic of our theoretical analysis. Based on this insight, we arrive at a key conclusion: RL agents inherently confront two critical challenges that exert profound adverse effects on loss function optimization.The first is the potential rank deficiency of the Neural Tangent Kernel (NTK) (Jacot et al., 2018)—a core factor that governs the network’s fitting capacity, specifically its ability to approximate the optimal value function in RL.The second is a gradient magnitude decay , which directly regulates the neural network’s fitting rate and dictates the time required to escape saddle points.

Our theoretical results reveal two causal mechanisms for the occurrence of plasticity loss. The first mechanism echoes prior empirical findings from the theoretical perspective and sheds light on the effects of existing methods. Differently, we focus primarily on the second mechanism, which has not been well explored, and aim to alleviate plasticity loss by addressing the gradient attenuation issue from an orthogonal angle to existing methods. In this paper, we design an anti-decay sampling strategy as a compensation measure. We observe that gradient decay is governed by the linearly decaying term $\frac{1}{k}$, where k represents the number of learning iteration. In response to this, we construct a set of linearly weighted coefficients, where the sampling probability decreases linearly with the *age* of the samples. Specifically, we propose **Sample Weight Decay (`SWD`)** — a lightweight method tailored to mitigate plasticity loss in deep Reinforcement Learning (RL) algorithms. `SWD` effectively maintains the gradient magnitude at a appropriate scale, ensuring stable learning dynamics.

Building on the SimBa-SAC (Lee et al., 2025a; Haarnoja et al., 2018) , TD3 (Fujimoto et al., 2018) and Double DQN (Hasselt et al., 2016) algorithms as base algorithm, `SWD` significantly enhances learning stability and performance in continuous control tasks and pixel-based tasks. To validate its effectiveness, we evaluated `SWD` across three well-established online reinforcement learning (RL) benchmarks: the MuJoCo (Brockman, 2016) , Arcade Learning Environment (Bellemare et al., 2013) and the DeepMind Control (DMC) Suite (Tassa et al., 2018). For our evaluation protocol, we adopted the Interquartile Mean (IQM) as the core performance metric, while leveraging `GraMa` (Liu et al., 2025) as the key indicator to quantify plasticity. As illustrated in Figure 1, `SWD` consistently delivers state-of-the-art (SOTA) performance.

The contributions of this paper are summarized as follows:

- We have developed a unified theory to account for plasticity in deep reinforcement learning (RL), thereby shedding clear light on the origins of such plasticity, bridging the gap between empirical practice and theoretical research.

- We propose `SWD`, a theoretically grounded plug-and-play method to different RL algorithms for mitigating plasticity loss and improving learning performance.

- The experiments demonstrate the efficacy of `SWD` in improving learning stability and performance. Additionally, `SWD` achieves state-of-the-art (SOTA) performance in challenging DMC Humanoid tasks.

2 Related Work

Plasticity loss refers to the phenomenon in neural network training where the model gradually loses its ability to adapt to new data, objectives, or tasks during the learning process (Dohare et al., 2024). This usually reflects that the network becomes overly specialized to the early stages of training, resulting in reduced learning capacity, slower convergence, or even a collapse in later stages of training (Nikishin et al., 2022; 2023a). To gain a better understanding of plasticity loss and address it effectively, many efforts have been made to conduct various empirical investigations and propose different solutions (Ash & Adams, 2020; Lewandowski et al., 2023; Kumar et al., 2023; Ceron et al., 2023; Asadi et al., 2023; Ellis et al., 2024; Chung et al., 2024; Tang & Berseth, 2024; Frati et al., 2024; Ceron et al., 2024).

Sokar et al. (2023) first identified the *dormant neuron phenomenon* in deep reinforcement learning (RL) networks, where neurons progressively fall into an inactive state and their expressive capacity diminishes over the course of training. To address this issue, they proposed Recycle Dormant neurons (ReDo) — a strategy that continuously detects and recycles dormant neurons throughout the training process. In a separate line of work, Nikishin et al. (2023a) proposed Plasticity Injection, a minimal-intervention technique that boosts network plasticity without altering trainable parameters or introducing biases into predictive outputs. More recently, Liu et al. (2025) introduced Reset guided by Gradient Magnitude (ReGraMa), which addresses neuronal activity loss in deep RL agents by transitioning from activation statistics to gradient-based neuron reset strategies, maintaining network plasticity through `GraMa` metrics. While these approaches have empirically validated their effectiveness in combating plasticity loss, they predominantly operate at the model level — modifying network architectures without addressing the fundamental theoretical questions: *why* plasticity loss occurs and *how* different underlying mechanisms contribute to this phenomenon. This presents a significant gap between empiricism and theory.

This theoretical gap motivates our work, which targets the fundamental gradient decay mechanism identified through our theoretical analysis. Our proposed Sample Weight Decay (`SWD`) approach operates at the strategic level — focused on weighting in experience replay — and provides a principled means of compensating for the $\Theta(1/k)$ gradient attenuation, a challenge unaddressed by recent techniques. A key distinguishing feature of `SWD` is its *orthogonality* to existing methods: whereas prior approaches modify network structures or plasticity injection patterns, `SWD` acts at the data distribution level via intelligent experience reweighting, ensuring compatibility with existing plasticity-preserving techniques and enabling synergistic performance improvements.

3 Preliminaries

We consider an episodic Markov Decision Process (MDP) $(\mathbb{S}, \mathbb{A}, H, \{P_h\}_{h=1}^H, \{r_h\}_{h=1}^H)$ with horizon $H \in \mathbb{Z}^+$ (Puterman, 2014). Here, $\mathbb{S}, \mathbb{A}$ are measurable state, action spaces; $P_h(\cdot \mid s, a)$ is the transition kernel at step h ; $r_h : \mathbb{S} \times \mathbb{A} \to [0, 1]$ is the reward at step h. At each episode, an initial state x_1 is drawn. At step $h \in [H]$, the agent observes $x_h \in \mathbb{S}$, chooses $a_h \in \mathbb{A}$, receives $r_h(x_h, a_h)$, and transits to $x_{h+1} \sim P_h(\cdot \mid x_h, a_h)$. A policy is $\pi = \{\pi_h\}_{h=1}^H$ with $\pi_h(\cdot \mid x)$.

For policy π, the value and action-value functions are defined as:

$$V_h^\pi(x) = \mathbb{E}\left[\sum_{t=h}^{H} r_t(x_t, a_t) \,\middle|\, x_h = x,\ a_t \sim \pi_t(\cdot \mid x_t)\right], \qquad \forall x \in \mathbb{S},\ h \in [H],$$
$$Q_h^\pi(x, a) = r_h(x, a) + \mathbb{E}_{x' \sim P_h(\cdot|x,a)}\big[V_{h+1}^\pi(x')\big], \qquad \forall (x, a) \in \mathbb{S} \times \mathbb{A},\ h \in [H],$$

with terminal condition $V_{H+1}^\pi \equiv 0$. It is convenient to write the transition expectation operator $\mathbb{P}_h$ and policy expectation operator $\mathbb{J}_h^\pi$:

$$(\mathbb{P}_h V)(x, a) = \mathbb{E}_{x' \sim P_h(\cdot|x,a)}[V(x')], \quad (\mathbb{J}_h^\pi Q)(x) = \mathbb{E}_{a \sim \pi_h(\cdot|x)}[Q(x, a)].$$

Then the policy Bellman equations compactly read,

$$Q_h^\pi(x, a) = r_h(x, a) + (\mathbb{P}_h V_{h+1}^\pi)(x, a),$$
$$V_h^\pi(x) = (\mathbb{J}_h^\pi Q_h^\pi)(x), \qquad V_{H+1}^\pi \equiv 0.$$

For any function $g : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$, define the value maximization operator $\mathbb{V}$ and the step-h optimality Bellman operator $\mathcal{T}_h$ by

$$\mathbb{V}_g(x) := \max_a g(x,a)$$
$$(\mathcal{T}_h g)(x,a) := r_h(x,a) + (\mathbb{P}_h \mathbb{V}_g)(x,a).$$

4 Theory Analysis: The Rank Loss and Gradient Attenuation

In this section, our primary objective is to establish a rigorous connection between the optimization process and plasticity loss. To this end, we first utilize Equation 3 to derive a formal bound on the model's performance. We then simplify the dynamic optimization process by reducing it to an initialization problem—a key step that streamlines subsequent analyses. Finally, we elaborate on the derivation of our core results, with the full details presented in Section 4.1 and Section 4.2.

For the sake of clarity and analytical tractability, we focus our discussion on the simplest variant of Fitted Q-Iteration (FQI) (Ernst et al., 2005). Importantly, the theoretical framework proposed herein is not limited to this specific algorithm; it can be readily extended to accommodate a wider class of value-based reinforcement learning methods. Of note, analogous analytical findings hold for entropy-regularized Markov Decision Processes (MDPs). A comprehensive treatment of this extension, including detailed proofs and supplementary analyses, is provided in Appendix B.4.

Let $\mathcal{D}_h^k$ denote the replay buffer at step h following k episodes, and let $\hat{f}_{h+1}^k$ represent the estimated Q-value at step $h+1$ after k episodes. The loss function is then defined as follows:

$$\mathcal{L}_h^k(f, \hat{f}_{h+1}^k) := \frac{1}{|\mathcal{D}_h^k|} \sum_{(s_h, a_h, s_{h+1}) \sim \mathcal{D}_h^k} \left[\left(f(s_h, a_h) - \left(r(s_h, a_h) + \max_{a'} \hat{f}_{h+1}^k(s_{h+1}, a') \right) \right)^2 \right]$$
$$\hat{f}_h^k = \arg\min_{f \in \mathcal{F}} \mathcal{L}_h^k(f, \hat{f}_{h+1}^k), \qquad \hat{f}_{H+1} \equiv 0$$

Define the empirical distribution μ_h^k of the replay buffer over (s,a) and the empirical state-action visitation frequency of the behavior policy π^{k+1} at time h in episode k:

$$\mu_h^k(s,a) := \frac{1}{|\mathcal{D}_h^k|} \sum_{(s_i, a_i, s_i') \in \mathcal{D}_h^k} \mathbb{I}\{s = s_i,\ a = a_i\},$$
$$\hat{d}_h^{\pi^{k+1}}(s,a) := \mathbb{I}\{s = s_h^{k+1}, , a = a_h^{k+1}\}, \qquad (s_h^{k+1}, a_h^{k+1}) \sim \mathbb{P}_h^{\pi^{k+1}}(s,a)$$

To establish a mathematical formulation for distribution shift and thereby quantify its impact on the loss function, we rely on Proposition 1 to characterize such distributional non-stationarity. Furthermore, to facilitate the subsequent gradient decomposition, we express the loss function in the form specified in Theorem 1. Finally, to connect the agent's performance to the loss function, we leverage Theorem 2 to provide a bound on the agent's final performance.

Proposition 1 (Empirical distribution recursion). *The empirical distribution satisfies*

$$\mu_h^{k+1} = \frac{k}{k+1} \mu_h^k + \frac{1}{k+1} \hat{d}_h^{\pi^{k+1}}. \tag{1}$$

Proof (sketch). By construction, $|\mathcal{D}_h^{k+1}| = k+1$ and $\mathcal{D}_h^{k+1} = \mathcal{D}_h^k \cup \{(s_h^{k+1}, a_h^{k+1}, s_{h+1}^{k+1})\}$. Expanding the definition of μ_h^{k+1} and regrouping terms yields the stated convex combination. □

Theorem 1 (Population loss limit). *Let $\mathcal{F}$ be a measurable function class. As the cardinality (or appropriate size measure) of $\mathcal{D}_h^k$ tends to infinity (i.e., $|\mathcal{D}_h^k| \to \infty$), the following probabilistic convergence holds:*

$$\mathcal{L}_h^k(f, \hat{f}_{h+1}^k) \xrightarrow{p} \mathbb{E}_{(s_h, a_h) \sim \mu_h^k} \left[\left(f(s_h, a_h) - (\mathcal{T}_h \hat{f}_{h+1}^k)(s_h, a_h) \right)^2 \right] + C_h^k \tag{2}$$

where C_h^k is a constant independent of f. Henceforth, we do not rigorously distinguish between the empirical risk and the expected loss, focusing instead on the underlying optimization problem.

