Uncertainty-Aware Action Repeating Options

Joongkyu Lee
Seoul National University
jklee0717@snu.ac.kr

Seung Joon Park
Samsung Research
soonjun.park@samsung.com

Yunhao Tang
Columbia University
yt2541@columbia.edu

Min-hwan Oh
Seoul National University
minoh@snu.ac.kr

Abstract

In reinforcement learning, employing temporal abstraction within the action space is a prevalent strategy for simplifying policy learning through temporally-extended actions. Recently, algorithms that repeat a primitive action for a certain number of steps, a simple method to implement temporal abstraction in practice, have demonstrated better performance than traditional algorithms. However, a significant drawback of earlier studies on action repetition is the potential for repeated sub-optimal actions to considerably degrade performance. To tackle this problem, we introduce a new algorithm that employs ensemble methods to estimate uncertainty when extending an action. Our framework offers flexibility, allowing policies to either prioritize exploration or adopt an uncertainty-averse stance based on their specific needs. We provide empirical results on various environments, highlighting the superior performance of our proposed method compared to other action-repeating algorithms. These results indicate that our uncertainty-aware strategy effectively counters the downsides of action repetition, enhancing policy learning efficiency.

1 Introduction

Temporal abstraction is a promising approach to solving complex tasks in reinforcement learning (RL) with complex structures and long horizons [15, 14, 27, 36, 30, 4, 7, 20]. Hierarchical reinforcement learning (HRL) enables the decomposition of this sequential decision-making problem into simpler lower-level actions or subtasks. Intuitively, an agent explores the environment more effectively when operating at a higher level of abstraction and solving smaller subtasks [20]. One of the most prominent approaches for HRL is the *option* framework [36, 30], which describes the hierarchical structure in decision making in terms of temporally-extended courses of action. Temporally-extended actions have been shown to speed up learning, potentially providing more effective exploration compared to single-step explorative action and requiring a smaller number of high-level decisions when solving a problem [34, 10]. From a cognitive perspective, such observations are also coherent with how humans learn, generalize from experiences, and perform abstraction over tasks [41].

There has been a line of works that propose repetition of action for an extended period as a specialized form of temporal abstraction [17, 33, 13, 22, 10, 26].[1] Hence, the action-repetition methods address the problem of learning when to perform a new action while repeating an action for multiple time-steps [13, 10]. The extension length, the interaction steps to repeat the same action, is learned

[1] In fact, action repetition for a fixed number of steps was one of the strategies deployed in solving Atari 2600 games [23, 19]. Despite its simplicity, the action repetition provided sufficient performance gains so that almost all modern methods of solving Atari games are still implementing such action repetitions.

NeurIPS 2023 Workshop on Generalization in Planning (GenPlan 2023).

by an agent along with what action to execute [33, 10]. As shown by the improved empirical performances [13, 10], these action repetition approaches can be well justified by the *commitment* to action for deriving a deeper exploration. These approaches can help suppress the dithering behavior of the agent that can result in short-sighted exploration in a local neighborhood.

However, simple action repetition alone cannot guarantee performance improvement. Repetition of a sub-optimal action for an extended period can lead to severe deterioration in the performance. For example, a game may terminate due to reckless action repetition when an agent is in a dangerous region. A more uncertainty-averse behavior would be helpful in this scenario. On the other hand, an agent may linger in the local neighborhood due to a lack of optimism, especially in sparse reward settings. In that case, a more exploration-favor behavior can be beneficial. In either case, a suitable control of uncertainty of value estimates over longer horizons can be a crucial element. In particular, the calibration of how much exploration the agent can take, or how uncertainty-averse the agent should be, can definitely depend on an environment. Thus, the degree of uncertainty to be considered should be adaptive depending on the environment. To this end, we propose to account for uncertainties when repeating actions. To our best knowledge, consideration of uncertainty in the future when instantiating action repetition has been not addressed previously. Such consideration is essential in action repetition in both uncertainty-averse and exploration-favor environments.

In this paper, we propose a novel method that learns to repeat actions while incorporating the estimated uncertainty of the repeated action values. We can either impose aggressive or uncertainty-averse exploration by controlling the degree of uncertainty in order to take suitable uncertainty-aware strategy for the environment. Through extensive experiments and ablation studies, we demonstrate the efficacy of our proposed method and how it enhances the performances of deep reinforcement learning agents in various environments. In comparison with the benchmarks, we show that our proposed method outperforms baselines, consistently outperforming the existing action repetition methods. Our contributions are:

- We present a novel framework that allows the agent to repeat actions in a uncertainty-aware manner using an ensemble method. Suitably controlling the amount of uncertainty induced by repeated actions, our proposed method learns to choose extension length and learns how *optimistic* or *pessimistic* it should be, hence enabling efficient exploration (Section 4).
- Our method offers a notable insight: it's advantageous to account for the inherent uncertainty preference of an environment. Some environments favor uncertainty, while others are uncertainty-averse (see Section 5.1).
- In a series of test environments, we demonstrate that `UTE` consistently surpasses existing action-repetition baselines like `DAR`, ϵz-`Greedy`, `DQN`, and `B-DQN`, in both final evaluation scores and learning speeds (see Section 5.2).

2 Related Work

Temporal Abstraction and Action Repetition. Temporal abstractions can be viewed as an attempt to find a time scale that is adequate for describing the actions of an AI system [30]. The options framework [36, 30, 4] formalizes the idea of temporally-extended actions. An MDP endowed with a set of options are called Semi-Markov Decision Process (SMDP) which we define in Section 3. The generalization of conventional action-value functions for the options framework is called *option*-value functions [36]. The mapping from states to probabilities of taking an option is called policy over options. In the options framework, the agent attempts to learn a policy over options that maximizes the option-value functions.

