Offline Reinforcement Learning at Multiple Frequencies

Kaylee Burns[1], **Tianhe Yu**[1,2], **Chelsea Finn**[1,2], **Karol Hausman**[1,2]
[1]Stanford University [2]Google Research
kayburns@cs.stanford.edu

Abstract: To leverage many sources of offline robot data, robots must grapple with the heterogeneity of such data. In this paper, we focus on one particular aspect of this challenge: learning from offline data collected at different control frequencies. Across labs, the discretization of controllers, sampling rates of sensors, and demands of a task of interest may differ, giving rise to a mixture of frequencies in an aggregated dataset. We study how well offline reinforcement learning (RL) algorithms can accommodate data with a mixture of frequencies during training. We observe that the Q-value propagates at different rates for different discretizations, leading to a number of learning challenges for off-the-shelf offline RL algorithms. We present a simple yet effective solution that enforces consistency in the rate of Q-value updates to stabilize learning. By scaling the value of N in N-step returns with the discretization size, we effectively balance Q-value propagation, leading to more stable convergence. On three simulated robotic control problems, we empirically find that this simple approach significantly outperforms naïve mixing both in terms of absolute performance and training stability, while also improving over using only the data from a single control frequency.

Keywords: offline reinforcement learning, robotics

1 Introduction

Given the cost of robotic data collection, we would like robots to learn from all possible sources of data. However, the robot data that is available may be heterogeneous, which can present challenges to offline reinforcement learning (RL) algorithms. One particular form of variability that we focus on in this work is the control frequency. Data collected independently may have different frequencies due to different sensor sampling rates or how the data was collected. For example, data collected through teleoperation may have a higher frequency to accommodate a more reactive and intuitive user interface, while data collected via online RL may use lower frequencies for more stable learning [1]. In light of the benefits from learning from large and diverse datasets [2, 3], our goal in this paper is to study whether offline RL algorithms can effectively learn from data with multiple underlying frequencies, and improve over learning from a subset of frequencies.

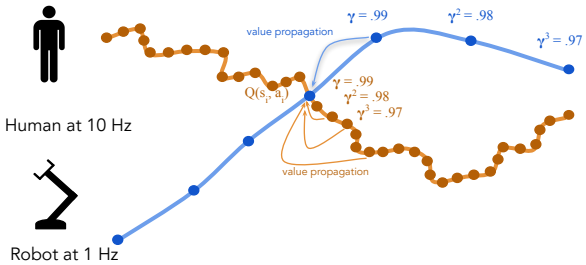

Figure 1: Training offline RL with diverse frequencies is challenging because the discrete MDP depends on the size of the timestep in between observations. This discretization affects the rate of value propagation, with smaller discretizations requiring more training updates to propagate the values.

It is known that RL at high frequencies can lead to instability; indeed, prior work has presented solutions to such instability [1]. However, much less attention has been paid to learning from data with multiple frequencies. Work on time-series analysis has studied how to handle irregularly sampled data [4, 5], but not in the context of the approximate dynamic programming updates that arise in

value-based reinforcement learning settings. As we will find in Section 4, running offline RL algorithms on data with multiple control frequencies leads to unstable training and diminished performance, a problem that, to our knowledge, has not been analyzed or addressed in previous research.

A key insight in this work is that Q-learning from different data discretizations leads to different rates of value propagation: value propagates more quickly for coarser time discretizations. Optimizing with different rates can lead to high variance updates and training instability. Fortunately, we can use knowledge of the time discretization to normalize the rate of value propagation across different discretizations: our updates can look ahead to farther Q-values for data with a smaller discretization. This intuition leads us to a simple adjustment to N-step returns [6, 7] to align the discretizations, where we adaptively select the value of N based on the time discretization of the particular trajectory.

The main contribution of this paper is to analyze and provide a solution to the problem of offline reinforcement learning from data with multiple time discretizations. We show that naïvely mixing data of different discretizations leads to diminished performance, and posit that this issue is caused by varying rates of value propagation for different discretizations. We provide a simple technique that addresses this challenge via adaptive N-step returns. On three simulated robotic control problems, including two manipulation problems on simulated Sawyer and Panda robot arms, we find that our adaptive N-step update rule leads to more stable training and significantly improved performance, compared to naïve mixing or only using the subset of data that has a single discretization.

2 Related Work

Supervised learning of irregular time series. Datasets with varying frequencies can arise whenever observations are pulled inconsistently from a continuous time system. Some approaches focus on consuming irregularly sampled data [8, 4, 5] by modeling the latent state of the data-consuming process as a differential equation. Unlike past work that integrates continuous time models into control tasks [9, 10], we are focused on the setting where the discretization remains constant within each trajectory, but varies within batch updates.