Takeaway 1. The non-stationarity in training process

The population loss limit established in Theorem 1 identifies two key sources of non-stationarity in the training process of Fitted Q-Iteration: the non-stationary distribution μ_h^k and the non-stationary target $\mathcal{T}_h \hat{f}_{h+1}^k$. Both of these sources drive variations in the target population risk across training episodes k.

Theorem 2 (Suboptimality bound via squared bellman residuals)**.** *Fix horizon H. Let $\{\hat{f}_h\}_{h=1}^H$ denote the final value estimates (e.g., from the K-th iteration; write $\hat{f}_h := \hat{f}_h^{(K)}$). Define the greedy policy*

$$\pi_{\hat{f},h}(s) \in \arg\max_{a\in\mathbb{A}} \hat{f}_h(s,a), \qquad h = 1,\ldots,H.$$

For functions $f, g : \mathbb{S}\times\mathbb{A} \to \mathbb{R}$, define the step-$h$ squared Bellman residual

$$\Delta_h(f,g)(s,a) \;:=\; \big(f(s,a) - (\mathcal{T}_h g)(s,a)\big)^2$$

Then for any start state x,

$$V_1^*(x) - V_1^{\pi_{\hat{f}}}(x) \le \sqrt{H}\left(\sqrt{\mathbb{E}_{\pi^*}\left[\sum_{h=1}^{H}\Delta_h(\hat{f}_h,\hat{f}_{h+1})(s_h,a_h)\,\middle|\, s_1 = x\right]} + \sqrt{\mathbb{E}_{\pi_{\hat{f}}}\left[\sum_{h=1}^{H}\Delta_h(\hat{f}_h,\hat{f}_{h+1})(s_h,a_h)\,\middle|\, s_1 = x\right]}\right). \tag{3}$$

Takeaway 2. Suboptimality bound

Equation 3 links model performance to the loss function for optimization. This means the agent's performance depends on Bellman residuals from the current and optimal policy trajectories, not historical ones.

Building on the foundational framework established above, we can observe that the loss function evolves at each iteration k; this phenomenon is analogous to initiating an entirely new round of training. A key distinction from supervised learning lies in the initialization process: whereas supervised learning relies on **random initialization**, reinforcement learning (RL) commences optimization from the $\arg\min$ of the loss function obtained in the previous iteration. Consequently, investigating the properties of initialization points under such non-steady-state conditions becomes particularly crucial—an insight that further underscores the necessity of the theoretical exploration presented in our work.

4.1 Neural Tangent Kernel (NTK) Degeneration

A key advantage of random initialization is that it ensures the Neural Tangent Kernel (NTK) matrix of an overparameterized neural network is **full-rank with probability 1**. This comes from the property that low-dimensional manifolds have zero measure in high-dimensional spaces, making the NTK matrix rank-deficient extremely unlikely. However, this random initialization is violated in Reinforcement Learning (RL), so the initial NTK matrix's structural properties (e.g., rank, conditioning) are no longer guaranteed, adding uncertainty to learning.

Prior research (Du et al., 2019; Allen-Zhu et al., 2019) shows overparameterized networks achieve global convergence to zero training error via Gradient Descent (GD) or Stochastic Gradient Descent (SGD) if two conditions hold: *(i)* The initial NTK matrix $\mathbf{K}_0$ is well-conditioned: its eigenvalues are bounded (no extremely large/small values to distort optimization). *(ii)* The NTK matrix $\mathbf{K}$ remains stable during training, so its structural properties do not degrade and harm convergence.

4.2 Gradient Attenuation

Another advantage of proper random initialization lies in preserving an appropriate initial gradient magnitude—a factor whose critical role in the optimization process has been well validated through both experimental observations and theoretical analyses. As illustrated in (Dixit et al., 2023), the time an optimizer needs to escape a saddle point is dictated by the magnitude of the initial state's projection onto the unstable (negative-curvature) subspace of the saddle point. In consequence, excessively small initial gradients prolong the optimizer's stagnation near saddle points, resulting in a significant deterioration in optimization performance. Unfortunately, however, gradient decay is an inherent and unavoidable phenomenon in reinforcement learning (RL) training, as evidenced by the theorem presented below.

Theorem 3 (Gradient Dynamics at Initialization)**.** *For the optimization objective defined in Equation 1, the initial gradient of the loss function (evaluated at the parameter values that minimized the loss of the previous iteration, $\hat{f}_h^{k-1}$) satisfies:*

$$\nabla \mathbb{E}_{\mu_h^k}\left[\left(f - \mathcal{T}_h \hat{f}_{h+1}^k\right)^2\right]\Big|_{\hat{f}_h^{k-1}} = \underbrace{\frac{1}{k}\,\nabla \mathbb{E}_{\hat{d}_h^{\pi^k}}\left[\left(f - \mathcal{T}_h \hat{f}_{h+1}^{k-1}\right)^2\right]\Big|_{\hat{f}_h^{k-1}}}_{\textit{Distributional shift}} + \underbrace{\mathbb{E}_{\mu_h^k}\left[\nabla f^2\big|_{\hat{f}_h^{k-1}} \cdot \left(\mathcal{T}_h \hat{f}_{h+1}^{k-1} - \mathcal{T}_h \hat{f}_{h+1}^k\right)\right]}_{\textit{Target drift}} \tag{4}$$

By setting $\hat{f}_{H+1} \equiv 0$. This eliminates the target-drift term entirely, leaving only the distributional-shift component—where the $\Theta(1/k)$ scaling factor becomes the dominant driver of gradient decay. As the number of training iterations k grows large, the magnitude of the initial gradient will tend to approach zero. This near-zero gradient signal risks trapping the optimization process at saddle points, as the model lacks sufficient directional information to escape these suboptimal regions.

5 Sample Weight Decay (SWD)

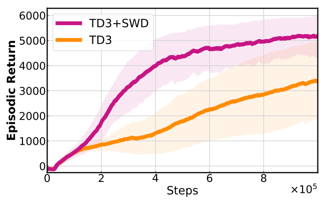
Algorithm 1 Sample Weight Decay (SWD)

```
Require: Linear decay steps $T$, minimum weight $w_{\min}$, Current time $t$, timestamps $\{t_i\}_{i=1}^{|\mathcal{D}|}$
1: for $i = 1$ to $|\mathcal{D}|$ do
2:     $\text{age}_i = t - t_i$
3:     $w_i = \max\left(w_{\min}, 1 - \frac{age_i}{T}\right)$
4: end for
5: $p_i = \frac{w_i}{\sum_{j=1}^{|\mathcal{D}|} w_j}$ for $i = 1, \ldots, |\mathcal{D}|$
6: $\mathcal{I} \sim \text{Categorical}(\{p_i\}_{i=1}^{|\mathcal{D}|}, B)$
7: return $\mathcal{B} = \{(s_i, a_i, r_i, s_i', d_i)\}_{i \in \mathcal{I}}$
```

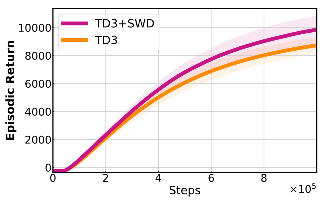
SWD is a principled algorithmic intervention to mitigate gradient signal degradation in non-stationary reinforcement learning environments. As in Algorithm 1, it addresses the core challenge in Theorem 3: the harmful $\frac{1}{k}$ decay of gradient contributions from new data. It uses a linear decay mechanism, assigning each sample a weight $w_i = \max(w_{\min}, 1 - \frac{\text{age}_i}{T})$, where $\text{age}_i = t - t_i$ (t = current training step, t_i = sample collection step). The key insight of Algorithm 1 is its rigorous sample weighting. It identifies the $\frac{1}{k}$ coefficient—overly attenuating gradients from the current policy distribution $\hat{d}^{\pi^k}$—as the root of gradient degradation. To counter this, SWD introduces a linear weighting scheme: each sample gets a probability $p_i = \frac{w_i}{\sum_{j=1}^{|D|} w_j}$, with p_i proportional to sample recency. This neutralizes the $\frac{1}{k}$ attenuation, restoring gradient magnitude and sustaining model plasticity during training.

6 Experiments

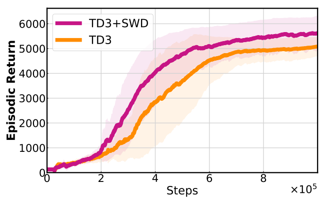
The core objective of the experiment is to validate the efficacy of the proposed SWD method in mitigating plasticity loss during long-horizon training and quantify its performance advantages with multifaceted analyses. Specifically, we focus on the following five key research questions:



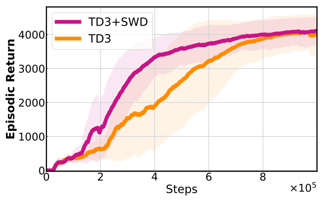
(a) Ant



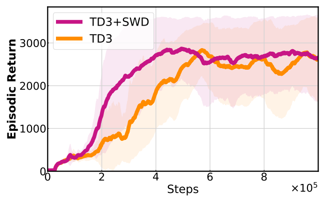
(b) Halfcheetah



(c) Humanoid



(d) Walker2d



(e) Hopper

Figure 2: Empirical validation of `SWD` across TD3 in MuJoCo environments (mean $\pm$ std over 5 runs). `SWD` consistently improves sample efficiency and performance.

- **Q1**: Does the proposed method `SWD` consistently improve the training performance of mainstream reinforcement learning (RL) algorithms across different continuous and discrete control tasks?
- **Q2**: Does the temporal weighting strategy of the proposed method `SWD` play a critical role in alleviating plasticity loss?
- **Q3**: Can `SWD` adapt to the training scenarios with increased Update-to-Data (UTD) ratio configurations, where more severe plasticity loss should be addressed for better data efficiency?
- **Q4**: How does `SWD` compare with other methods designed to address plasticity issues? And is it feasible to combine `SWD` with these other methods?
- **Q5**: How sensitive is the proposed `SWD` to the hyperparameters? How do different choices of heuristics influence the results?

To address Q1, we conduct experiments using the Double DQN, TD3, and SAC algorithms within the SimBa architecture (Lee et al., 2025a), evaluating their performance across the Arcade Learning Environment (Bellemare et al., 2013), the MuJoCo environments (Brockman, 2016), and the DMC suite (Tassa et al., 2018). We also include the canonical method Prioritized Experience Replay (PER) (Schaul et al., 2016) as a direct baseline method. Furthermore, to provide reverse validation of `SWD`'s effectiveness, we use a variant called Sample Weight Augmentation (`SWA`), i.e., a counterpart designed to produce the opposite effect by assigning higher weights to older samples. For Q2, we adopt `GraMa` (Liu et al., 2025) as the metric for plasticity, using it to empirically demonstrate the superiority of our proposed method in alleviating plasticity loss. To answer Q3, we evaluate the performance of `SWD` based on Simba-SAC under different UTD ratios, with a specific focus on the Humanoid Run environment. To address Q4, we compare `SWD` against other representative methods designed to address plasticity issues. For Q5, we conduct extensive experiments to analyze the hyperparameter sensitivity of `SWD` and the effects of different decay strategies, such as exponential decay and polynomial decay.