One simple form of an option is repeating a primitive action for certain number of steps [32]. Action repetition has been widely explored in the literature [17, 33, 13, 22, 10, 26]. Action repetition has been empirically shown to induce deeper exploration [13] and lead to efficient learning by reducing the granularity of control [17, 33, 22, 10]. Action repetition can be implemented by deciding the extension length of an action which is either sampled from a distribution [13] or returned by a policy [17, 33]. The closest related to our work is Biedenkapp et al. [10]. They proposed an algorithm called `TempoRL` that not only selects an action in a state but also for how long to commit to that action. `TempoRL` [10] proposes a hierarchical structure in which *behavior* policy determines the action a to be played given the current state s, and a *skip* policy determines how long to repeat this action. However, our main intuition is that simply repeating the chosen action is not enough. We

may encounter undesirable states while repeating the action. This could lead to catastrophic failure when an agent enters a "risky" area, as we describe in Section 5.1.2. Our method has been shown to effectively manage this issue by quantifying the uncertainty of the option in form of repeating actions.

Uncertainty in Reinforcement Learning. Recently, many works have made significant advances in empirical studies by quantifying and incorporating uncertainty [25, 8, 5, 18]. There are two types of uncertainty: aleatoric and epistemic. Aleatoric uncertainty is the uncertainty caused by the uncontrollable stochastic nature of the environment and cannot be reduced. Epistemic uncertainty is caused by the current imperfect training of the neural network and can be reducible.

One mainstream of estimating the uncertainty in deep RL relies on bootstrapping. Osband et al. [25] introduced Bootstrapped DQN as a method for effcient exploration. This approach is a variation of the classic DQN neural network architecture, which has a shared torso with $K \in \mathbb{Z}^+$ heads. Anschel et al. [1], Peer et al. [28] leveraged an ensemble of Q-functions to mitigate overestimation in DQN. In this paper, we propose an algorithm that quantifies uncertainty of Q-value estimates of the states reached under the repeated-action. This algorithm utilizes multiple randomly-initialized bootstrapped heads that stretch out from a shared network, providing multiple estimates of the *option*-value function. The variance between these estimates is then used as a measure of uncertainty. Notably, this approach allows us to capture both aleatoric and epistemic uncertainty. Then, we establish a UCB-style [3, 2] option-selecting algorithm that simply adds the estimated uncertainty to the averaged ensemble Q-values and chooses an action that maximizes the quantity [11, 28].

3 Preliminaries and Notations

In reinforcement learning, an agent interacts with an environment whose underlying dynamics is modeled by a Markov Decision Process (MDP) [31]. The tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$ defines an MDP $\mathcal{M}$, where $\mathcal{S}$ is a state space, $\mathcal{A}$ is an action space, $P : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is a transition dynamics function, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function, and $\gamma \in [0, 1]$ is the discount factor. We consider a Semi-Markov Decision Process (SMDP) model to incorporate the options framework [36, 30]. An SMDP is an original MDP with a set of options, i.e., $\mathcal{M}_o := \langle \mathcal{S}, \Omega, P_o, R_o \rangle$, where $\omega \in \Omega$ is an option in the option space, $P_o(s' \mid s, \omega) : \mathcal{S} \times \Omega \to \mathcal{S}$ is the probability of transitioning from state s to state s' after taking an option ω and $R_o : \mathcal{S} \times \Omega \to \mathbb{R}$ is the reward function for the option.

For any set $\mathcal{X}$, let $\mathcal{P}(\mathcal{X})$ denote the space of probability distributions over $\mathcal{X}$. Then a policy over option $\pi_\omega : \mathcal{S} \to \mathcal{P}(\Omega)$ assigns a probability to an option conditioned on a given state. Our goal is to learn a policy π_ω that maximizes the expectation of discounted return starting from a initial state s_0; then, define the value functions $V^{\pi_\omega}(s_0) = \mathbb{E}_{\pi_\omega}[\sum_{t=0}^{\infty} \gamma^t R_t \mid s_0]$, the action-value functions $Q^{\pi_\omega}(s_0, a) = \mathbb{E}_{\pi_\omega}[\sum_{t=0}^{\infty} \gamma^t R_t \mid s_0, a]$, or the option-value functions $\tilde{Q}^{\pi_\omega}(s_0, \omega) = \mathbb{E}_{\pi_\omega}[\sum_{t=0}^{\infty} \gamma^t R_t \mid s_0, \omega]$.

In general, options depend on the entire *history* between time step t when they were initiated and the current time step $t + k$, $h_{t:t+k} := s_t a_t s_{t+1} ... a_{t+k-1} s_{t+k}$. Let $\mathcal{H}$ be the space of all possible histories h, then a *semi-Markov option* ω is a tuple $\omega := \langle \mathcal{I}_o, \pi_o, \beta_o \rangle$, where $\mathcal{I}_o \subset \mathcal{S}$ is an initiation set, $\pi_o : \mathcal{H} \to \mathcal{P}(\mathcal{A})$ is an *intra-option* policy, and $\beta_o : \mathcal{H} \to [0, 1]$ is a termination function. In this framework, we define an action repeating option to be $\omega_{aj} := \langle \mathcal{S}, \mathbb{1}_a, \beta(h) = \mathbb{1}_{|h|=j} \rangle$, in which $h \in \mathcal{H}$ and $\mathbb{1}_a$ indicates $|\mathcal{A}|$-dimensional vector where the element corresponding to a is 1 and 0 otherwise. This action repeating option takes action a for j times and then terminates.

When an agent plays a chosen action for extension length j, total of $\frac{j(j+1)}{2}$ skip-transitions are observed and stored in the replay buffer [10]. Specifically, when repeating the action for j times from state s, we can also experience $(s \to s'_{(1)}), (s \to s'_{(2)}), \ldots, (s'_{(1)} \to s'_{(2)}), \ldots, (s'_{(j-1)} \to s'_{(j)})$, in total $\frac{j \cdot (j+1)}{2}$ transitions. We leverage these transitions to update option-values. Consequently, the observations for short extensions are updated more frequently, leading to smaller uncertainties for short extensions and larger uncertainties for long extensions.