Continuous time RL. We focus on control tasks learned with offline RL, so in addition to learning a policy that can operate at multiple time scales, we also need to learn a value function at multiple time scales. Past work [11, 12, 13] uses the Hamilton-Jacobi Bellman equation to derive algorithms for estimating the value function. These formalisms have been used to create algorithms that can handle settings where the environment evolves concurrently with planning [14] and to learn to budget coarse and fine-grained time scales during training [15]. In addition to modeling challenges, continuous time introduces challenges during training as well. Tallec et al. [1] shows that deep Q-learning can fail in near-continuous time (i.e., fine-grained) settings and uses the formalisms from continuous RL to make Q-learning more reliable for small discretizations by using an advantage update.

Offline RL. Offline RL [16, 17, 18, 19] has emerged as a promising direction in robotic learning with the goal of learning a policy from a static dataset without interacting with the environment [20, 21, 22, 23]. Prior offline RL approaches focus on mitigating the distributional shift between the learned policy and the data-collecting policy [24] via either explicit or implicit policy regularization [25, 26, 27, 28, 29, 30, 31], penalizing value backup errors [32], uncertainty quantification [26, 33, 34], and model-based methods [35, 36, 37, 38, 39]. None of these works consider the practical problem of learning from offline data with different control frequencies, which is the focus of this work.

3 Preliminaries

A discrete-time MDP is a process that approximates an environment that is fundamentally continuous. For example, a robot may be moving continuously but receiving observations and sending actions at a fixed sampling rate, $1/\delta t$, where δt is the time between the observations. A discrete-time MDP that takes into account δt can be derived by discretizing a continuous-time MDP [12, 1].

Given a time-discretization δt, we arrive at the discrete MDP [1] $M_{\delta t} = (\mathcal{S}, \mathcal{A}, T_{\delta t}, r, \gamma)$, where $T_{\delta t}(s'|s, a)$ denotes the transition function over the timestep δt, $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces, r the reward function, and $\gamma \in (0, 1)$ the discount factor. The cumulative discounted reward of this MDP can be written as: $R_{\delta t} := \sum_{k=0}^{\infty} \gamma^{k\delta t} r(s_k, a_k)\delta t$; and the δt-dependent Q-function can

be written as: $Q^{\pi}_{\delta t} = r(s,a)\delta t + \gamma^{\delta t}\mathbb{E}_{\tau\sim\pi,T_{\delta t}}\left[\sum_{k=0}^{\infty}\gamma^{k\delta t}r(s_k,a_k)\delta t|s_0=s\right]$. We will use the δt-dependent rewards and Q-functions to analyze the case of mixing data with different discretizations, i.e. with different values of δt, in offline RL.

Offline RL with conservative Q-learning. Offline RL tackles the problem of learning control policies from offline datasets where the main challenge is the distribution shift between the learned policy and the behavior policy that was used to collect the data. A common offline RL approach that mitigates this issue focuses on an additional pessimism loss that encourages the Q-value update to stay close to the actions that it has seen in the data. One popular instantiation of this idea is conservative Q-learning (CQL) [32], which optimizes the following objective: $Q^{k+1} = \arg\min_Q\Big[\alpha\cdot\big(\mathbb{E}_{s\sim\mathcal{D},a\sim\mu(a|s)}[Q(s,a)] - \mathbb{E}_{s\sim\mathcal{D},a\sim\hat{\pi}(a|s)}[Q(s,a)]\big) + \frac{1}{2}\mathbb{E}_{s,a,s'\sim\mathcal{D}}\left[\big(Q(s,a) - r(s,a) - \gamma\mathbb{E}_{\pi}(a|s)[Q^k(s,a)]\big)\right]\Big]$, where $\mathcal{D}$ is the offline dataset of states, actions, and rewards, μ is a wide action distribution close to the uniform distribution, and $\hat{\pi}$ is the behavior policy that collects the offline data. We will show how we can use a modified version of CQL to incorporate data from different discretizations in offline RL.

4 Mixed Discretizations Can Destabilize Offline RL

We study how mixing frequencies naïvely creates instability during offline RL training. We introduce the problem of learning from a distribution of different frequencies and describe how the differing rates of value propagation can create instability during training and empirically verify this intuition on a simple task.

Problem setup. Our objective is to learn a policy or set of policies that maximizes the expected return over a dataset that contains a mixture of different observation frequencies. Concretely, our goal is to learn $\pi^*(a|s,\delta t)$ over a distribution, Δ, of discretizations: $\pi^*_{\delta t}(a|s) = \pi^*(a|s,\delta t) = \arg\max_{\pi}\mathbb{E}_{\delta t\sim\Delta}\left[\mathbb{E}_{\pi,T_{\delta t}}\left[R_{\delta t}\right]\right]$. To accomplish this goal, we utilize offline RL and focus our analysis on the behavior of Q-values in the presence of mixed discretizations. Specifically, we are interested in studying the objective $\mathbb{E}_{\delta t\sim\Delta}\left[\mathcal{L}_Q\right]$, where $\mathcal{L}_Q$ is any loss based on the Bellman equation.