6.1 Performance Evaluation

Experimental Setup. We evaluate methods on three benchmark suites: *(i)* For the five MuJoCo environments (Ant, HalfCheetah, Hopper, Humanoid, Walker2d), we use TD3 (Fujimoto et al., 2018) as the base algorithm with conventional MLP networks. *(ii)* For the three ALE environments (DemonAttack, Phoenix, and Breakout), we use Double Deep Q-Network (Hasselt et al., 2016) as the base algorithm with the typical CNN-MLP networks. *(iii)* For the four difficult DMC tasks (Humanoid-Run, Humanoid-Walk, Dog-Run, Dog-Walk),we use SAC (Haarnoja et al., 2018) as the base algorithm with the SimBa network architecture (Lee et al., 2025a). In this subsection,

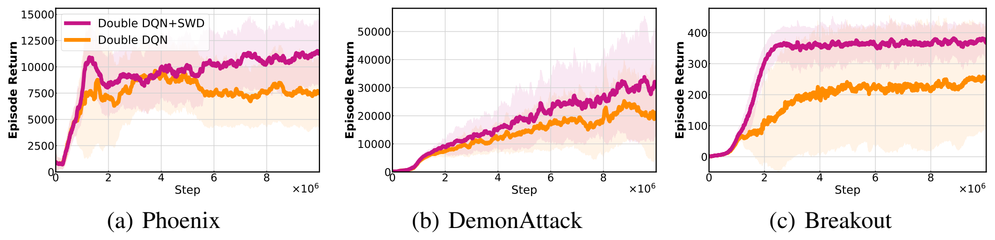


Figure 3: Empirical validation of SWD across Double DQN in ALE environments (mean ± std over 5 runs). SWD consistently improves sample efficiency and performance.

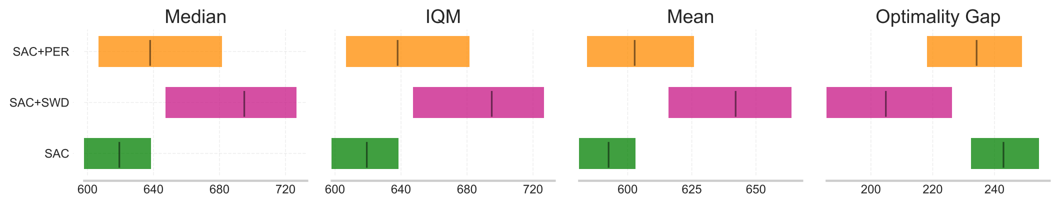


Figure 4: Performance comparison between SWD and PER based on SAC. Aggregate *Reliable* metrics (Agarwal et al., 2021) with 95% Stratified Bootstrap CIS in DMC tasks.

we include PER as a canonical baseline for comparison. Detailed hyperparameters and details are provided in Appendix C.

Results As illustrated in Figure 2, Figure 3 and Figure 4, SWD demonstrates a remarkable ability to enhance the algorithm's performance. Specifically, it facilitates accelerated learning during the early phases of training and attains superior final policy quality upon convergence—an advantage that is particularly prominent in the Ant and Humanoid environments. In sharp contrast, PER (Prioritized Experience Replay) demands nearly several times more training time, while the performance improvements it yields remain extremely limited. This observation aligns well with our theoretical framework, and Equation 3 further confirms that performance enhancement can only be achieved by optimizing the TD errors along both the optimal policy path and the current policy path.

6.2 Ablation Study

To provide reverse validation of SWD's effectiveness, we develop a contrasting method calledSample Weight Augmentation (SWA), which implements the opposite weighting strategy by assigning higher weights to older data samples. This design allows us to empirically verify our theoretical hypothesis that prioritizing recent experiences is crucial for maintaining neural plasticity.More details are shown in Appendix F, where we employed GraMa (Liu et al., 2025) as our measure of neural plasticity.

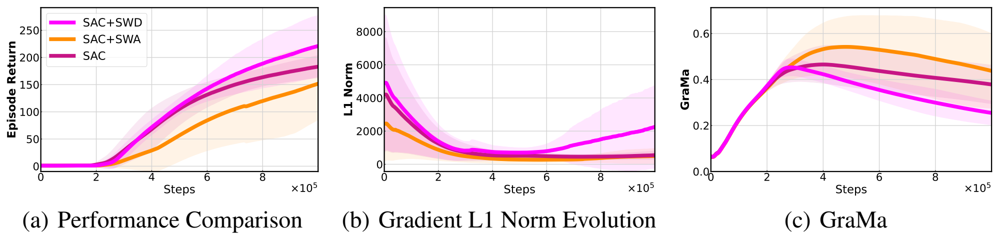
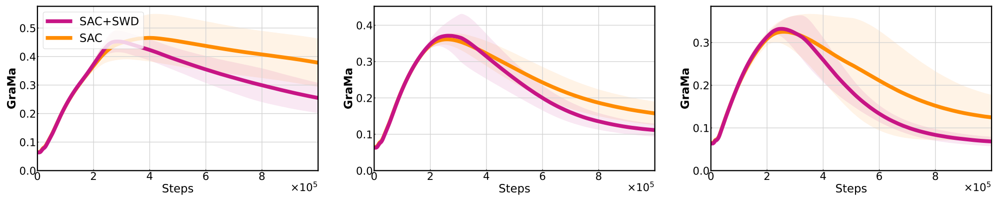


Figure 5: Experiments conducted in the humanoid-run environment demonstrate that SWA exhibits a lower gradient magnitude, GraMa, and inferior performance, which validates our hypothesis.

Results The reverse validation experiment yields key insights: *(i)* As shown in Figure 5(a), SWA consistently underperforms SWD and uniform sampling, validating that prioritizing recent experiences is critical for non-stationary RL learning; *(ii)* Figure 5(b) shows SWA reduces gradient L1 norms during training (weakened learning signals), aligning with our gradient attenuation analysis and confirming older data exacerbates plasticity loss; *(iii)* GraMa analysis in Figure 5(c) reveals SWA causes sparser gradients and greater plasticity loss than SWD (reduced neural activation/adaptation



(a) Humanoid Run (b) Humanoid Walk (c) Humanoid Stand

Figure 6: GraMa Metric in Humanoid Locomotion: Run, Walk, and Stand: The results clearly demonstrate that SWD effectively mitigates the loss of plasticity in humanoid robots across these key locomotor states.

capacity), providing direct empirical support for our theoretical framework's plasticity degradation prediction.

6.3 The Effect in Alleviating Plasticity Loss

To verify whether SWD can mitigate plasticity, we employed GraMa as the evaluation metric to quantify the degree of plasticity during the model training process. Notably, a larger GraMa value indicates a weaker learning capability of the neural network.

Results The corresponding results are illustrated in Figure 6. As depicted in this figure, our proposed SWD effectively alleviates the gradient sparsity that arises during the training process. Notably, the most pronounced effects are observed in the Humanoid Run environment and the Humanoid Stand environment. It can be clearly seen from the figure that SWD exerts its function in the middle and late stages of training — gradient attenuation is not severe in the early stage — and this observation is consistent with our theoretical predictions.

6.4 Compatibility Against Higher Update-to-Data Ratios

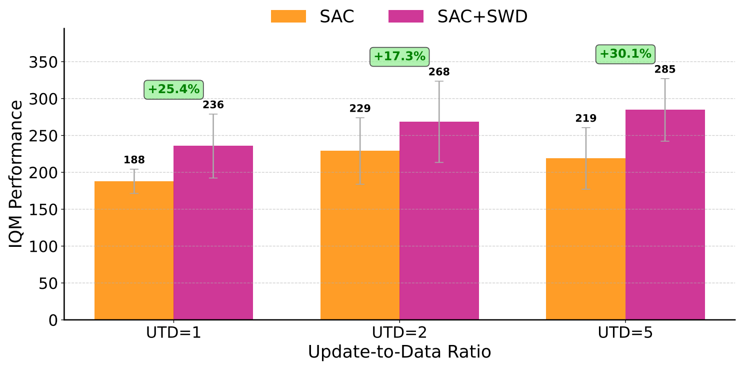


Figure 7: Performance comparison across different UTD ratios (1, 2, 5) in Humanoid Run. SWD consistently outperforms uniform sampling across all UTD settings, with improvements ranging from 17.3% to 30.1%.

The Update-to-Data Ratio (UTD) is a critical metric for measuring an algorithm's data utilization efficiency. Intuitively, uniform sampling assigns equal weight to each sample; after multiple updates, the gradient signals that can effectively guide the update of network parameters become very weak. In contrast, our SWD method assigns greater weight to more recent samples, ensuring that sufficiently strong gradient signals are maintained even after multiple updates.

As shown in Figure 7, SWD demonstrates consistent effectiveness across UTD ratios of 1, 2, and 5. Notably, the method shows the largest improvement (+30.1%) at UTD=5, suggesting that SWD is particularly beneficial when gradient updates are frequent. This robustness indicates that our approach is broadly applicable across different algorithmic configurations without requiring UTD-specific tuning.

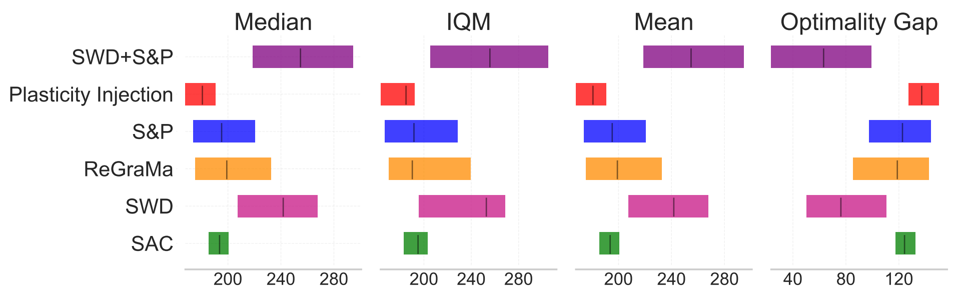


Figure 8: Performance comparison between `SWD+S&P` and other methods designed to address plasticity issues. Aggregate *Reliable* metrics (Agarwal et al., 2021) with 95% Stratified Bootstrap CIS in Humanoid run.

6.5 Comparison with Other Methods Designed to Address Plasticity Loss

To further evaluate the effectiveness of `SWD`, we compare it in the Humanoid Run environment with three representative methods that are designed to address plasticity issues: ReGraMa (Liu et al., 2025), S&P (Ash & Adams, 2020), and Plasticity Injection (Nikishin et al., 2023b). Moreover, we explore `SWD`'s synergistic potential with S&P, i.e., `SWD+S&P`, which demonstrates the orthogonality.

Results As in Figure 8, `SWD` outperforms other NTK-based methods on the SimBa (Lee et al., 2025a) network. Moreover, `SWD` combined with S&P yields the best result, validating its orthogonality to NTK-based methods. We provide a detailed discussion on the relationship between `SWD` and prior works for plasticity loss in Appendix C.2.

6.6 Other Results

Hyperparameter Choices and Decay Strategies To analyze the hyperparameter sensitivity of `SWD`, we conduct a grid-search test for two core hyperparameters, i.e., linear decay steps T and minimum weight threshold w_{min}. In Table 12 of Appendix F, `SWD` exhibits low sensitivity to different choices, demonstrating its stability. Moreover, we compare the linear decay strategy of `SWD` with the other two commonly adopted strategies, i.e., exponential decay and polynomial decay. Table 13 shows that the linear decay strategy outperforms the other two strategies.

Compute-efficient Approximation of `SWD` To further reduce the computational overhead of per-sample weight, we propose a bucket-based approximation method. As in Table 2 of Appendix D, this approximation significantly reduces the training time at no compromise of policy performance.