4 Uncertainty-aware Temporal Extension

In this section, we propose our algorithm UTE: **U**ncertainty-Aware **T**emporal **E**xtension, which repeats the action in consideration of uncertainty in Q-values. We first demonstrate temporally-extended Q-learning by decomposing the action repeating option. We then describe how we estimate the uncertainty of an option-value function $\tilde{Q}^{\pi_\omega}$ by utilizing the ensemble method to select an extension length j in consideration of uncertainty. We additionally show that *n*-step targets can be used for learning the action-value function Q^{π_ω} without worrying about off-policy correction.

4.1 Temporally-Extended Q-Learning

In this work, we mainly depend on techniques based on the Q-learning algorithm [39], which seeks to approximate the Bellman optimality operator to learn the optimal policy:

Definition 4.1. We define the optimal action-value function $Q^{\pi^*_\omega}$ and the optimal option-value function $\tilde{Q}^{\pi^*_\omega}$ respectively as

$$Q^{\pi^*_\omega}(s,a) = \mathbb{E}_{s'_{(1)} \sim P}\Big[R(s,a) + \gamma \max_{a'} Q^{\pi^*_\omega}(s'_{(1)}, a')\Big], \tag{1}$$

$$\tilde{Q}^{\pi^*_\omega}(s,\omega_{aj}) = \mathbb{E}_{s'_{(j)} \sim P_o}\Big[R_o(s,\omega_{aj}) + \gamma^j \max_{\omega'} \tilde{Q}^{\pi^*_\omega}(s'_{(j)}, \omega')\Big], \tag{2}$$

where $s'_{(0)}$ and $s'_{(j)}$, respectively, indicate one-step and j-step later state from the state s . In practice, it is common to use a function approximator to estimate each Q-value, $Q^{\pi_\omega}(s,a;\theta) \approx Q^{\pi^*_\omega}(s,a)$ and $\tilde{Q}^{\pi_\omega}(s,\omega_{aj};\phi) \approx \tilde{Q}^{\pi^*_\omega}(s,\omega_{aj})$. We use two different neural network function approximators parameterized by θ and ϕ respectively.

Option Decomposition. Learning the optimal policy over options, instead of the optimal action policy, has the same effect as enlarging the action space from $|\mathcal{A}|$ to $|\mathcal{A}| \times |\mathcal{J}|$, where $\mathcal{J}$ ={1, 2, ... , max repetition}. Generally, inaccuracies in Q-function estimations can cause the learning process to converge to a sub-optimal policy, and this phenomenon is amplified in situations with large action spaces [37, 42]. Therefore, we consider decomposed policy over option [10], $\pi_\omega(\omega_{aj} \mid s) := \pi_a(a \mid s) \cdot \pi_e(j \mid s, a)$, in which an action policy $\pi_a(a \mid s) : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ assigns some probability to each action conditioned on a given state, and then an extension policy $\pi_e(j \mid s, a) : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{J})$ assigns some probability to each extension length conditioned on a given state and action. Note that there exists a hierarchy between decomposed policies π_a and π_e, thus, π_a always has to be queried before π_e at every time an option initiates. The agent first chooses an action a from action policy π_a based on the action-value function Q^{π_ω} (e.g. ϵ-greedy). Then, given this action a, it selects extension length j from π_e according to the option-value function $\tilde{Q}^{\pi_\omega}$.

By decomposing the policy over option π_ω, we can decrease the search space from $|\mathcal{A}| \times |\mathcal{J}|$ to $|\mathcal{A}| + |\mathcal{J}|$. However, this learning process may converge to a sub-optimal policy because it is intractable to search all the possible combinations of actions and extension lengths (a, j). The agent may repeat the sub-optimal action excessively or sometimes be overly myopic. Our algorithm can mitigate this issue by controlling the level of uncertainty when executing the extension policy π_e.

Proposition 4.2. *In a Semi-Markov Decision Process (SMDP), let an option $\omega \in \Omega$ be the action repeating option defined by action a and extension length j, i.e. $\omega_{aj} := \langle \mathcal{S}, \mathbb{1}_a, \beta(h) = \mathbb{1}_{h=j} \rangle$. For all $\omega \in \Omega$, a policy over option, π_ω, can be decomposed by an action policy $\pi_a(a \mid s) : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ and an extension policy $\pi_e(j \mid s, a) : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(|J|)$, i.e. $\pi_\omega(\omega_{aj} \mid s) := \pi_a(a \mid s) \cdot \pi_e(j \mid s, a)$. Then, for the corresponding optimal policy π^*_ω, the following holds:*

$$V^{\pi^*_\omega}(s) = \max_{\omega_{aj}} Q^{\pi^*_\omega}(s, \omega_{aj}) = \max_{a} Q^{\pi^*_\omega}(s,a).$$

Proof of Propositon 4.2. For any $s \in \mathcal{S}$, define the value of executing an action in the context of a state-option pair as $Q_U : \mathcal{S} \times \Omega \times \mathcal{A} \to \mathbb{R}$. Let $\pi_U(a \mid s, \omega_{aj}) : \mathcal{S} \times \Omega \to \mathcal{P}(\mathcal{A})$ be an *intra-option* policy [36], which returns an action a when executing an option ω_{aj} at state s. Then, option-value functions can be written as:

$$\tilde{Q}^{\pi_\omega}(s, \omega_{aj}) = \sum_{a'} \pi_U(a'|s, \omega_{aj}) Q_U(s, \omega_{aj}, a') = \sum_{a'} \mathbb{1}_a Q_U(s, \omega_{aj}, a') = Q_U(s, \omega_{aj}, a), \tag{3}$$