We use CQL as our offline RL algorithm of choice, but our analysis is applicable to other value-based offline RL methods. We modify the CQL objective described in Sec. 3 to incorporate a distribution of discretizations, δt as follows:

$$Q^{k+1}_{\delta t} = \arg\min_Q \mathbb{E}_{\delta t\sim\Delta}\Bigg[\alpha\cdot\big(\mathbb{E}_{s\sim\mathcal{D}_{\delta t},a\sim\mu(a|s)}[Q_{\delta t}(s,a)] - \mathbb{E}_{s\sim\mathcal{D}_{\delta t},a\sim\hat{\pi}_{\delta t}(a|s)}[Q_{\delta t}(s,a)]\big) + \frac{1}{2}\mathbb{E}_{s,a,s'\sim\mathcal{D}_{\delta t}}\left[\big(Q_{\delta t}(s,a) - r(s,a)\delta t - \gamma^{\delta t}\mathbb{E}_{\pi_{\delta t}}(a|s)[Q^k_{\delta t}(s,a)]\big)^2\right]\Bigg]$$

The Q-targets of different δt may receive value updates at different rates because it takes longer for value to propagate when the state space is divided by more fine-grained steps. We study the challenges this creates for Q-learning next. We refer to this unmodified objective as Naïve Mixing.

Value propagates in algorithmic time, not physical time. A key challenge in mixing data of different discretizations together is that algorithmic time, k, scales inversely with the size of the discretization: $k = \frac{t}{\delta t}$. This implies that value propagates along the state space at a rate that is proportional to the discretization. This phenomenon is illustrated for a simple gridworld environment in Figure 2. The agent starts on the right side and receives a reward of 10 for moving forward to the left. In the top row, the agent can take steps of size 1 and in the bottom row the agent can take steps of size 2, which simulates policies that operate at different frequencies. At every step of the algorithm, k, we perform the update rule $V^{\delta t}_k(s) = \max_a r(s,a) + \gamma V^{\delta t}_{k-1}(s')$. Although in this toy example, we are performing tabular updates, we can see how the nature of the Bellman update generates a distribution of targets that is difficult for a function approximator, like a neural network, to match. For example, at step $k=3$, a network would receive the following targets for the first three states from the start: 0 for $\delta t = 1$ and 5.3 for $\delta t = 2$ for the first and second state, and 0 for $\delta t = 1$ and 6.6 for $\delta t = 2$ for the third state. We posit that the rate at which the targets are updated for different discretizations and the resulting difference in the target values leads to training instability and subpar performance. In the next section, we verify this intuition empirically.

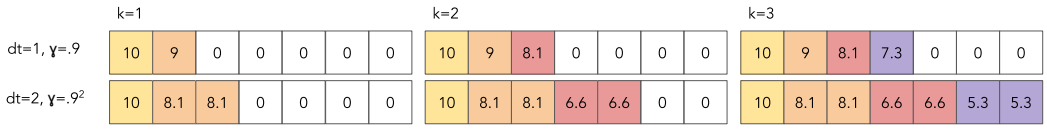

Figure 2: We visualize how value propagates for different δt in a simple gridworld. In the top row, the agent takes actions of length 1 and in the bottom of length 2. For each update step k, the value propagates twice as much in the second setting, creating regression targets that are difficult to match. For example, at step $k = 2$, the Q-value needs to regress to targets 6.6 and 0 from the same states. Boxes are color-coded by update step.

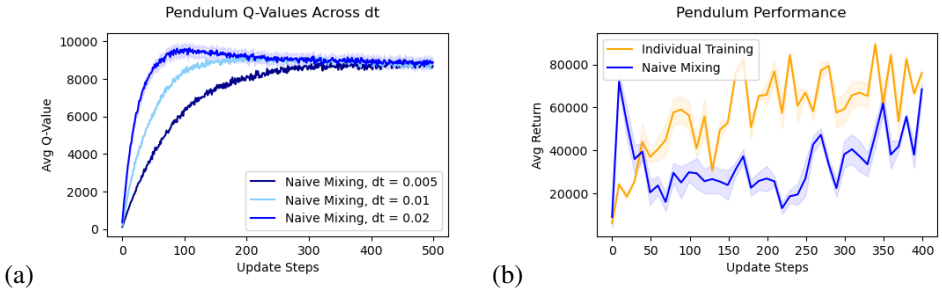

Figure 3: We analyze training with mixed discretizations for the pendulum swing up task. (a) Over the course of training, the average Q-value scales with δt. In result, the Q-value of more fine-grained discretizations propagates more slowly than for coarser discretizations. (b) Compared to training individually, the final performance of a policy trained for $\delta t = 0.02$ is corroded by adding more data of different discretizations.

4.1 Analyzing Naïve Mixing

We study how offline RL behaves when naïvely mixing discretizations in a pendulum environment and posit that differing rates of Q-function updates lead to training instability that hurts performance.