7 Conclusion

In this paper, we identified and addressed the critical issue of plasticity loss in long-horizon reinforcement learning through both theoretical analysis and algorithmic innovation. Our theoretical framework reveals that gradient attenuation follows a $\Theta(1/k)$ decay pattern, fundamentally limiting the agent's ability to adapt to new experiences over extended training periods. To counteract this degradation, we proposed Sample Weight Decay (`SWD`), a simple yet effective method that applies age-based weighting to replay buffer sampling. Through comprehensive experiments across MuJoCo, ALE and DMC environments with TD3, DDQN and SAC algorithms, we demonstrated consistent performance improvements ranging from 13.7% to 30.1% in IQM scores. Our ablation studies and reverse validation experiments confirm that temporal weighting direction is crucial for maintaining neural plasticity. The broad applicability of `SWD` across different algorithms, environments, and training configurations, combined with its minimal computational overhead, makes it a practical solution for enhancing long-horizon RL performance. This work opens new avenues for understanding and mitigating plasticity loss in deep reinforcement learning.

Limitations Owing to computational constraints, our evaluation is restricted to tasks within the MuJoCo, ALE and DeepMind Control Suite (DMC). Additionally, our exploration and practical application of the proposed theoretical framework remain at a preliminary stage—representing merely the "tip of the iceberg." Moving forward, future research will extend `SWD` to more complex scenarios, real-world environments. Ultimately, our goal is to develop `SWD` into a practical, robust tool that effectively preserves the learning capacity of deep reinforcement learning (RL) agents.

Reproducibility Statement

To promote transparency and reproducibility within the scientific community, we provide comprehensive details regarding training parameters and associated resources in the appendix. Additionally, the complete codebase for both the training and inference processes has been uploaded to the supplementary material.

Ethics Statement

This paper is committed to advancing the field of plasticity loss to develop more effective Reinforcement Learning (RL) algorithms. Our research adheres rigorously to responsible research practices and is fully aligned with the ICLR Code of Ethics. All training data utilized in this study was sourced from open-access datasets, and every asset employed strictly complies with the original licensing agreements and terms of service of the respective data providers.

REFERENCES

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *NeurIPS*, 2021.

Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *ICML*, 2019.

Kavosh Asadi, Rasool Fakoor, and Shoham Sabach. Resetting the optimizer in deep RL: an empirical study. In *NeurIPS*, 2023.

Jordan Ash and Ryan P Adams. On warm-starting neural network training. *NeurIPS*, 2020.

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*, 2013.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Celeste Biever. Chatgpt broke the turing test-the race is on for new ways to assess ai. *Nature*, 2023.

G Brockman. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Johan Samir Obando Ceron, Marc G Bellemare, and Pablo Samuel Castro. Small batch deep reinforcement learning. In *NeurIPS*, 2023.

Johan Samir Obando Ceron, Aaron Courville, and Pablo Samuel Castro. In value-based deep reinforcement learning, a pruned network is a good network. In *ICML*. PMLR, 2024.

Wesley Chung, Lynn Cherif, Doina Precup, and David Meger. Parseval regularization for continual reinforcement learning. In *NeurIPS*, 2024.

Rishabh Dixit, Mert Gurbuzbalaban, and Waheed U. Bajwa. Exit time analysis for approximations of gradient descent trajectories around saddle points. *Information and inference*, 2023.

Shibhansh Dohare, J. Fernando Hernandez-Garcia, Qingfeng Lan, Parash Rahman, A. Rupam Mahmood, and Richard S. Sutton. Loss of plasticity in deep continual learning. *Nature*, 632(8026): 768–774, 2024.

Simon S. Du, Jason D. Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of over-parameterized neural networks. In *ICLR*, 2019.

Benjamin Ellis, Matthew T. Jackson, Andrei Lupu, Alexander D. Goldie, Mattie Fellows, Shimon Whiteson, and Jakob Foerster. Adam on local time: Addressing nonstationarity in rl with relative adam timesteps. *arXiv preprint*, arXiv:2412.17113, 2024.

Mohamed Elsayed and A Rupam Mahmood. Addressing loss of plasticity and catastrophic forgetting in continual learning. *arXiv preprint arXiv:2404.00781*, 2024.

Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *JMLR*, 2005.

Lapo Frati, Neil Traft, Jeff Clune, and Nick Cheney. Reset it and forget it: Relearning last-layer weights improves continual and transfer learning. In *ECAI 2024*, pp. 2998–3005. 2024.

Scott Fujimoto, Herke Van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *ICLR*, 2018.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, and Pieter Abbeel. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *AAAI*, 2016.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *NeurIPS*, 2018.

Zilin Kang, Chenyuan Hu, Yu Luo, Zhecheng Yuan, Ruijie Zheng, and Huazhe Xu. A forget-and-grow strategy for deep reinforcement learning scaling in continuous control. *ICML*, 2025.

Saurabh Kumar, Henrik Marklund, Ashish Rao, Yifan Zhu, Hong Jun Jeon, Yueyang Liu, and Benjamin Van Roy. Continual learning as computationally constrained reinforcement learning. *arXiv preprint, arXiv:2307.04345*, 2023.

Hojoon Lee, Dongyoon Hwang, Donghu Kim, Hyunseung Kim, Jun Jet Tai, Kaushik Subramanian, Peter R. Wurman, Jaegul Choo, Peter Stone, and Takuma Seno. Simba: Simplicity bias for scaling up parameters in deep reinforcement learning. In *ICLR*, 2025a.

Hojoon Lee, Youngdo Lee, Takuma Seno, Donghu Kim, Peter Stone, and Jaegul Choo. Hyperspherical normalization for scalable deep reinforcement learning, 2025b.

Alex Lewandowski, Haruto Tanaka, Dale Schuurmans, and Marlos C. Machado. Directions of curvature as an explanation for loss of plasticity. volume arXiv:2312.00246, 2023.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Jiashun Liu, Zihao Wu, Johan Obando-Ceron, Pablo Samuel Castro, Aaron Courville, and Ling Pan. Measure gradients, not activations! enhancing neuronal activity in deep reinforcement learning. *arXiv preprint arXiv:2505.24061*, 2025.

Skander Moalla, Andrea Miele, Daniil Pyatko, Razvan Pascanu, and Caglar Gulcehre. No representation, no trust: connecting representation, collapse, and trust issues in ppo. *NeurIPS*, 37: 69652–69699, 2024.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In *ICML*, 2022.

Evgenii Nikishin, Junhyuk Oh, Georg Ostrovski, Clare Lyle, Razvan Pascanu, Will Dabney, and André Barreto. Deep reinforcement learning with plasticity injection. *arXiv preprint arXiv:2305.15555*, 2023a.

Evgenii Nikishin, Junhyuk Oh, Georg Ostrovski, Clare Lyle, Razvan Pascanu, Will Dabney, and André Barreto. Deep reinforcement learning with plasticity injection. *NeurIPS*, 2023b.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *ICLR*, 2016.

Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning. In *ICML*, 2023.

Hongyao Tang and Glen Berseth. Improving deep reinforcement learning by reducing the chain effect of value and policy churn. In *NeurIPS*, 2024.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

A THE USE OF LARGE LANGUAGE MODELS (LLMS)

Large Language Models (LLMs) were utilized to support the writing and refinement of this manuscript. Specifically, an LLM was employed to assist in enhancing language clarity, improving readability, and ensuring coherent expression across different sections of the paper. It aided in tasks like rephrasing sentences, checking grammar, and optimizing the overall textual flow.

It should be noted that the LLM played no role in the conception of research ideas, the formulation of research methodologies, or the design of experiments. All research concepts, ideas, and analyses were independently developed and carried out by the authors. The LLM's contributions were strictly limited to elevating the linguistic quality of the paper, without any involvement in the scientific content or data analysis.

The authors fully assume responsibility for the entire content of the manuscript, including any text generated or polished with the help of the LLM. We have verified that the text produced with the LLM complies with ethical guidelines and does not lead to plagiarism or any form of scientific misconduct.

B PROOF

B.1 PROOF OF THEOREM 1

In this section, we prove Theorem1.

$$
\begin{aligned}
&\mathbb{E}\mathcal{L}_h^k(f, \hat{f}_{h+1}^k) \\
&= \mathbb{E}_{(s_h,a_h)\sim\mu_h^k}\left[\mathbb{E}_{s_{h+1}\sim p_h(\cdot|s_h,a_h)}\left[\left(f(s_h,a_h) - r(s_h,a_h) - \max_{a'} \hat{f}_{h+1}^k(s_{h+1},a')\right)^2\right]\right] \\
&= \mathbb{E}_{(s_h,a_h)\sim\mu_h^k}\Bigg[\mathbb{E}_{s_{h+1}\sim p_h(\cdot|s_h,a_h)}\Bigg[\Big(f(s_h,a_h) - \mathcal{T}_h\hat{f}_{h+1}^k(s_h,a_h) \\
&\qquad\qquad + \mathbb{P}_h \max_{a'} \hat{f}_{h+1}^k(s_{h+1},a')(s_h,a_h) - \max_{a'} \hat{f}_{h+1}^k(s_{h+1},a')\Big)^2\Bigg]\Bigg] \\
&= \mathbb{E}_{(s_h,a_h)\sim\mu_h^k}\Bigg[\mathbb{E}_{s_{h+1}\sim p_h(\cdot|s_h,a_h)}\Bigg[\left(f(s_h,a_h) - \mathcal{T}_h\hat{f}_{h+1}^k(s_h,a_h)\right)^2 \\
&\qquad\qquad + \left(\mathbb{P}_h \max_{a'} \hat{f}_{h+1}^k(s_{h+1},a')(s_h,a_h) - \max_{a'} \hat{f}_{h+1}^k(s_{h+1},a')\right)^2 \\
&\qquad\qquad + 2\left(f(s_h,a_h) - \mathcal{T}_h\hat{f}_{h+1}^k(s_h,a_h)\right) \\
&\qquad\qquad\quad \times \left(\mathbb{P}_h \max_{a'} \hat{f}_{h+1}^k(s_{h+1},a')(s_h,a_h) - \max_{a'} \hat{f}_{h+1}^k(s_{h+1},a')\right)\Bigg]\Bigg] \\
&= \mathbb{E}_{(s_h,a_h)\sim\mu_h^k}\left[\left(f(s_h,a_h) - \mathcal{T}_h\hat{f}_{h+1}^k(s_h,a_h)\right)^2\right] \\
&\quad + \mathbb{E}_{(s_h,a_h)\sim\mu_h^k}\left[\mathbb{E}_{s_{h+1}\sim p_h(\cdot|s_h,a_h)}\left[\left(\mathbb{P}_h \max_{a'} \hat{f}_{h+1}^k(s_{h+1},a')(s_h,a_h) - \max_{a'} \hat{f}_{h+1}^k(s_{h+1},a')\right)^2\right]\right] \\
&= \mathbb{E}_{(s_h,a_h)\sim\mu_h^k}\left[\left(f(s_h,a_h) - \mathcal{T}_h\hat{f}_{h+1}^k(s_h,a_h)\right)^2\right] + \mathbb{E}_{(s_h,a_h)\sim\mu_h^k}\left[\mathrm{Var}_{s_{h+1}\sim p_h(\cdot|s_h,a_h)}\left[\max_{a'} \hat{f}_{h+1}^k(s_{h+1},a')\right]\right]
\end{aligned}
$$

The loss function can be decomposed into two components:

- **Bellman Residual Term**: $\mathbb{E}_{(s_h,a_h)\sim\mu_h^k}\left[\left(f(s_h,a_h) - \mathcal{T}_h\hat{f}_{h+1}^k(s_h,a_h)\right)^2\right]$–Measures the function approximation error.

- **Environmental Stochasticity Term**: $\mathbb{E}_{(s_h,a_h)\sim\mu_h^k}\left[\mathrm{Var}_{s_{h+1}\sim p_h(\cdot|s_h,a_h)}\left[\max_{a'} \hat{f}_{h+1}^k(s_{h+1},a')\right]\right]$– Reflects the intrinsic randomness of state transitions.