Algorithm 1 UTE: Uncertainty-aware Temporal Extension

Input: uncertainty parameter λ, the number of output heads of option-value functions B.
Initialize: Q^{π_ω}, $\{\tilde{Q}^{\pi_\omega}_{(b)}\}_{b=1}^{B}$.
for episode = $1, \ldots, K$ **do**
    Obtain initial state s from environment
    repeat
        $a \leftarrow \epsilon$-greedy $\mathrm{argmax}_{a'}\ Q^{\pi_\omega}(s, a)$
        Calculate $\hat{\mu}_{\pi_\omega}(s, \omega_{aj})$, $\hat{\sigma}^2_{\pi_\omega}(s, \omega_{aj})$ by Eq. (4).
        $j \leftarrow \mathrm{argmax}_{j'} \{\hat{\mu}_{\pi_\omega}(s, \omega_{aj'}) + \lambda \hat{\sigma}_{\pi_\omega}(s, \omega_{aj'})\}$
        while $j \neq 0$ and s is not terminal **do**
            Take action a and observe s', r
            $s \leftarrow s'$, $j \leftarrow j - 1$
        end while
    until episode ends
end for

where the second equality holds since π_U deterministically returns action a. Therefore we have,

$$V^{\pi^*_\omega}(s_0) = \max_{\omega_{aj}} \tilde{Q}^{\pi^*_\omega}(s_0, \omega_{aj}) = \max_{a,j} \tilde{Q}^{\pi^*_\omega}(s_0, \omega_{aj}) = \max_{a,j} Q^*_U(s, \omega_{aj}, a)$$
$$= \max_a \left\{ \max_j Q^*_U(s, \omega_{aj}, a) \right\} = \max_a Q^{\pi^*_\omega}(s, a),$$

where the second equality holds since ω_{aj} is determined by an action a and extension length j, the third equality is by Eq.(3), and the last equality holds since π^*_ω is the optimal policy. This concludes the proof. □

Proposition 4.2 implies that target value for the option selection of repeated actions can be the same as the target for a single-step action selection within the option. In our implementation, we use $\max_{a'} Q^{\pi^*_\omega}(s'_{(j)}, a')$ instead of $\max_{\omega'} \tilde{Q}^{\pi^*_\omega}(s'_{(j)}, \omega'_{aj})$ for the target value in Eq.(2). This can stabilize the learning process by sharing the same target.

4.2 Ensemble-based Uncertainty Quantification

In the previous action repetition methods [17, 33, 13, 10], they extend the chosen action without considering uncertainty which could easily run to failure. The only situation where these problems do not occur is when their extension policies are optimal, which means they need to expect the j step later state precisely. However, it is improbable in the sense that this situation rarely occurs in the learning process. In order to solve this problem, we propose a strategy of choosing a extension length j in an uncertainty-aware manner. `UTE` is a uncertainty-aware version of the `TempoRL` [10]. Our main intuition is that it is crucial to consider the uncertainty of option-value functions $\tilde{Q}^{\pi_\omega}$, when selecting extension length j by extension policy π_e.

We use the ensemble method, which has recently become prevalent in RL [25, 12, 6], to estimate uncertainty in our estimated option-value functions. We use a network consisting of a shared architecture with B independent. "head" branching off from the shared network. Each head corresponds to a option-value function, $\tilde{Q}^{\pi_\omega}_{(b)}$, for $b \in \{1, 2, \ldots, B\}$. Each head is randomly-initialized and trained by different samples from an experience buffer. Unlike Bootstrapped DQN (`B-DQN`) [25] where each one of the value function heads is trained against its own target network, our `UTE` trains each value function head against the same target. If each head has its own target head respectively, since the objective function of neural networks is generally non-convex, each Q-value may converge to different modes. In this case, as training the policy, the estimated uncertainty of option Q-value, $\hat{\sigma}_{\pi_\omega}$, could not converge to zero. This means that it is unable to learn an optimal policy. Therefore, using the same target is one of the key points of our implementation.

Given state s and action a, $\tilde{Q}^{\pi_\omega}_{(b)}$-values are aggregated by extension length j to estimate mean and variance as follows:

$$\hat{\mu}_{\pi_\omega}(s,\omega_{aj}) := \frac{1}{B}\sum_{b=1}^{B}\tilde{Q}^{\pi_\omega}_{(b)}(s,\omega_{aj}), \quad \hat{\sigma}^2_{\pi_\omega}(s,\omega_{aj}) := \frac{1}{B}\sum_{b=1}^{B}(\tilde{Q}^{\pi_\omega}_{(b)}(s,\omega_{aj}))^2 - (\hat{\mu}_{\pi_\omega}(s,\omega_{aj}))^2. \tag{4}$$

Then, we define *uncertainty-aware* extension policy π_e, which takes extension length j deterministically given state and action, by introducing the uncertainty parameter $\lambda \in \mathbb{R}$:

$$j = \underset{j' \in \mathcal{J}}{\operatorname{argmax}} \{\hat{\mu}_{\pi_\omega}(s,\omega_{aj'}) + \lambda\hat{\sigma}_{\pi_\omega}(s,\omega_{aj'})\}.$$

where λ indicates the level of uncertainty to be considered. The positive λ induces more aggressive exploration, and the negative one causes uncertainty-averse exploration.

4.3 Adaptive Uncertainty Parameter λ.

Instead of fixing λ during the learning process, we propose the adaptive selection of λ utilizing a non-stationary multi-arm bandit algorithm, as described in [5]. Consider Λ as the predefined set of uncertainty parameters. At the onset of each episode k, the bandit selects an arm, denoted by $\lambda_k \in \Lambda$, and subsequently receives feedback in the form of episode returns $R_k(\lambda_k)$. Given that the reward signal $R_k(\lambda_k)$ is non-stationary, we employ a sliding-window UCB combined with ϵ_{ucb}-greedy exploration to optimize the process.

Let $\tau \in \mathbb{Z}^+$ be the size of window such that $\tau < K$. The number of time episodes an arm $\lambda \in \Lambda$ has been played in episode k for a window size τ as:

$$N^\tau_k(\lambda) = \sum_{k'=\max(0,k-\tau)}^{k-1} \mathbb{1}(A_{k'} = \lambda),$$

where A_k indicates the chosen action at episode k and $\mathbb{1}(A_k = \lambda)$ is an indicator function. Define the empirical mean reward of an arm λ for a window size τ as:

$$\widehat{\mu}^\tau_k(\lambda) = \frac{1}{N^\tau_k(\lambda)} \sum_{k'=\max(0,k-\tau)}^{k=1} R_{k'}(\lambda)\mathbb{1}(A_{k'} = \lambda).$$

Since we use the sliding window UCB with ϵ_{ucb}-greedy exploration, our bandit algorithm is as follows:

$$\begin{cases} \forall 0 \le k \le N-1, & A_k = k, \\ \forall N \le k \le K-1 \text{ and } U_k \ge \epsilon_{ucb}, & A_k = \operatorname{argmax}_{\lambda\in\Lambda}\widehat{\mu}^\tau_{k-1}(\lambda) + \beta\sqrt{\frac{\log(k-1)}{N^\tau_{k-1}(\lambda)}}, \\ \forall N \le k \le K-1 \text{ and } U_k < \epsilon_{ucb}, & A_k = Y_k, \end{cases}$$

where U_k is a random variable drawn uniformly from $[0,1]$ and Y_k is a random action sampled uniformly from Λ.