Setup. To test our hypothesis, we use the previously-used simple pendulum environment from the OpenAI Gym [1]. We modify the pendulum environment by changing the underlying time discretization of the simulator and collect data with discretizations $\delta t \in \{0.02, 0.01, 0.005\}$ seconds. We make the reward function for this task sparse by creating a window of angles and velocities around 0 where the algorithm receives a constant reward. We first collect data by running Deep Advantage Updating (DAU) [1] for each discretization individually and then store buffers of $500k$ observations per discretization. The data collected is comparable to an expert-replay dataset. We combine the data coming from the three different discretizations together into a single offline dataset.

Value propagation. We look at the average Q-value for different discretizations over the course of training. The average cumulative reward in the dataset for each δt is similar, so we expect the average Q-value of each discretization to be the same. However, because more fine-grained data propagates the sparse reward at a slower rate than for coarser data as shown in the previous subsection, we see in Figure 3 (a) that the coarser discretization of $\delta t = 0.02$ reaches a higher average Q-value more quickly than a fine-grained discratization of $\delta t = 0.01$, which in turn reaches a higher Q-value more quickly than the most fine-grained discretization $\delta t = 0.005$. By the end of training the average Q-values across different δt converge. This confirms the hypothesis that the Q-values grow at different rates because of the nature of the Bellman update and not because the true value is different.

Optimization challenges. Next, we want to understand if the different rates of the Q-value update for different discretizations cause optimization challenges during offline RL. In Figure 3 (b) we plot performance over the course of training with and without adding data. Ideally our algorithms would be aided by the addition of more data, but after introducing data of different frequencies, we see the average return across training drop.

To summarize, these results suggest that, even within a simple control task, naïvely mixing data with different discretizations hinders the performance by slowing value propagation and introducing optimization challenges. Next, we present a method that aims to avoid these issues.

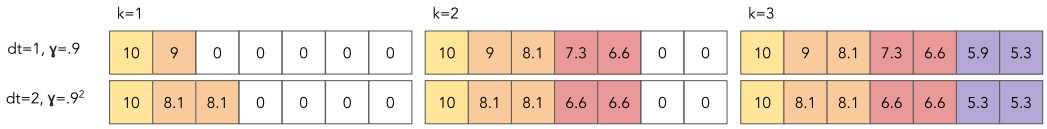

Figure 4: We modify value iteration on a simple gridworld environment. In the bottom row, the agent can take actions that are double the size of the other agents actions. However, because the agent in the top row uses an N-step update rule with $N = 2$, the value propagates along the state space at the same rate for each agent.

5 Adaptive N-Step Returns

The goal of our method is to "align" the Bellman update rate for different discretizations so that they can be updated at a similar pace, creating more consistent value targets. Intuitively, we aim to mimic the "update stride" of coarser discretizations when training on more fine-grained discretizations to bring consistency to the Q-value updates.

The core idea behind our approach is to utilize N-step returns as a tool that accelerates the propagation of Q-values. In particular, we calculate Q-targets with adaptive N-step returns, where we scale N by the value of δt. To better understand the idea behind this approach, we revisit the same intuition as previously presented in Figure 2, but with the updated N-step modifications in Figure 4. In this case, we use the following value iteration update rule: $V_k(s_t) = \sum_{t'=t}^{\frac{N}{\delta t}-1+t} \gamma^{t'-t} r(s_{t'}, a_{t'}) + \gamma^{\frac{N}{\delta t}} V_{k-1}(s_{t+\frac{N}{\delta t}})$. This update rule is equivalent to N-step returns for value iteration, but instead of a fixed N, we use an N scaled by δt. In Figure 4 we pick $N = 2$. In the top row we sum the reward over the next $\frac{N}{\delta t} = \frac{2}{2} = 1$ transitions and in the bottom row we sum over the next $\frac{N}{\delta t} = \frac{2}{1} = 2$ transitions. While there are still slight differences in the target values due to the precision allowed by the discretization, the differences in the targets available to a function approximator are significantly smaller in magnitude than in the previous case presented in Figure 2.

We can easily apply this approach to Q-learning resulting in a simple Q-value update rule that can be integrated into any offline RL algorithm. At every update step, we calculate the target Q at δt as:

$$\sum_{t'=0}^{\frac{N}{\delta t}-1} \gamma^{t'} r(s_{t+t'}, a_{t+t'}) + \gamma^{\frac{N}{\delta t}} Q^{\delta t}(s_{\frac{N}{\delta t}}, a_{\frac{N}{\delta t}}) \tag{1}$$

In all of our experiments, we select N to be the largest δt present in the dataset (i.e., the most coarse discretization). With this choice of N, the more fine-grained discretizations exactly match the update stride of the coarsest discretization under standard offline RL.