B.2 Proof of Theorem 2

First, we prove one lemma to help the proof.

Lemma 1. *Consider an episodic MDP with horizon H. Let $\pi' = \{\pi'_h\}_{h=1}^H$ denote any policy, and let $\{\hat{Q}_h\}_{h=1}^H$ denote any set of estimated Q-functions. Let $\pi = \{\pi_h\}_{h=1}^H$ be the greedy policy induced by $\{\hat{Q}_h\}_{h=1}^H$.*

For all $h \in [H]$, define:

- *Value function: $\hat{V}_h(s) = \mathbb{J}_h^\pi \hat{Q}_h(s)$ where $\mathbb{J}_h^\pi f(s) = \mathbb{E}_{a\sim\pi_h(\cdot|s)}[f(s,a)]$*
- *Bellman residual: $l_h(s,a) := \hat{Q}_h(s,a) - (\mathcal{T}_h\hat{Q}_{h+1})(s,a)$*

Then, for all elements $x \in \mathcal{S}, the following holds:$

$$\hat{V}_1(x) - V_1^{\pi'}(x) = \sum_{h=1}^H \mathbb{E}_{\pi'}\left[(\mathbb{J}_h^\pi - \mathbb{J}_h^{\pi'})\hat{Q}_h(s_h) \mid s_1 = x\right] + \sum_{h=1}^H \mathbb{E}_{\pi'}\left[\hat{Q}_h(s_h,a_h) - (\mathcal{T}_h\hat{Q}_{h+1})(s_h,a_h) \mid s_1 = x\right]$$

Proof.

$$\begin{aligned}
\hat{V}_h(x) - V_h^{\pi'}(x) &= \mathbb{J}_h^\pi\hat{Q}_h(x) - \mathbb{J}_h^{\pi'}Q_h^{\pi'}(x) \\
&= \mathbb{J}_h^\pi\hat{Q}_h(x) - \mathbb{J}_h^{\pi'}\hat{Q}_h(x) + \mathbb{J}_h^{\pi'}\hat{Q}_h(x) - \mathbb{J}_h^{\pi'}Q_h^{\pi'}(x) \\
&= \mathbb{J}_h^{\pi'}(\hat{Q}_h - Q_h^{\pi'})(x) + (\mathbb{J}_h^\pi - \mathbb{J}_h^{\pi'})\hat{Q}_h(x) \\
&= \mathbb{J}_h^{\pi'}(l_h + \mathcal{T}_\langle\hat{Q}_{h+1} - r_h - \mathbb{P}_h V_{h+1}^{\pi'})(x) + (\mathbb{J}_h^\pi - \mathbb{J}_h^{\pi'})\hat{Q}_h(x) \\
&= \mathbb{J}_h^{\pi'}(l_h + \mathbb{P}_h\hat{V}_{h+1} - \mathbb{P}_h V_{h+1}^{\pi'})(x) + (\mathbb{J}_h^\pi - \mathbb{J}_h^{\pi'})\hat{Q}_h(x) \\
&= \mathbb{J}_h^{\pi'}l_h(x) + \mathbb{J}_h^{\pi'}\mathbb{P}_h(\hat{V}_{h+1} - V_{h+1}^{\pi'})(x) + (\mathbb{J}_h^\pi - \mathbb{J}_h^{\pi'})\hat{Q}_h(x)
\end{aligned}$$

Using recurrence relations and the boundary condition $\hat{V}_{H+1} = V_{H+1}^{\pi'} \equiv 0$, we can derive that

$$\hat{V}_1(x) - V_1^{\pi'}(x) = \sum_{h=1}^H \left(\prod_{k=1}^{h-1}\mathbb{J}_k^{\pi'}\mathbb{P}_k\right)\mathbb{J}_h^{\pi'}l_h(x) + \sum_{h=1}^H \left(\prod_{k=1}^{h-1}\mathbb{J}_k^{\pi'}\mathbb{P}_k\right)(\mathbb{J}_h^\pi - \mathbb{J}_h^{\pi'})\hat{Q}_h(x)$$

Which complete our proof. □

let π' be the optimal policy π^*,π be the greedy policy induced by $\{\hat{Q}_h\}_{h=1}^H$, and $\{\hat{V}_h\}_{h=1}^H$ be the corresponding value function.Then, the suboptimal bound is given by:

$$V_1^*(x) - V_1^\pi(x) = V_1^*(x) - \hat{V}_1(x) + \hat{V}_1(x) - V_1^\pi(x)$$

$$= \underbrace{\sum_{h=1}^H \mathbb{E}_{\pi^*}\left[\mathcal{T}_h\hat{Q}_{h+1}(s_h,a_h) - \hat{Q}_h(s_h,a_h) \mid s_1 = x\right]}_{①} + \underbrace{\sum_{h=1}^H \mathbb{E}_{\pi}\left[\hat{Q}_h(s_h,a_h) - \mathcal{T}_h\hat{Q}_{h+1}(s_h,a_h) \mid s_1 = x\right]}_{②}$$

$$+ \underbrace{\sum_{h=1}^H \mathbb{E}_{\pi^*}\left[(\mathbb{J}_h^{\pi^*} - \mathbb{J}_h^{\pi})\hat{Q}_h \mid s_1 = x\right]}_{③}$$

Since π is the greedy policy with respect to $\hat{Q}$, we have $③ \leq 0$, and for ① we can drive that:

$$① \leq \sum_{h=1}^H \mathbb{E}_{\pi^*}\left[\left|\mathcal{T}_h\hat{Q}_{h+1}(s_h,a_h) - \hat{Q}_h(s_h,a_h)\right| \Big| s_1 = x\right]$$
$$\leq \sum_{h=1}^H \sqrt{\mathbb{E}_{\pi^*}\left[\left(\mathcal{T}_h\hat{Q}_{h+1}(s_h,a_h) - \hat{Q}_h(s_h,a_h)\right)^2 \mid s_1 = x\right]}$$
$$\leq \sqrt{H}\sqrt{\sum_{h=1}^H \mathbb{E}_{\pi^*}\left[\left(\mathcal{T}_h\hat{Q}_{h+1}(s_h,a_h) - \hat{Q}_h(s_h,a_h)\right)^2 \mid s_1 = x\right]}$$

The last step makes use of the Cauchy-Schwarz inequality, and the second step employs Jensen's inequality.

Similarly, we can derive that ② also satisfies:

$$② \leq \sqrt{H}\sqrt{\sum_{h=1}^H \mathbb{E}_{\pi}\left[\left(\mathcal{T}_h\hat{Q}_{h+1}(s_h,a_h) - \hat{Q}_h(s_h,a_h)\right)^2 \mid s_1 = x\right]}$$

By combining the above results, we complete the proof of Theorem 2.

B.3 Proof of theorem 3

In this section, we prove Theorem 3.

Proof.

$$\nabla\mathbb{E}_{\mu_h^k}\left[(f - \mathcal{T}_h\hat{f}_{h+1}^k)^2\right]\Big|_{\hat{f}_h^{k-1}} = \mathbb{E}_{\mu_h^k}\left[2\left(f - \mathcal{T}_h\hat{f}_{h+1}^k\right)\nabla f\Big|_{\hat{f}_h^{k-1}}\right]$$
$$= \mathbb{E}_{\mu_h^k}\left[2\left(f - \mathcal{T}_h\hat{f}_h^{k-1} + \mathcal{T}_h\hat{f}_h^{k-1} - \mathcal{T}_h\hat{f}_{h+1}^k\right)\nabla f\Big|_{\hat{f}_h^{k-1}}\right]$$
$$= \mathbb{E}_{\mu_h^k}\left[2\left(\mathcal{T}_h\hat{f}_h^{k-1} - \mathcal{T}_h\hat{f}_{h+1}^k\right)\nabla f\Big|_{\hat{f}_h^{k-1}}\right] + \underbrace{\mathbb{E}_{\mu_h^k}\left[2\left(f - \mathcal{T}_h\hat{f}_{h+1}^{k-1}\right)\nabla f\Big|_{\hat{f}_h^{k-1}}\right]}_{①}$$

Recall the define of $\hat{f}_h^{k-1}$ and the Proposition 1 of μ_h^k,

$$① = \nabla\mathbb{E}_{\mu_h^k}\left[\left(f - \mathcal{T}_h\hat{f}_{h+1}^{k-1}\right)^2\right]\Big|_{\hat{f}_h^{k-1}}$$
$$= \underbrace{\frac{k-1}{k}\nabla\mathbb{E}_{\mu_h^{k-1}}\left[\left(f - \mathcal{T}_h\hat{f}_{h+1}^{k-1}\right)^2\right]\Big|_{\hat{f}_h^{k-1}}}_{=0} + \frac{1}{k}\nabla\mathbb{E}_{\hat{d}_h^{\pi^k}}\left[\left(f - \mathcal{T}_h\hat{f}_{h+1}^{k-1}\right)^2\right]\Big|_{\hat{f}_h^{k-1}}$$

By combining the above results, we complete the proof of Theorem 3 □

B.4 Entropy Regularized MDP

In this section, we present the theoretical analysis and error bounds for the Entropy-Regularized Markov Decision Process (MDP). Specifically, the state value function with an entropy reward is defined as follows:

$$V_h^{\text{soft},\pi}(x) = \mathbb{E}\left[\sum_{t=h}^{H}(r_t(x_t,a_t)+\alpha\log\pi_t(a_t\mid x_t))\middle| x_h = x, a_t\sim\pi_t(\cdot|x_t)\right], \qquad \forall x\in\mathbb{S}, h\in[H],$$

$$Q_h^{\text{soft},\pi}(x,a) = r_h(x,a)+\mathbb{P}_h V_{h+1}^{\text{soft},\pi}(x,a), \qquad \forall (x,a)\in\mathbb{S}\times\mathbb{A}, h\in[H]$$

with terminal condition $V_{H+1}^{\text{soft},\pi}\equiv 0$. Then the policy Bellman equations compactly read

$$Q_h^{\text{soft},\pi}(x,a) = r_h(x,a)+\mathbb{P}_h V_{h+1}^{\text{soft},\pi}(x,a)$$
$$V_h^{\text{soft},\pi}(x) = \mathbb{J}_h^{\pi}(Q_h^{\text{soft},\pi}-\alpha\log\pi_h)(x), \qquad V_{H+1}^{\text{soft}}\equiv 0$$

For any function $g:\mathbb{S}\times\mathbb{A}\to\mathbb{R}$, define the soft value operator $\mathbb{V}^{\text{soft}}$ and the step-h soft optimality Bellman operator $\mathcal{T}_h^{\text{soft}}$ by

$$\mathbb{V}_g^{\text{soft}}(s) := \max_{\pi}\mathbb{E}_{a\sim\pi}\left[g(s,a)-\alpha\log\pi(a|s)\right],$$
$$\left(\mathcal{T}_h^{\text{soft}}f\right)(s,a) := r(s,a)+\left(\mathbb{P}_h\mathbb{V}_f^{\text{soft}}\right)(s,a)$$

We define the Boltzmann policy π_f^{soft} induced by the function $f:\mathbb{S}\times\mathbb{A}\to\mathbb{R}$, which is given by:

$$\pi_f^{\text{soft}} = \arg\max_{\pi}\mathbb{E}_{a\sim\pi}[f(s,a)-\alpha\log\pi(a|s)].$$

Similarly, we have the following lemma.