4.4 n-step Q-Learning

We make use of n-step Q-learning [35] to learn both Q^{π_ω} and $\tilde{Q}^{\pi_\omega}$, whereas `TempoRL` [10] used it only for updating $\tilde{Q}^{\pi_\omega}$. We found that *n*-step targets can also be used to update Q^{π_ω}-values without any off-policy correction [16], e.g., importance sampling. Given the sampled n-step transition $\tau_t = (s_t, a_t, R_o(s_t, o_{an}), s_{t+n})$ from replay buffer $\mathcal{R}$, as long as n is smaller than or equal to the current extension policy π_e's output j, the transition τ_t trivially follows our target policy π_ω. Thus, τ_t can be directly used to update the action-value function Q^{π_ω}. Instead of one step Q-learning in Eq.(1), `UTE` uses n-step Q-Learning to update Q^{π_ω}:

$$\mathcal{L}_{Q^{\pi_\omega}}(\theta) = \mathbb{E}_{\tau_t\sim\mathcal{R}}\Big[(Q^{\pi_\omega}(s_t,a_t;\theta) - \sum_{k=0}^{n-1}\gamma^k r_{t+k} - \gamma^n \max_{a'} Q^{\pi^*_\omega}(s_{t+n},a';\bar{\theta}))^2 \,\Big|\, n \le j \sim \pi_e\Big],$$

where $\bar{\theta}$ are the delayed parameters of action-value function Q^{π_ω} and $j \sim \pi_e(j_t \mid s_t, a_t)$. In general, n-step returns can be used to propagate rewards faster [40, 29]. It mitigates the overestimation problem in Q-learning as well [21]. Note that we don't need to pre-define n because it is dynamically determined by current extension policy π_e.

5 Experiments

In this section, we validate our hypothesis that a positive λ promotes aggressive exploration, while a negative λ leads to uncertainty-averse exploration (Section 5.1). We then highlight the profound influence of a well-tuned λ on performance in more intricate environments, showcasing that the adaptive choice of λ consistently surpasses other baseline measures (Section 5.2).

5.1 Tabular RL

5.1.1 Chian MDP

We conducted experiments in the Chain MDP environment, as depicted in Figure 1 [25]. In this setting, there are two possible actions: {left, right}. When the agent reaches the left end (s_1) of the chain and chooses the left action, it receives a small deceptive reward of 0.001. Conversely, if the agent reaches the right end (s_n) and selects the right action, it is rewarded with a larger amount of 1.0. This makes the optimal policy straightforward: always take the right action. The agent's interactions with the environment have a fixed horizon length of $N + 8$, where N represents the chain length. As a result, the agent can achieve rewards ranging from zero to 10 in each episode. Given the sparse nature of the rewards, a "deep" exploration strategy is necessary to discern the optimal policy. In this toy environment, our goal is to validate the hypothesis that a positive uncertainty parameter (λ) promotes deep exploration. We anticipate that this will lead to `UTE` outperforming other baselines such as `DDQN`[38], ϵz-`Greedy`[13], and `TempoRL` [10]. For both `TempoRL` and `UTE`, we set a maximum extension length of 10.

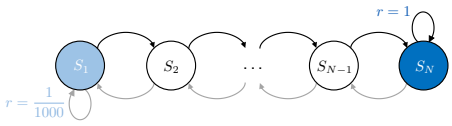


Figure 1: Chain MDP

Chain Length	`TempoRL`	ϵz-`Greedy`	`UTE` (λ) -2.0	-1.0	0.0	1.0	2.0
10	0.90	0.65	0.88	0.91	0.90	0.92	**0.92**
30	0.74	0.43	0.45	0.55	0.62	0.73	**0.76**
50	0.25	0.43	0.07	0.05	0.37	0.47	**0.67**
70	0.06	0.13	0.01	0.01	0.01	0.07	**0.19**

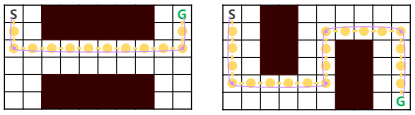
Table 1: Normalized AUC for reward over 20 random seeds in Chain MDP.

Exploration-Favor. Table 1 summarizes the results for various chain lengths in terms of the normalized area under the reward curve (AUC). A reward AUC value closer to 1.0 indicates that the agent found the optimal policy more quickly. The total number of training episodes used to calculate the AUC was set to $1,000$ for chain lengths of 10, 30, and 50, and $5,000$ for 70. The table's results demonstrate that `UTE` consistently outperforms the other two baselines across various chain lengths, even in more challenging scenarios with longer chain lengths, especially when uncertainty parameter is set to $\lambda = +2.0$. When the agent selects a random action using the ϵ-greedy action policy, it can explore more deeply by adopting a more optimistic stance, leading to quicker convergence to the optimal solution. An aggressive exploration strategy is advantageous in this environment, as there are no risky areas where the game terminates due to repeated actions.

5.1.2 Gridworlds

In this subsection, we examine the empirical behavior of various algorithms within the Gridworlds environment, specifically the *Lava* scenario (refer to Figure 2). This environment is represented as a 6×10 grid, comprising discrete states and actions. An agent commences its journey from the top-left corner and strives to reach its goal to obtain a positive reward of $+1$. However, the agent must avoid the lava, as stepping into it results in a reward of -1. Unlike the chain MDP environment, the presence of the perilous "lava" region implies that an uncertainty-averse strategy is more advantageous. For our analysis, we compare our approach with several other methods, including vanilla `DDQN` [38], ϵz-`Greedy`[13], and `TempoRL`[10]. All agents were trained over a span of 3.0×10^3 episodes. We employed three distinct types of ϵ-greedy exploration schedules: one that linearly decays from 1.0 to 0.0 throughout the episodes, a logarithmically decaying schedule, and a fixed schedule with $\epsilon = 0.1$.