Although this update rule could be integrated into any Q-learning algorithm, we use CQL in our experiments. This requires adapting both the standard Bellman loss as well as the pessimism and optimism terms. We replace the Bellman operator in Equation 1 with N-step returns scaled by discretization. The pessimism and optimism terms are applied to the state and action at $\frac{N}{\delta t}$, where the target Q-function is evaluated. This results in the complete objective:

$$\mathcal{L}_{\text{cql}} = \mathbb{E}_{s\sim\mathcal{D}_{\delta t}, a\sim\mu(a|s)}[Q^{\delta t}(s_{t+\frac{N}{\delta t}}, a_{t+\frac{N}{\delta t}})] - \mathbb{E}_{s\sim\mathcal{D}_{\delta t}, a\sim\hat{\pi}(a|s,\delta t)}[Q^{\delta t}(s_{t+\frac{N}{\delta t}}, a_{t+\frac{N}{\delta t}})]$$

$$\mathcal{L} = \arg\min_{Q} \mathbb{E}_{\delta t\sim\Delta}\left[\alpha \cdot \mathcal{L}_{\text{cql}} + \frac{1}{2}\mathbb{E}_{s_t,a_t,s_{t+1}\sim\mathcal{D}_{\delta t}}\left[\left(Q^{\delta t}(s_t, a_t) - Q^{\delta t}_{\text{target}}(s_t, a_t)\right)^2\right]\right]$$

When $\frac{N}{\delta t}$ is not an integer, we can sample the number of steps from the nearest digits with a Bernoulli distribution where p is the fractional part of $\frac{N}{\delta t}$.

6 Experiments

In Section 4, we used a simple RL problem to demonstrate how naïve mixing of data at different discretizations updates the Q-function at different rates leading to instability in training. In our experiments, we aim to confirm this trend on popular benchmarks and verify that our method proposed in Section 4 can remedy these challenges. In particular, we seek to answer the following questions:

Env Name	δt	Naïve mixing	Adaptive N-step (ours)
pendulum	.02	64.8 ± 8.4	**91.1** ± 3.7
	.01	57.5 ± 10.5	**81.5** ± 2.4
	.005	37.5 ± 3.5	**62.0** ± 6.3
	Avg	53.3 ± 7.5	**78.2** ± 4.2
door-open (max success)	10	**100.0** ± 0.0	89.5 ± 8.1
	5	**95.0** ± 3.9	92.9 ± 5.7
	2	**100.0** ± 0.0	**100.0** ± 0.0
	1	5.7 ± 4.2	**87.5** ± 7.5
	Avg	75.2 ± 2.0	**92.5** ± 5.3
kitchen-complete-v0	40	20.2 ± 5.7	**34.6** ± 8.7
	30	9.3 ± 4.4	**19.9** ± 7.2
	Avg	14.7 ± 5.0	**27.3** ± 7.9

Table 1: We compare the performance of naïve mixing against adaptive N-step. Within the sparse reward environments of pendulum and kitchen, naïve mixing corrodes performance across δt and this drop in performance is recovered by adaptive N-step. For the dense reward door-open task, only the finest-grained discretization is hindered by mixing and N-step returns recovers performance on this discretization while maintaining strong performance on coarser discretizations.

(1) Does adaptive N-step returns fix the performance challenges introduced by mixing different discretizations? (2) What impact does adaptive N-step returns have on the learned Q-value during training? (3) What is the influence of our method on training stability? (4) Does mixing N-step allow us to leverage more data sources to build a model that works better on all discretizations?

6.1 Experimental Setup

Tasks and datasets. We evaluate our method on three different simulation environments of varying degrees of difficulty described below.

Pendulum. Similarly to Section 4, we used a modified Open AI Gym [40] pendulum environment. We modify it by changing the underlying discretization of the simulator and collect data with disctretizations of $0.02, 0.01, 0.005$ by training DAU [1] on each discretization independently and storing the replay buffer, which contains $500k$ observations. To aid visualizations and ease debugging, we make the task sparse by creating a window of angles and velocities around 0 where the agent receives a constant reward of 100, scaled by δt. We condition the Q-value and policy on the discretization to train a single RL algorithm on all discretizations.

Meta-World. We modify the Meta-World [41] environments to support different discretizations by adjusting the frame-skip. We experiment with frame skips of $1, 2, 5$, and 10. We focus on the door-open because it is a straightforward manipulation task. In this environment, we find that not conditioning on δt to give stronger performance for our method as well as the baselines. To evaluate performance, we look at the maximum success score, which measures whether or not the environment was in the goal configuration at any point during the trajectory.

FrankaKitchen. FrankaKitchen [42] is a kitchen environment with a 9-DoF Franka robot that can interact with 5 household objects. We study the long-horizon task sequence of opening the microwave, moving the kettle, turning on a light, and opening a drawer with a sparse reward. Upon completion of any of the four tasks, the agent receives a reward of 1. We mix data with frame skips of 30 and 40. The data at 40, which is the default frame skip, comes from the D4RL dataset [43]. Specifically, we use the expert data called `kitchen-complete-v0`. To generate data at a frame skip of 30, we run a trained policy at that frame skip. The quality of this policy is lower, making the data at 30 most similar to a medium-replay dataset.