Lemma 2. *Consider an entropy-regularized episodic MDP with horizon H. Let $\pi' = \{\pi'_h\}_{h=1}^H$ denote any policy, and let $\hat{Q} = \{\hat{Q}_h\}_{h=1}^H$ denote any set of estimated soft Q-functions. Let $\pi = \{\pi_h\}_{h=1}^H$ be the Boltzmann policy induced by $\hat{Q} = \{\hat{Q}_h\}_{h=1}^H$. For all $h\in[H]$, define:*

- *Value function: $\hat{V}_h(s) = \mathbb{J}_h^{\pi}(\hat{Q}_h-\alpha\log\pi_h)(s)$ where $\mathbb{J}_h^{\pi}f(s) = \mathbb{E}_{a\sim\pi_h(\cdot|s)}[f(s,a)]$*
- *Bellman residual: $l_h(s,a) := \hat{Q}_h(s,a)-(\mathcal{T}_h^{soft}\hat{Q}_{h+1})(s,a)$*
- *Entropy: $\mathcal{H}(\pi(\cdot|s)) = -\mathbb{E}_{a\sim\pi(\cdot|s)}[\log\pi(\cdot|s)]$*

Then for all $x\in\mathbb{S}$, we have

$$\hat{V}_1(x)-V_1^{soft,\pi'}(x) = \sum_{h=1}^{H}\mathbb{E}_{\pi'}\left[(\mathbb{J}_h^{\pi}-\mathbb{J}_h^{\pi'})\hat{Q}_h(s_h)+\alpha(\mathcal{H}(\pi_h(\cdot|s_h))-\mathcal{H}(\pi'_h(\cdot|s_h)))\mid s_1=x\right]$$
$$+\sum_{h=1}^{H}\mathbb{E}_{\pi'}\left[\hat{Q}_h(s_h,a_h)-(\mathcal{T}_h^{soft}\hat{Q}_{h+1})(s_h,a_h)\mid s_1=x\right].$$

Proof.

$$\begin{aligned}
\hat{V}_h(x)-V_h^{\text{soft},\pi'}(x) &= \mathbb{J}_h^{\pi}(\hat{Q}_h-\alpha\log\pi)(x)-\mathbb{J}_h^{\pi'}(Q_h^{\text{soft},\pi'}-\alpha\log\pi')(x)\\
&= \mathbb{J}_h^{\pi}\hat{Q}_h(x)-\mathbb{J}_h^{\pi'}Q_h^{\text{soft},\pi'}(x)+\alpha(\mathcal{H}(\pi_h(\cdot|x))-\mathcal{H}(\pi'_h(\cdot|x)))\\
&= \mathbb{J}_h^{\pi}\hat{Q}_h(x)-\mathbb{J}_h^{\pi'}\hat{Q}_h(x)+\mathbb{J}_h^{\pi'}\hat{Q}_h(x)-\mathbb{J}_h^{\pi'}Q_h^{\text{soft},\pi'}(x)+\alpha(\mathcal{H}(\pi_h(\cdot|x))-\mathcal{H}(\pi'_h(\cdot|x)))\\
&= \mathbb{J}_h^{\pi'}(\hat{Q}_h-Q_h^{\text{soft},\pi'})(x)+\mathbb{J}_h^{\pi}\hat{Q}_h(x)-\mathbb{J}_h^{\pi'}\hat{Q}_h(x)+\alpha(\mathcal{H}(\pi_h(\cdot|x))-\mathcal{H}(\pi'_h(\cdot|x)))\\
&= \mathbb{J}_h^{\pi'}(l_h+\mathcal{T}_h^{\text{soft}}\hat{Q}_{h+1}-r-\mathbb{P}_hV_{h+1}^{\text{soft},\pi'})(x)+\mathbb{J}_h^{\pi}\hat{Q}_h(x)-\mathbb{J}_h^{\pi'}\hat{Q}_h(x)+\alpha(\mathcal{H}(\pi_h(\cdot|x))-\mathcal{H}(\pi'_h(\cdot|x)))\\
&= \mathbb{J}_h^{\pi'}(l_h+\mathbb{P}_h(\hat{V}_{h+1}-V_{h+1}^{\text{soft},\pi'}))(x)+\mathbb{J}_h^{\pi}\hat{Q}_h(x)-\mathbb{J}_h^{\pi'}\hat{Q}_h(x)+\alpha(\mathcal{H}(\pi_h(\cdot|x))-\mathcal{H}(\pi'_h(\cdot|x)))\\
&= \mathbb{J}_h^{\pi'}l_h(x)+\mathbb{J}_h^{\pi'}\mathbb{P}_h(\hat{V}_{h+1}-V_{h+1}^{\text{soft},\pi'})(x)+\mathbb{J}_h^{\pi}\hat{Q}_h(x)-\mathbb{J}_h^{\pi'}\hat{Q}_h(x)+\alpha(\mathcal{H}(\pi_h(\cdot|x))-\mathcal{H}(\pi'_h(\cdot|x))).
\end{aligned}$$

Using recurrence relations and the boundary condition $\hat{V}_{H+1} = V_{H+1}^{\text{soft},\pi'} \equiv 0$, we can derive that

$$\hat{V}_1(x) - V_1^{\text{soft},\pi'}(x) = \sum_{h=1}^{H} \left(\prod_{k=1}^{h-1} \mathbb{J}_k^{\pi'} \mathbb{P}_k \right) \mathbb{J}_h^{\pi'} l_h(x)$$
$$+ \sum_{h=1}^{H} \left(\prod_{k=1}^{h-1} \mathbb{J}_k^{\pi'} \mathbb{P}_k \right) \Big((\mathbb{J}_h^{\pi} - \mathbb{J}_h^{\pi'}) \hat{Q}_h(x) + \alpha \left(\mathcal{H}(\pi_h(\cdot|x)) - \mathcal{H}(\pi'_h(\cdot|x)) \right) \Big).$$

This completes the proof. □

Theorem 4 (Suboptimality bound for entropy-regularized MDP via squared Bellman residuals)**.** *Fix horizon H. Let $\{\hat{Q}_h\}_{h=1}^H$ be the soft value estimates. Define $\{\pi_{\hat{Q},h}\}_{h=1}^H$ as the Boltzmann policy induced by $\{\hat{Q}_h\}_{h=1}^H$. Let $\{\pi_h^*\}_{h=1}^H$ be the optimal policy.*

For functions $f, g : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$, define the step-h squared Bellman residual:

$$\Delta_h(f,g)(s,a) = \Big(f(s,a) - \mathcal{T}_h^{soft} g(s,a) \Big)^2.$$

Then we have

$$V_1^{soft,\pi^*}(x) - V_1^{soft,\pi_{\hat{Q}}}(x) \le \sqrt{H} \left(\sqrt{\mathbb{E}_{\pi^*}\left[\sum_{h=1}^{H} \Delta_h(\hat{Q}_h, \hat{Q}_{h+1})(s_h,a_h) \,\middle|\, s_1 = x \right]} + \sqrt{\mathbb{E}_{\pi_{\hat{Q}}}\left[\sum_{h=1}^{H} \Delta_h(\hat{Q}_h, \hat{Q}_{h+1})(s_h,a_h) \,\middle|\, s_1 = x \right]} \right).$$

Proof.

$$V_1^{\text{soft},\pi^*}(x) - V_1^{\text{soft},\pi_{\hat{Q}}}(x) = V_1^{\text{soft},\pi^*}(x) - \hat{V}_1(x) + \hat{V}_1(x) - V_1^{\text{soft},\pi_{\hat{Q}}}(x)$$
$$= \underbrace{\sum_{h=1}^{H} \mathbb{E}_{\pi^*}\left[\mathcal{T}_h^{\text{soft}} \hat{Q}_{h+1}(s_h,a_h) - \hat{Q}_h(s_h,a_h) \mid s_1 = x \right]}_{①} + \underbrace{\sum_{h=1}^{H} \mathbb{E}_{\pi_{\hat{Q}}}\left[\hat{Q}_h(s_h,a_h) - \mathcal{T}_h^{\text{soft}} \hat{Q}_{h+1}(s_h,a_h) \mid s_1 = x \right]}_{②}$$
$$+ \underbrace{\sum_{h=1}^{H} \mathbb{E}_{\pi^*}\left[\left(\mathbb{J}_h^{\pi^*} - \mathbb{J}_h^{\pi_{\hat{Q}}} \right) \hat{Q}_h + \alpha \Big(\mathcal{H}(\pi_h^*(\cdot|s_h)) - \mathcal{H}(\pi_{\hat{Q},h}(\cdot|s_h)) \Big) \mid s_1 = x \right]}_{③}.$$

Since $\pi_{\hat{Q}}$ is the Boltzmann policy induced by $\{\hat{Q}_h\}_{h=1}^H$—a property that satisfies Equation B.4—we can deduce that ③ ≤ 0. The remainder of the proof follows the same reasoning as that of Theorem 2. □

C Related Preliminaries

In this section, we present the detailed parameters and settings of the experiments.

C.1 Algorithm

TD3 In our paper, we utilize TD3 as a representative of deterministic policies. TD3, an Actor - Critic algorithm, is widely adopted as a baseline in various decision - making scenarios and has given rise to a multitude of variants, which have established new state - of - the - art (SOTA) results on numerous occasions. Different from the traditional policy gradient method DDPG (Lillicrap et al., 2015), TD3 makes use of two heterogeneous critic networks, denoted as $Q_{\theta_{1,2}}$, to alleviate the problem of over - optimization in Q - learning. Thus, the loss function of the critics is

$$\mathcal{L}_Q(\theta_i) = \mathbb{E}_{a,s,r,s'}\left[(y - Q_{\theta_i}(s,a))^2 \right] \text{ for } \forall i \in \{1,2\}.$$

Where $y = r + \gamma \min_{j=1,2} Q_{\tilde{\theta}_j}(s', \pi_\phi(s'))$, $\tilde{\theta}$ denotes the target network parameters. The actor is updated according to the Deterministic Policy Gradient:

$$\nabla_\phi J(\phi) = \mathbb{E}_s \left[\nabla_a Q_{\theta_1}(s, \pi_\phi(s)) \nabla_\phi \pi_\phi(s) \right].$$

SAC We select SAC as a representative of stochastic policies and combine it with `SWD` in the main experiment. SAC is devised to maximize expected cumulative rewards while also boosting exploration via the maximum entropy principle. The actor strives to learn a stochastic policy that outputs a distribution over actions, where the critics estimate the value of taking a specific action in a given state. This enables a more diverse range of actions, facilitating better exploration of the action space. In traditional reinforcement learning, the objective is to maximize the expected return. However, SAC introduces an additional term that maximizes the entropy of the policy, encouraging exploration. The objective function for optimizing the policy is given by:

$$J(\pi) = \mathbb{E}_{s_t, a_t} \left[r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t)) \right]$$

where $H(\pi(\cdot|s_t))$ denotes the entropy of the policy, and α is a temperature parameter that balances the trade-off between the immediate reward and the policy entropy. The training procedure of SAC involves two main updates: updating the value function and updating the policy. The value function is updated by minimizing the following loss:

$$\mathcal{L}(Q) = \mathbb{E}_{(s,a,r,s') \sim D} \left[\frac{1}{2} \left(Q(s,a) - (r + \gamma V(s')) \right)^2 \right]$$

where γ is the discount factor, dictating the weight assigned to future rewards. $V(s')$ denotes the value function of the next state, which is typically approximated using a separate neural network. The policy is updated by maximizing the following objective:

$$J(\pi) = \mathbb{E}_{s_t \sim D} \left[\mathbb{E}_{a_t \sim \pi} \left[Q(s_t, a_t) - \alpha \log \pi(a_t|s_t) \right] \right]$$

Here, $-\alpha \log \pi(a_t|s_t)$ represents the entropy of the policy, which serves to promote exploration.

SimBa We adopt SimBa (Lee et al., 2025a) as our SAC network architecture, which is specifically designed for reinforcement learning (RL) scenarios. Distinctive for embedding a ”simplicity bias,” SimBa not only mitigates overfitting but also enables parameter scaling in deep RL—addressing two key challenges in large-scale RL model training. Concretely, SimBa comprises three core components: (i) an observation normalization layer that standardizes input data using running statistics, ensuring stable data distribution for subsequent layers; (ii) a residual feedforward block that establishes a direct linear pathway from input to output, facilitating gradient propagation and preserving low-complexity feature representations; and (iii) a layer normalization module that regulates feature magnitudes, preventing excessive value drift during training.