Additionally, we capped the maximum extension length at 7. It's important to mention that instead of utilizing tabular Q-learning, we opted for neural networks to deduce the Q-value functions.



(a) Bridge (b) ZigZag

Figure 2: 6×10 Gridworlds. Agents have to reach a goal state (G) from a starting state (S) detouring the lava. Dots represent decision steps with and without temporally-extended actions.

Env	ϵ-decay	DDQN	TempoRL	UTE (λ) -0.5	-1.0	-1.5
Bridge	Linear	0.61	0.44	0.83	0.84	**0.86**
	Log	0.54	0.32	0.85	0.88	**0.92**
	Fixed	0.57	0.41	0.72	0.82	**0.83**
Zigzag	Linear	0.38	0.14	0.73	0.82	**0.84**
	Log	0.46	0.12	0.66	0.86	**0.89**
	Fixed	0.34	0.19	0.62	0.70	**0.76**

Table 2: Normalized AUC for reward across different ϵ exploration schedules over 20 random seeds.

Uncertainty-Averse. Table 2 shows that, across all ϵ exploration strategies, UTE consistently outperforms other methods. Notably, its performance improves as the uncertainty parameter λ becomes more negative (i.e., a larger negative value for λ). This finding aligns with our assertion that a pessimistic strategy is more advantageous in environments fraught with unsafe regions. Additionally, even when the exploration rate for π_a is set relatively high (e.g., fixed at $\epsilon = 0.1$), UTE consistently exhibits superior performance compared to other methods. These results underscore UTE's robustness against various ϵ-greedy schedules.

Intriguingly, the performance of TempoRL falls significantly short of that described in the original paper [10]. This deviation can be attributed to our use of function approximation for estimating Q-values, in contrast to the traditional tabular Q-learning method. It's well-documented that function approximation can sometimes introduce uncontrolled or undesirable overestimation biases [24]. Consequently, opting for the extension length with the highest value can lead to unfavorable outcomes, especially in settings relying on function approximation.

5.2 Deep RL: Atari 2600

In this subsection, we assess UTE's performance on the Atari benchmark. We compare it against six baseline algorithms: i) vanilla DDQN [38], ii) Fixed Repeat with $j = 4$, iii) ϵz-Greedy[13], iv) DAR (Dynamic Action Repetition[17]), v) TempoRL[10], and vi) B-DQN (Bootstrapped DQN)[25].

Fixed Repeat is an algorithm that consistently repeats a given action a set number of times. Due to resource constraints, each algorithm underwent training for a total of 2.5×10^6 steps, equivalent to 10 million frames. All algorithms, with the exception of B-DQN, employed a linearly decaying ϵ-greedy exploration schedule for the initial $200,000$ time-steps, subsequently settling at a fixed ϵ of 0.01. We evaluated all agents after every $10,000$ training steps, evaluating over 3 episodes with a very small ϵ exploration rate of 0.001. The experiments were conducted using *OpenAi Gym's* Atari environment, incorporating a 4 frame-skip [9]. We set the maximum extension length value to 10. To guarantee a fair comparison, we extensively explored a broad range of hyperparameters to determine the optimal value for each algorithm. For a detailed breakdown, please refer to Tables C.1 and C.2 in the Appendix.

Uncertainty-Awareness. Figure 3 presents the learning curves for UTE alongside other baseline algorithms (a comprehensive version can be found in Figure C.1). Table 3 summarizes the game results in terms of average rewards accrued over the last $100,000$ time steps (for results on other environments, see Table C.3). Overall, especially when using the best λ, UTE consistently achieves higher final rewards than other agents. These findings underscore that when λ is adeptly calibrated to the environment, our method significantly outstrips the performance of existing action repetition methods like DAR, ϵz-Greedy, and TempoRL, as well as the deep exploration algorithm B-DQN. Notably, we observed that the Fixed Repeat algorithm struggles to learn effectively in most games, underscoring the importance of mastering a extension policy to optimize performance.

Moreover, the results in Table 3 highlight that implementing n-step learning dramatically bolsters performance. This lends empirical weight to our contention that off-policy correction is superfluous within our action-repeating options framework, as discussed in Section 4.4.

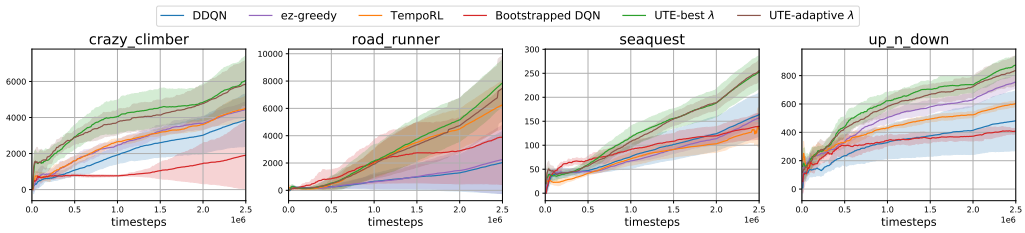


Figure 3: Learning curves of UTE with best λ, UTE with adaptive λ and other baseline algorithms on Atari environments. The shaded area represents the standard deviation over 7 random seeds.