Implementation and training details. We build on top of the CQL implementation from Geng [44], and use the following CQL hyperparameters: all of our models use a CQL alpha term of 5, a policy learning rate of $3e-5$, and a Q-function learning rate of $3e-4$. For Pendulum we use a discount scaled with the size of δt, specifically $.99^{\frac{\delta t}{\max_{\delta t \in \Delta} \delta t}}$. For Meta-World, we found a constant discount

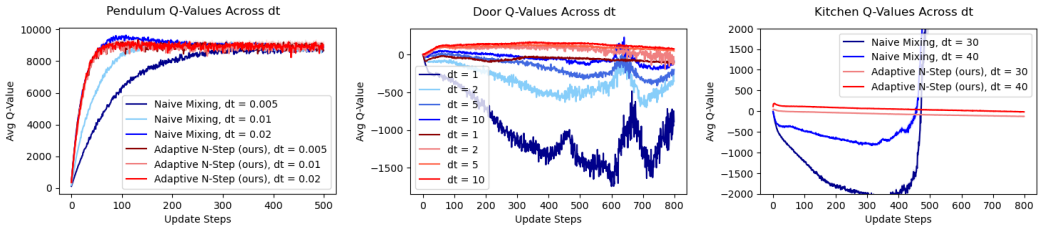

Figure 5: Adaptive N-step results in more consistent q-values across discretizations and tasks. In all figures, blue denotes naive mixing and red denotes Adaptive N-step returns.

factor of .99 to work best across the experiments. Each task has a maximum trajectory length of 500 regardless of the discretization size. We implement $Q^{\delta t}$ as an optionally δt-conditioned neural network by adding a normalized δt feature to the state.

6.2 Results

Final performance with and without N-step. To answer question (1) (*Does δt-scaled N-step returns fix the performance challenges introduced by mixing different discretizations?*), we compare the normalized performance at the end of training with and without N-step modification in Table 1. Results are averaged and reported with standard error over runs from 5 seeds with 5 evaluations from each run. For the sparse reward tasks – pendulum and kitchen – δt scaled N-step significantly improves final training performance across discretizations, resulting in an almost double average return in the case of kitchen, and more than 50% improvement in pendulum. In the dense reward tasks, i.e. door-open, N-step provides an advantage for more fine-grained discretizations without compromising the high performance naïve mixing already achieves on larger discretizations. These results confirm that our method improves mixing discretization performance across the benchmark tasks. Next, we aim to understand its impact on important quantities such as Q-values.

Q-values across training. In Section 4, we observed that the Q-value learned with the naïve mixing strategy is updated differently for different discretizations. We aim to confirm this hypothesis on other, more complex environments as well as analyze whether our adaptive N-step method can remedy this issue and lead to more consistent Q-value updates. To accomplish this goal, we present Fig. 5, where we visualize the average Q-value for all discretizations with and without adaptive N-step method. When we compare the Q-values after using adaptive N-step, we observe that the Q-values become more consistent across discretizations. In the top left of Figure 5, we see that the Q-values obtained for pendulum are nearly identical for different discretizations across the course of training, resolving the issue pointed out in Sec. 4. Interestingly, we also note that the Q-values of naïve mixing eventually converge to the same value as N-step, but the performance of the policy learned with these values is still diminished as previously described in Table 1. This suggests that the consistency of the updates across training and not just the absolute value of the Q-function are important in learning a good policy. In the top right and bottom left plots in Fig. 5, we see that, in Meta-World, adaptive N-step returns has two effects. It makes the average Q higher for all discretizations and minimizes the difference across discretizations similarly to the pendulum experiment. Within Kitchen, N-step has the same effect of pulling up the Q-values and has the added effect of minimizing the standard error of the Q-value across seeds. With naïve mixing, on the other hand, the Q-values are inconsistent throughout training and, contrary to pendulum, they end up exploding by the end of training. These experiments indicate a positive, stabilizing impact of adaptive N-step on the learned Q-values during training, answering question (2).