Prioritized Experience Replay We adopt Prioritized Experience Replay (PER) (Schaul et al., 2016) to bias sampling toward transitions that are expected to yield larger learning progress. Instead of drawing mini-batches uniformly from the replay buffer, PER assigns each transition i a priority p_i based on its temporal-difference (TD) error and samples proportionally:

$$\delta_i = \left| r_i + \gamma \hat{V}(s'_i) - Q(s_i, a_i) \right|, \qquad p_i = \left(\delta_i + \varepsilon \right)^\alpha, \qquad P(i) = \frac{p_i}{\sum_j p_j},$$

where $\varepsilon > 0$ avoids zero priorities, $\alpha \in [0, 1]$ controls the degree of prioritization ($\alpha = 0$ recovers uniform sampling). To correct the sampling bias introduced by $P(i)$, PER uses importance-sampling (IS) weights

$$w_i = \left(\frac{1}{N\,P(i)} \right)^\beta, \qquad \tilde{w_i} = \frac{w_i}{\max_j w_j},$$

where N is the buffer size and $\beta \in [0, 1]$ is annealed toward 1 during training.

Gradient Magnitude-based neuron activity assessment We employ `GraMa` (Liu et al., 2025)—a gradient-magnitude-driven, architecture-agnostic metric— as our plasticity metric. Specifically, for each individual neuron (or predefined parameter group), `GraMa` calculates the magnitude of gradients computed over mini-batches and maintains a normalized score for each layer; crucially, higher scores correspond to greater neural plasticity.

Given an input distribution D, let $|\nabla h_\ell^i L(x)|$ denote the gradient magnitude of neuron i in layer ℓ under an input x $\in$ D, and let H_ℓ represent the number of neurons in layer ℓ. The learning capacity score for each individual neuron by leveraging the normalized average of its corresponding layer ℓ, as formulated below:

$$G_\ell^i = \frac{\mathbb{E}_{x \in D}\left[|\nabla h_\ell^i L(x)|\right]}{\frac{1}{H_\ell}\sum_{k \in \mathcal{H}_\ell} \mathbb{E}_{x \in D}\left[|\nabla h_\ell^k L(x)|\right]}$$

GraMa (Gradient Magnitude-based neuron activity assessment) identifies neuron i in layer ℓ as inactive if $G_\ell^i \leq \tau$, where τ denotes the predefined inactivity threshold.

Double DQN We adopt Double Deep Q-Network (DDQN) (Hasselt et al., 2016) as our reinforcement learning (RL) baseline, specifically chosen for both pixel-based input scenarios and tasks with long time horizons. By decoupling action selection from target value estimation, DDQN effectively mitigates the overestimation bias inherent in standard DQN, ensuring more stable and accurate value learning. Concretely, while standard DQN maximizes the estimated value using the same network, DDQN utilizes the online network with parameters θ to select the optimal action and the target network with parameters θ^- to evaluate that action.

C.2 Relationship and Complementarity with Existing Work

Prior research on plasticity loss has predominantly centered on **NTK-based methods**, which we classify into three core categories based on their underlying mechanisms:

(1) **Reset-based methods (leveraging random initialization properties)**: These approaches capitalize on a key characteristic of over-parameterized neural networks: randomly initialized networks exhibit full-rank Neural Tangent Kernel (NTK) matrices. To mitgate plasticity loss, they periodically reset network parameters to refresh the NTK and restore the model's capacity for learning. Representative examples include:

- **ReDo** (Sokar et al., 2023): Employs activation-driven reinitialization to reset critical network components
- **ReGraMa** (Liu et al., 2025): Utilizes gradient information to guide parameter reinitialization, targeting degraded NTK structures
- **S&P** (Ash & Adams, 2020): Introduces controlled noise into network parameters to reactivate dormant plasticity
- **Plasticity Injection** (Nikishin et al., 2023b): Under the premise of keeping the output unchanged, thoroughly refresh the final linear layer.

(2) **Implicit NTK regularization methods**: This category focuses on detecting early signs of NTK rank deficiency—such as unconstrained parameter norm growth—and implementing targeted constraints to avert rank collapse. Key strategies within this framework are:

- **Reducing Churn** (Tang & Berseth, 2024): Suppresses off-diagonal elements of the NTK matrix to minimize gradient correlations, while dynamically adjusting step sizes in reinforcement learning (RL) settings to preserve NTK integrity
- **Auxiliary-loss-based representation stabilization** (Moalla et al., 2024): Integrates additional loss terms to stabilize feature representations, indirectly safeguarding NTK rank

(3) **Architecture-based methods**: These approaches address plasticity loss at the network design level, either by constructing inherently larger and more robust architectures or by dynamically expanding parameter counts during training to prevent NTK rank collapse. Notable instances include:

- **Hyperspherical Normalization for Scalable Deep RL** (Lee et al., 2025b): Designs architectures with built-in stability, leveraging hyperspherical normalization to maintain NTK full-rank properties

- **Forget-and-Grow Strategy for Deep RL Scaling** (Kang et al., 2025): Implements dynamic parameter expansion to sustain NTK rank and preserve plasticity

Our work is **fundamentally orthogonal** to these NTK-based paradigms. Unlike existing methods—which tackle plasticity loss through architectural modifications, explicit NTK regularization, or parameter resetting—we adopt a **novel gradient dynamics perspective**: our core objective is to mitigate the **temporal distribution shift in the replay buffer**, a primary driver of gradient magnitude decay and subsequent plasticity loss. Theoretically, our distribution-aware sampling strategy does not overlap with NTK-based plasticity preservation techniques; instead, it offers a complementary approach to addressing the root causes of plasticity loss in deep learning systems.

D Approximate bucket-based sampling

Efficient Approximation via Bucket Sampling To mitigate the computational overhead of recalculating weights for the entire replay buffer, we exploit the **monotonic age property** of our weighting scheme. Since the weights are strictly determined by the temporal age of transitions, we propose a bucket-based approximation method:

1. **Partitioning:** We divide the N transitions in the buffer into B sequential buckets (where $B \ll N$).
2. **Approximation:** Leveraging the monotonicity, we estimate the total weight of each bucket using the weight of its median sample, significantly reducing calculation redundancy.
3. **Hierarchical Sampling:** We first sample a bucket according to the approximated probability distribution, then uniformly sample a transition within that bucket.

As shown in Table 1, this approach reduces the sampling complexity from $\mathcal{O}(N)$ to $\mathcal{O}(B)$. With $B = 2000$ and a buffer size of $N = 10^6$, this yields a theoretical **500$\times$ speedup** in the weight computation phase, rendering the overhead negligible.

Table 1: Computational complexity comparison. N denotes the buffer size (10^6), M the batch size, and B the number of buckets (2000).

Method	**Complexity**	**Scale Dependency**
Uniform Sampling	$\mathcal{O}(M)$	Independent of Buffer
Exact SWD	$\mathcal{O}(N+M)$	Linear w.r.t Buffer
Approximate SWD	$\mathcal{O}(\mathbf{B}+\mathbf{M})$	**Linear w.r.t Buckets**

Empirical Validation We validate the efficiency and effectiveness of this approximation on the `Humanoid-run` task. As presented in Table 2, the Approximate SWD method matches the wall-clock training time of Uniform sampling (approx. 8.7 hours) while preserving the performance gains of the exact method, achieving a high episode return of 224.9 ± 17.5.

Table 2: Runtime and performance comparison on `Humanoid Run`. The approximate method retains performance while significantly reducing training time.

Method	**Wall-Clock Time**	**Episode Return**
Uniform	8.65 h	190.46 ± 7.99
Exact SWD	10.43 h	229.01 ± 37.43
Approximate SWD	**8.70 h**	$\mathbf{224.93 \pm 17.47}$

E EXPERIMENTAL DETAILS

E.1 STRUCTURE

TD3 In this paper, we adopt the official network architecture of Twin Delayed Deep Deterministic Policy Gradient (TD3) for baseline comparison, with detailed layer-wise configurations provided in Table 3.

Table 3: Network Structures of the Twin Delayed Deep Deterministic Policy Gradient (TD3)

Network Component	**Actor Network**	**Critic Network**†
Fully Connected Layer	(state_dim) $\rightarrow$ (256)	(state_dim + action_dim) $\rightarrow$ (256)
Activation	ReLU	ReLU
Fully Connected Layer	(256) $\rightarrow$ (128)	(256) $\rightarrow$ (128)
Activation	ReLU	ReLU
Output Fully Connected Layer	(128) $\rightarrow$ (action_dim)	(128) $\rightarrow$ (1)
Activation	Tanh‡	None

†: TD3 adopts two identical critic networks (Critic 1 & Critic 2) for delayed Q-value update, both following the above structure;
‡: Tanh activation constrains the actor's output action to the range $[-1, 1]$, consistent with standard continuous action space settings.

Double DQN We adopt the Nature CNN network architecture, the detailed specifications of which are presented in Table 4 and Table 5. Our implementation refers to the official code repository[1] to ensure consistency with the original design.

Table 4: Architecture of the Nature CNN Encoder used in Double DQN. The input consists of 4 stacked frames of size 84×84.

Layer	**Input Channels**	**Kernel Size / Stride**	**Output Channels**	**Activation**
Conv1	4	8×8 / 4	32	ReLU
Conv2	32	4×4 / 2	64	ReLU
Conv3	64	3×3 / 1	64	ReLU

Table 5: Architecture of the Double DQN Q-Network. The input is the flattened feature vector from the Encoder.

Layer	**Configuration**	**Activation**
Input (Flatten)	3136 units ($7 \times 7 \times 64$)	-
FC1	Linear($3136 \rightarrow 512$)	ReLU
Output	Linear($512 \rightarrow \lvert\mathcal{A}\rvert$)	-

SAC In this paper, we adopt the same configuration of SimBa as used in the Soft Actor-Critic (SAC) algorithm, with detailed network structures provided in Table 6, Table 7, and Table 8. Our implementation refers to the official SimBa code repository[2] to ensure consistency with the original design.

[1]https://github.com/google-deepmind/dqn
[2]https://github.com/SonyResearch/simba

Table 6: Architecture of the SimBa Residual Block

Layer/Operation	Input/Output Dimensions	Activation Function
Layer Normalization	(hidden_dim) $\rightarrow$ (hidden_dim)	None
Fully Connected (Expansion)	(hidden_dim) $\rightarrow$ (4$\times$hidden_dim)	ReLU
Fully Connected (Compression)	(4$\times$hidden_dim) $\rightarrow$ (hidden_dim)	None
Residual Connection	Input $\oplus$ Block Output*	None

*: "$\oplus$" denotes element-wise addition between the original input and the block output.

Table 7: Architecture of the SimBa Encoder

Component	Structure & Dimension Flow
Input Projection (Fully Connected)	(input_dim) $\rightarrow$ (hidden_dim)
Residual Block Stack	$\times$ num_blocks† (each block follows Table 6)
Final Layer Normalization	(hidden_dim) $\rightarrow$ (hidden_dim)

†: "num_blocks" denotes the number of stacked residual blocks, configurable based on task requirements.

Table 8: Network Structures of the SimBa-SAC Framework

Component	Actor Network	Critic Network
Input Dimension	(state_dim)	(state_dim + action_dim)
SimBa Encoder	hidden_dim=128; num_blocks=1	hidden_dim=512; num_blocks=2
Fully Connected	(128) $\rightarrow$ (action_dim)	(512) $\rightarrow$ (1)
Output Activation	Tanh‡	None

‡: Tanh activation is used to constrain the action output within the range $[-1, 1]$, consistent with standard SAC implementations.