Environment	DDQN	Fixed-j	ϵz-Greedy	DAR	TempoRL	B-DQN	UTE 1-step	UTE n-step	UTE Adaptive λ
Crazy Climber	5265.8 ± 4063.4	3731.1 ± 2997.2	5295.1 ±3609.7	2059.1 ±1225.1	4885.5 ±3378.3	2961.6 ±3080.0	6761.9 ±5061.9	**8175.6** ±5790.4	7046.3 ±5350.0
Road Runner	3277.0 ±4470.3	1230.3 ±1640.9	3733.8 ±4716.5	845.5 ±791.9	8131.5 ±4099.3	4976.8 ±6032.6	4935.6 ±5206.4	**12323.2** ±4177.1	10353.3 ±3283.3
Sea Quest	207.6 ±124.5	47.0 ±26.2	214.5 ±85.6	42.8 ±33.9	128.2 ±55.5	145.1 ±64.5	206.9 ±92.4	313.4 ±141.1	**320.3** ±159.4
Up n Down	536.4 ±361.5	594.8 ±324.6	823.1 ±320.0	348.7 ±227.0	641.5 ±428.6	383.2 ±242.8	911.5 ±476.7	**1072.8** ±664.0	990.4 ±707.5

Table 3: Average rewards and standard deviations (small numbers) over the last $100,000$ time steps over Atari environments.

Adaptive Uncertainty Parameter λ. As described in Section 4.3, each arm corresponds to extension length λ. At the beginning of each episode, the bandit algorithm chooses λ_k among the set, $\lambda_k \in \Lambda := \{+1.0, +0.5, +0.2, 0.0, -0.2, -0.5, -1.0, -1.5\}$, and gets the feedback of episode rewards $R_k(\lambda_k)$. Then, the bandit algorithm update $N_k^\tau(\lambda_k)$.

As shown in Figure 3 and Table 3, the learning speed of UTE when using an adaptively chosen λ is marginally slower than that of the standard UTE. This minor slowdown can be attributed to the need for extra samples to fine-tune λ. However, despite this adjustment, UTE with an adaptive λ consistently surpasses other baseline methods by a significant margin. These findings are particularly promising because they eliminate the necessity to predetermine the value of λ, thereby alleviating the challenges of hyperparameter tuning. Such a feature accentuates the practicality and efficacy of our approach in intricate learning environments.

6 Conclusion

We propose a new method that learns to repeat actions while explicitly considering the uncertainty over the Q-value estimates of the states reached under the repeated-action option. In various environments, we empirically showed that it is important to consider environment-inherent uncertainty, and a well-suited uncertainty parameter λ significantly outperforms other existing action repetition algorithms. The improved performance comes from its ability to repeat a sub-optimal action less in risky environments and explore deeper in exploration-favor environments. To our best knowledge, this is the first deep RL algorithm considering uncertainty in the future when instantiating temporally extended

References

[1] Oron Anschel, Nir Baram, and Nahum Shimkin. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. In *International conference on machine learning*, pages 176–185. PMLR, 2017.

[2] Jean-Yves Audibert, Rémi Munos, and Csaba Szepesvári. Exploration–exploitation tradeoff using variance estimates in multi-armed bandits. *Theoretical Computer Science*, 410(19): 1876–1902, 2009.

[3] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47:235–256, 2002.

[4] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[5] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. Agent57: Outperforming the atari human benchmark. In *International Conference on Machine Learning*, pages 507–517. PMLR, 2020.

[6] Chenjia Bai, Lingxiao Wang, Lei Han, Jianye Hao, Animesh Garg, Peng Liu, and Zhaoran Wang. Principled exploration via optimistic bootstrapping and backward induction. In *International Conference on Machine Learning (ICML 2021)*, pages 577–587. PMLR, 2021.

[7] André Barreto, Diana Borsa, Shaobo Hou, Gheorghe Comanici, Eser Aygün, Philippe Hamel, Daniel Toyama, Shibl Mourad, David Silver, Doina Precup, et al. The option keyboard: Combining skills in reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[8] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

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

[10] André Biedenkapp, Raghu Rajan, Frank Hutter, and Marius Lindauer. TempoRL: Learning when to act. In *Proceedings of the 38th International Conference on Machine Learning (ICML 2021)*, July 2021.

[11] Richard Y Chen, John Schulman, Pieter Abbeel, and Szymon Sidor. Ucb and infogain exploration via q-ensembles. *arXiv preprint arXiv:1706.01502*, 9, 2017.

[12] Felipe Leno Da Silva, Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. Uncertainty-aware action advising for deep reinforcement learning agents. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 5792–5799, 2020.

[13] Will Dabney, Georg Ostrovski, and Andre Barreto. Temporally-extended ε-greedy exploration. In *9th International Conference on Learning Representations, ICLR 2021*, 2020. URL `https://openreview.net/forum?id=ONBPHFZ7zG4`.

[14] Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. *Advances in neural information processing systems*, 5, 1992.

[15] Richard E Fikes, Peter E Hart, and Nils J Nilsson. Learning and executing generalized robot plans. *Artificial intelligence*, 3:251–288, 1972.

[16] Anna Harutyunyan, Marc G Bellemare, Tom Stepleton, and Rémi Munos. Q (λ) with off-policy corrections. In *International Conference on Algorithmic Learning Theory*, pages 305–320. Springer, 2016.

[17] Aravind Lakshminarayanan, Sahil Sharma, and Balaraman Ravindran. Dynamic action repetition for deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[18] Seunghyun Lee, Younggyo Seo, Kimin Lee, Pieter Abbeel, and Jinwoo Shin. Offline-to-online reinforcement learning via balanced replay and pessimistic q-ensemble. In *Conference on Robot Learning*, pages 1702–1712. PMLR, 2022.

[19] Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew J. Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

[20] Marlos C Machado, Andre Barreto, and Doina Precup. Temporal abstraction in reinforcement learning with the successor representation. *arXiv preprint arXiv:2110.05740*, 2021.

[21] Lingheng Meng, Rob Gorbet, and Dana Kulić. The effect of multi-step methods on overestimation in deep reinforcement learning. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 347–353. IEEE, 2021.

[22] Alberto Maria Metelli, Flavio Mazzolini, Lorenzo Bisi, Luca Sabbioni, and Marcello Restelli. Control frequency adaptation via action persistence in batch reinforcement learning. In *International Conference on Machine Learning (ICML 2020)*, pages 6862–6873. PMLR, 2020.