Performance across training. To answer question (3) (*What is the influence of our method on training stability?*), we plot the evaluation performance over the course of training averaged over all discretizations in each environment. The results for Pendulum, Meta-World Door, and Kitchen tasks are visualized in Figure 6. For Kitchen and Pendulum, reward corresponds to task success so we plot the average reward. For the Door task in Meta-World, to better reflect the task, we plot the maximum success attained in the evaluation trajectory. We observe a significant difference in

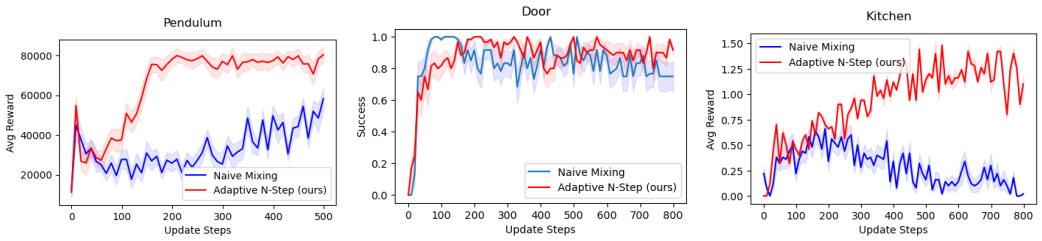

Figure 6: Without adaptive N-step returns, naïve mixing can suffer from instability.

performance and training stability in Pendulum and Kitchen when comparing adaptive N-step and naïve mixing. In particular, the Kitchen task indicates that naïve mixing experiences substantial instability in training, where around 200 steps into the training the performance goes down. This showcase that the performance is unstable even when a reasonable policy can be learned at some point during training. The performance plot for the Door task shows comparable performance when all discretizations are averaged together, however, we refer back to Table 1 to show that certain discretizations (i.e. $\delta_t = 1$) perform significantly worse than others compared to the adaptive N-step strategy.

Comparison to individual training. Following our motivation for this work, we investigate question (4), whether mixing multiple discretizations with our method outperforms training on a single discretization. In mixing diverse data sources together we hope to leverage the data from each source to learn a model that performs better than a set of independently trained models. We perform this experiment on the kitchen task and present the results in Fig. 6.2. The results clearly indicate that training individually on a single discretization on Kitchen data underperforms relative to training with adaptive N-step returns on both discretizations. Interestingly, for the data at 30, which is of lower quality, mixing discretizations with N-step returns also leads to more stable training than training individually on a single discretization.

7 Limitations

Although our adaptive N-step method works reliably for discretizations where the step sizes can be aligned, it is unclear how well adaptive N could work when the frequencies do not have a small common multiple. In the kitchen setting, we saw that the frequencies do not need to be multiples of one another to reap the benefits of adaptive N-step returns, but this strategy could become less viable as the least common multiple between frequencies grows. Other strategies, such as sampling different N with probability proportional to the fractional difference of the two frequencies may need to be adopted. Another limitation of this work is that the discretization of each trajectory is assumed to be fixed and given. If the frequency of observations varies within a trajectory, the value of N may need to be predicted from observations.

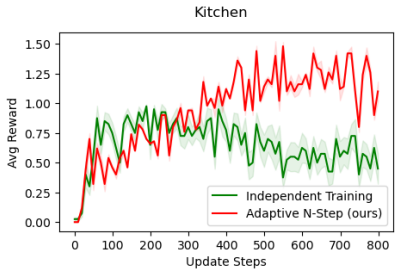

Figure 7: Adaptive N-step improves over training on a single discretization, indicating that a policy at one discretization can benefit from data in another discretization.

8 Conclusion

In this work, we studied the problem of mixing data from different discretizations in offline RL and presented a simple and effective approach for this problem that improved training stability and final performance across three simulated control tasks. We found that our adaptive N-step returns method counteracts varying rates of value propagation present when mixing data naïvely. While our approach provides a step towards offline RL methods that can consume data coming from diverse sources, more future work is needed to address training offline RL agents on other sources of variation that are commonly present in robotics data.

Acknowledgments

If a paper is accepted, the final camera-ready version will (and probably should) include acknowledgments. All acknowledgments go at the end of the paper, including thanks to reviewers who gave useful comments, to colleagues who contributed to the ideas, and to funding agencies and corporate sponsors that provided financial support.

References

[1] C. Tallec, L. Blier, and Y. Ollivier. Making deep q-learning methods robust to time discretization. In *International Conference on Machine Learning*, pages 6096–6104. PMLR, 2019.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[4] P. Kidger, J. Morrill, J. Foster, and T. Lyons. Neural controlled differential equations for irregular time series. *Advances in Neural Information Processing Systems*, 33:6696–6707, 2020.

[5] A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Ré. Hippo: Recurrent memory with optimal polynomial projections. *Advances in Neural Information Processing Systems*, 33:1474–1487, 2020.

[6] C. Watkins. Learning from delayed rewards. 01 1989.

[7] J. Peng and R. Williams. Incremental multi-step q-learning. *Machine Learning*, 22, 09 1998. doi:10.1007/BF00114731.

[8] Y. Rubanova, R. T. Chen, and D. K. Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. *Advances in neural information processing systems*, 32, 2019.

[9] S. Singh, F. M. Ramirez, J. Varley, A. Zeng, and V. Sindhwani. Multiscale sensor fusion and continuous control with neural cdes. *arXiv preprint arXiv:2203.08715*, 2022.

[10] S. Bahl, M. Mukadam, A. Gupta, and D. Pathak. Neural dynamic policies for end-to-end sensorimotor learning. In *NeurIPS*, 2020.