E.2 Implementation Details

Our codes are implemented with Python 3.10 and JAX. All experiments were run on NVIDIA GeForce GTX 3090 GPUs. Each single training trial ranges from 10 hours to 21 hours, depending on the algorithms and environments.

TD3 Implementation Our TD3 implementation refers to CleanRL[3], an efficient and reliable repository for reinforcement learning (RL) algorithm implementations.

Notably, for all OpenAI MuJoCo experiments, we directly use the raw state and reward signals from the environment without any normalization or scaling. To facilitate exploration, an exploration noise sampled from $\mathcal{N}(0, 0.1)$ is added to the action selection process of all baseline methods. The discount factor is set to 0.99, and the Adam optimizer is adopted for all algorithms.

Table 9 presents the complete hyperparameters of TD3 used in our experiments; to reproduce the learning curves reported in the main text, we recommend using random seeds 1 to 5.

Double DQN Implementation Our Double DQN implementation builds upon the CleanRL repository[4], recognized for its high-fidelity and reproducible reference algorithms. To ensure experimental fairness, we strictly align our configuration with standard Atari benchmarks.

The detailed hyperparameters are presented in Table 10. Notably, for the Arcade Learning Environment (ALE) tasks, we incorporate the bucket-based approximate sampling mechanism from `SWD` to enhance efficiency.

[3]https://github.com/vwxyzjn/cleanrl/blob/master/cleanrl/td3_continuous_action.py

[4]https://github.com/vwxyzjn/cleanrl/blob/master/cleanrl/dqn_atari.py

Table 9: Hyperparameters of the TD3 Algorithm

Hyperparameter	**TD3 Configuration**
Actor Learning Rate	10^{-4}
Critic Learning Rate	10^{-3}
Discount Factor	0.99
Batch Size	128
Replay Buffer Size	10^6
SWD-Specific Hyperparameters	
Linear Decay Steps	$100,000$
Minimum Weight (min_weight)	0.1

Table 10: Hyperparameters of Our Double DQN Implementation

Hyperparameter	**Value**
General Training	
Optimizer	Adam
Learning Rate	1×10^{-4}
Discount Factor (γ)	0.99
Buffer Size	1×10^6
Batch Size	32
Learning Starts	$80,000$ steps
Train Frequency	4 steps
Total Timesteps	10 M
Exploration (Epsilon-Greedy)	
Start Epsilon ($\varepsilon_{\text{start}}$)	1.0
End Epsilon (ε_{end})	0.01
Exploration Fraction	0.10 (1×10^6 steps)
Target Network	
Target Update Frequency	1000 steps
Target Update Rate (τ)	1.0 (Hard Update)
SWD-Specific Hyperparameters	
Linear Decay Steps	$80,000$
Minimum Weight	0.1
Number of Buckets	2000

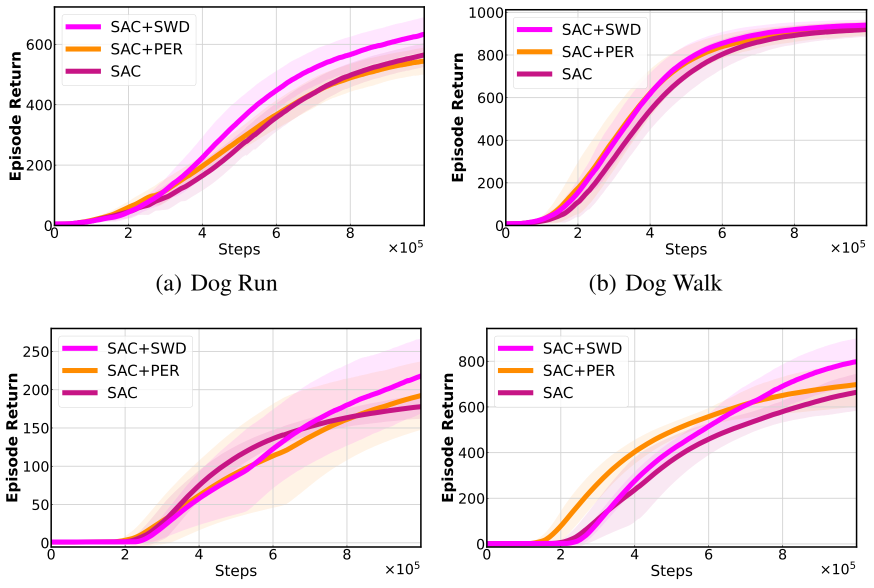
SAC Implementation Our Soft Actor-Critic (SAC) implementation is also based on the CleanRL repository, specifically referencing the continuous action SAC implementation[5].

Table 11: Hyperparameters of Our SAC Implementation (with SimBa Encoder)

Hyperparameter	**SAC (with SimBa Encoder)**
Optimizer	AdamW (weight decay = 10^{-2})
Policy (Actor) Learning Rate	1×10^{-4}
Q-Network (Critic) Learning Rate	1×10^{-4}
Discount Factor	0.99
Batch Size	256
Warmup Steps (for Policy Update)	5000
Target Q-Network Update Rate (τ)	0.005
Target Q-Network Update Interval	1 (step)
Policy (Actor) Update Interval	2 (steps, policy_frequency)
Entropy Target	$-\|A\|$ ($\|A\|$ = action space dimension)
SimBa Encoder (Actor): Hidden Dim / Blocks	128 / 1
SimBa Encoder (Critic): Hidden Dim / Blocks	512 / 2
SWD-Specific Hyperparameters	
Linear Decay Steps	$80,000$
Minimum Weight (min_weight)	0.1
PER-Specific Hyperparameters	
Prioritization Exponent (α)	0.6
Importance Sampling Exponent (β)	0.4
Beta Increment Rate	1×10^{-6}

The hyperparameters for our SAC (equipped with the SimBa encoder) are detailed in Table 11.

E.3 SAC LEANRING CURVE



(a) Dog Run (b) Dog Walk

(c) Humanoid Run (d) Humanoid Walk

Figure 9: SAC leanrning curve on DMC tasks

[5]https://github.com/vwxyzjn/cleanrl/blob/master/cleanrl/sac_continuous_action.py

F Additional Experiments

F.1 SWA

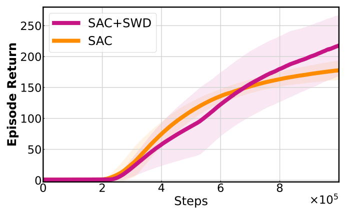
In this section, we provide detailed information about our ablation experiments. First, we present the algorithmic details of SWA, which are summarized in Algorithm 2.

We adopt the detailed parameter settings of Soft Actor-Critic (SAC), as presented in Table 11—specifically, we use the same Linear decay steps T and minimum weight $w_{\min}$ as specified therein.

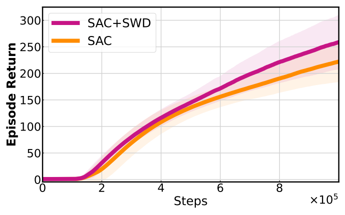
Algorithm 2 SWA

```
Require: Linear decay steps $T$, minimum weight $w_{\min}$, Current time $t$, timestamps $\{t_i\}_{i=1}^{|\mathcal{D}|}$
1: for $i = 1$ to $|\mathcal{D}|$ do
2:     $age_i = t - t_i$
3:     $w_i = \min\left(1, w_{\min} + \frac{age_i}{T}\right)$
4: end for
5: $p_i = \frac{w_i}{\sum_{j=1}^{|\mathcal{D}|} w_j}$ for $i = 1, \ldots, |\mathcal{D}|$
6: $\mathcal{I} \sim$ Categorical$(\{p_i\}_{i=1}^{|\mathcal{D}|}, B)$
7: return $\mathcal{B} = \{(s_i, a_i, r_i, s'_i, d_i)\}_{i \in \mathcal{I}}$
```

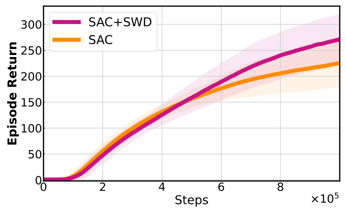
F.2 Ablation Study of Update-to-Data



(a) UTD=1



(b) UTD=2



(c) UTD=5

Figure 10: Sensitivity analysis regarding the UTD. Data represents the mean $\pm$ std of five experimental runs conducted on the Humanoid Run.

We adopt SAC (Soft Actor-Critic) as the backbone algorithm and aim to optimize the Update-to-Data (UTD) ratio. This optimization enables faster policy iteration, thereby better leveraging the advantages of SWD. As illustrated in Figure 10, with the increase in the UTD ratio, SWD consistently outperforms the uniform sampling baseline.

F.3 Parameter Sensitivity Analysis

To assess the robustness of our proposed method, we conducted an extensive grid search to evaluate the sensitivity of SWD to its two primary hyperparameters: the linear decay steps (T) and the minimum weight threshold ($w_{\min}$).

We constructed a 5×5 hyperparameter grid, varying T_{decay} from $20,000$ to $100,000$ and $w_{\min}$ from 0.02 to 0.10. Experiments were performed on the `Humanoid Run` task, with each of the 25 configurations averaged over 5 random seeds (totaling 125 independent runs). The results, summarized in Table 12, indicate that SWD maintains stable performance across a wide range of hyperparameter settings. While optimal performance fluctuates slightly, the method does not exhibit drastic failure modes within the tested range, demonstrating its robustness to hyperparameter selection.

Table 12: **Parameter Sensitivity Analysis.** Grid search results on `Humanoid Run` (Mean ± Std). The best performance is marked in **bold**.

Decay Steps	**Minimum Weight Threshold** ($w_{\min}$)				
(T_{decay})	**0.02**	**0.04**	**0.06**	**0.08**	**0.10**
20,000	229.7 ±26.4	**240.9** ±37.1	234.9 ±15.0	217.9 ±38.6	226.1 ±23.7
40,000	231.4 ±44.4	224.5 ±34.4	231.3 ±30.8	227.0 ±23.1	225.5 ±22.5
60,000	217.4 ±40.2	231.2 ±29.9	**240.5** ±55.0	215.7 ±27.9	**240.7** ±35.3
80,000	233.6 ±42.9	231.8 ±35.7	225.2 ±42.6	220.9 ±17.6	231.3 ±54.4
100,000	224.0 ±32.1	201.8 ±31.9	217.0 ±48.5	**241.6** ±38.4	229.2 ±29.5

F.4 Impact of Decay Strategy

We further investigate the influence of the weight decay schedule on performance. To this end, we compare our default **Linear Decay** against **Exponential** and **Polynomial** variants. The specific formulations are defined as follows:

- **Linear (Ours):** $w(t) = \max(w_{\min}, 1 - t/T)$, providing a constant rate of importance reduction.
- **Exponential:** $w(t) = \max(w_{\min}, \exp(-t/\tau))$, where $\tau = 1$, modeling rapid initial forgetting.
- **Polynomial:** $w(t) = \max(w_{\min}, (1-t/T)^p)$, where $p = 2$, penalizing older samples more aggressively than the linear approach.

The empirical results on `Humanoid-run` are summarized in Table 13. Our proposed Linear Decay strategy significantly outperforms alternative schedules. Notably, both Exponential and Polynomial decay perform worse than the SAC baseline, suggesting that overly aggressive weight reduction disrupts the learning stability required for high-dimensional control tasks.

Table 13: Performance comparison of different decay strategies on `Humanoid Run`. The relative difference is calculated with respect to our Linear Decay method.

Decay Strategy	**Episode Return**	**vs. Linear SWD**
Linear Decay (Ours)	**229.01 ± 37.43**	–
SAC (Baseline)	190.46 ± 7.99	−16.8%
Exponential Decay ($\tau = 1$)	187.04 ± 29.85	−18.3%
Polynomial Decay ($p = 2$)	132.91 ± 11.12	−42.0%