[11] R. Munos and P. Bourgine. Reinforcement learning for continuous stochastic control problems. *Advances in neural information processing systems*, 10, 1997.

[12] K. Doya. Reinforcement learning in continuous time and space. *Neural computation*, 12(1): 219–245, 2000.

[13] J. Kim and I. Yang. Hamilton-jacobi-bellman equations for q-learning in continuous time. In *Learning for Dynamics and Control*, pages 739–748. PMLR, 2020.

[14] T. Xiao, E. Jang, D. Kalashnikov, S. Levine, J. Ibarz, K. Hausman, and A. Herzog. Thinking while moving: Deep reinforcement learning with concurrent control. *International Conference on Learning Representations*, 2020.

[15] T. Ni and E. Jang. Continuous control on time. In *ICLR 2022 Workshop on Generalizable Policy Learning in Physical World*, 2022. URL https://openreview.net/forum?id=BtbG3NT4y-c.

[16] D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 2005.

[17] M. Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pages 317–328. Springer, 2005.

[18] S. Lange, T. Gabel, and M. A. Riedmiller. Batch reinforcement learning. In *Reinforcement Learning*, volume 12. Springer, 2012.

[19] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[20] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, pages 651–673. PMLR, 2018.

[21] R. Rafailov, T. Yu, A. Rajeswaran, and C. Finn. Offline reinforcement learning from images with latent space models. *Learning for Decision Making and Control (L4DC)*, 2021.

[22] A. Singh, A. Yu, J. Yang, J. Zhang, A. Kumar, and S. Levine. Cog: Connecting new skills to past experience with offline reinforcement learning. *arXiv preprint arXiv:2010.14500*, 2020.

[23] A. Kumar, A. Singh, S. Tian, C. Finn, and S. Levine. A workflow for offline model-free robotic reinforcement learning. *arXiv preprint arXiv:2109.10813*, 2021.

[24] S. Fujimoto, D. Meger, and D. Precup. Off-policy deep reinforcement learning without exploration. *arXiv preprint arXiv:1812.02900*, 2018.

[25] Y. Wu, G. Tucker, and O. Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

[26] A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, pages 11761–11771, 2019.

[27] W. Zhou, S. Bajracharya, and D. Held. Plas: Latent action space for offline reinforcement learning. *arXiv preprint arXiv:2011.07213*, 2020.

[28] S. K. S. Ghasemipour, D. Schuurmans, and S. S. Gu. Emaq: Expected-max q-learning operator for simple yet effective offline and online rl. In *International Conference on Machine Learning*, pages 3682–3691. PMLR, 2021.

[29] S. Fujimoto and S. S. Gu. A minimalist approach to offline reinforcement learning. *arXiv preprint arXiv:2106.06860*, 2021.

[30] I. Kostrikov, R. Fergus, J. Tompson, and O. Nachum. Offline reinforcement learning with fisher divergence critic regularization. In *International Conference on Machine Learning*, pages 5774–5783. PMLR, 2021.

[31] W. Goo and S. Niekum. You only evaluate once: a simple baseline algorithm for offline rl. In *Conference on Robot Learning*, pages 1543–1553. PMLR, 2022.

[32] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

[33] R. Agarwal, D. Schuurmans, and M. Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pages 104–114. PMLR, 2020.

[34] Y. Wu, S. Zhai, N. Srivastava, J. Susskind, J. Zhang, R. Salakhutdinov, and H. Goh. Uncertainty weighted actor-critic for offline reinforcement learning. *arXiv preprint arXiv:2105.08140*, 2021.

[35] T. Yu, G. Thomas, L. Yu, S. Ermon, J. Zou, S. Levine, C. Finn, and T. Ma. Mopo: Model-based offline policy optimization. *arXiv preprint arXiv:2005.13239*, 2020.

[36] R. Kidambi, A. Rajeswaran, P. Netrapalli, and T. Joachims. Morel: Model-based offline reinforcement learning. *arXiv preprint arXiv:2005.05951*, 2020.

[37] P. Swazinna, S. Udluft, and T. Runkler. Overcoming model bias for robust offline deep reinforcement learning. *arXiv preprint arXiv:2008.05533*, 2020.

[38] B.-J. Lee, J. Lee, and K.-E. Kim. Representation balancing offline model-based reinforcement learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=QpNz8r_Ri2Y.

[39] T. Yu, A. Kumar, R. Rafailov, A. Rajeswaran, S. Levine, and C. Finn. Combo: Conservative offline model-based policy optimization. *arXiv preprint arXiv:2102.08363*, 2021.

[40] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[41] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, pages 1094–1100. PMLR, 2020.

[42] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman. Relay policy learning: Solving long horizon tasks via imitation and reinforcement learning. *Conference on Robot Learning (CoRL)*, 2019.

[43] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2020.

[44] X. Geng. Cql. https://github.com/young-geng/cql, 2022.