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[24] Ted Moskovitz, Jack Parker-Holder, Aldo Pacchiano, Michael Arbel, and Michael Jordan. Tactical optimism and pessimism for deep reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[25] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped DQN. In *Advances In Neural Information Processing Systems 29*, pages 4026–4034, 2016.

[26] Seohong Park, Jaekyeom Kim, and Gunhee Kim. Time discretization-invariant safe action repetition for policy gradient methods. *Advances in Neural Information Processing Systems*, 34, 2021.

[27] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. *Advances in neural information processing systems*, 10, 1997.

[28] Oren Peer, Chen Tessler, Nadav Merlis, and Ron Meir. Ensemble bootstrapping for q-learning. In *International Conference on Machine Learning*, pages 8454–8463. PMLR, 2021.

[29] Jing Peng and Ronald J Williams. Incremental multi-step q-learning. In *Machine Learning Proceedings 1994*, pages 226–232. Elsevier, 1994.

[30] Doina Precup. *Temporal abstraction in reinforcement learning*. University of Massachusetts Amherst, 2000.

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

[32] Ralf Schoknecht and Martin Riedmiller. Speeding-up reinforcement learning with multi-step actions. In *International Conference on Artificial Neural Networks*, pages 813–818. Springer, 2002.

[33] Sahil Sharma, Aravind Srinivas, and Balaraman Ravindran. Learning to repeat: Fine grained action repetition for deep reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017*, 2017. URL `https://openreview.net/forum?id=B1GOWV5eg`.

[34] Martin Stolle and Doina Precup. Learning options in reinforcement learning. In *International Symposium on abstraction, reformulation, and approximation*, pages 212–223. Springer, 2002.

[35] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[36] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211, 1999.

[37] Sebastian Thrun and Anton Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*, volume 6, 1993.

[38] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[39] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[40] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[41] Liyu Xia and Anne GE Collins. Temporal and state abstractions for efficient learning, transfer, and composition in humans. *Psychological review*, 2021.

[42] Tom Zahavy, Matan Haroush, Nadav Merlis, Daniel J Mankowitz, and Shie Mannor. Learn what not to learn: Action elimination with deep reinforcement learning. *Advances in Neural Information Processing Systems*, 31, 2018.

A Details of Baselines

Fixed Repeat. Fixed Repeat in Atari experiment corresponds to a `DDQN` agent that always repeats the action for a fixed amount of times. In other words, the extension policy returns the same j at every decision time. In our settings, j is set to 4. The reason for evaluating this naive method is to confirm that this approach fails so that learning extension length is crucial.

Temporally-Extended ϵ-Greedy. ϵz-`Greedy` [13] is a simple add-on to the ϵ-greedy policy. The agent follows the current policy for one step with probability $1-\epsilon$, or with probability ϵ samples an action a from a uniform random distribution and repeats it for j times, which is drawn from a pre-defined duration distribution. We used the heavy-tailed zeta distribution, with $\mu = 1.25$ as the duration distribution in the Chain MDP and the Gridworlds environment. This was done by conducting a hyperparameter search on μ for the set {1.25, 1.5, 2.0, 2.5, 3.0}. In Atari games, we choose the best per-game μ among the set {1.5, 1.75, 2.0, 2.25, 2.5} for a fair comparison to our `UTE`. A combination of ϵ chance to explore and zeta-distributed duration is called ϵz-greedy exploration.

The experimental results from [13] show that ϵz-`Greedy` incorporated in existing R2D2 and Rainbow agents result higher median human-normalized score over the 57 Atari games. However, this algorithm is highly dependent on the exploration rate ϵ, which can cause difficulties in online learning.

Dynamic Action Repetition. `DAR` [17] is a framework for discrete-action space deep RL algorithms. `DAR` duplicates the output heads twice such that an agent can choose from $2 \times |A|$ actions. And each output heads corresponds to pre-defined repetition values, r_1, r_2, where r_1 and r_2 are fixed hyper parameters. Hence, action a_k is repeated r_1 number of times if $k < |\mathcal{A}|$ and r_2 number of times if $k \geq |\mathcal{A}|$. In our experiments, r_1 is fixed to maximum extension length J and r_2 to 1 to allow for actions at every time step.

There are some drawbacks to this approach. First, r_1 and r_2 have to be predefined, which means we need prior knowledge of the environments. And also, the learning process becomes a lot more difficult because the action space doubled.

TempoRL. `TempoRL` [10] proposes a "flat" hierarchical structure in which *behavior* policy (π_a) determines the action a to be played given the current state s, and a *skip* policy (π_j) determines how long to repeat this action. The flat hierarchical structure refers to *behavior* policy and *skip* policy having to make decisions at the same time-step. The action policy has to be always queried before the skip policy. When an agent plays a chosen action for extension length j, total of $\frac{j(j+1)}{2}$ skip-transitions are observed and stored in the replay buffer. The *behavior* and the *skip* Q-functions can be updated using one-step observations and the overarching skip-observation. Using the samples collected, the *behavior* policy can be learned by a classical one step Q-learning. The n-step Q-learning is used to learn the *skip* value with the condition that, at each step in the j steps, the action stays the same.

Bootstrapped DQN. `B-DQN` [25] is an algorithm for temporally-extended (or deep) exploration. Inspired by Thompson sampling, it selects an action without the need for an intractable exact posterior update. Osband et al. [25] suggest bootstrapped neural nets can produce reasonable posterior estimates. The network of bootstrapped DQN consists of a shared architecture with K bootstrapped "heads" stretching off independently. Each head is initialized randomly and trained only on its bootstrapped sub-sample of the data. The shared network learns a joint feature representation across all the data. For evaluation, an ensemble voting policy is used to decide action.

B Atari Experiments Details

All experiments were run on an internal cluster containing GeForce RTX 3090 GPUs. Atari experiments took 13 hours to train for 10 million frames on GPU. `TempoRL` baseline implementation is from Biedenkapp et al. [10]. This asset is licensed under Apache License 2.0. The Arcade Learning Environment (ALE) [9] for Atari games is licensed under the GNU General Public License Version 2. Baseline codes can be found at `https://github.com/automl/TempoRL`.

Network Architecture. The input size of images is 84×84, and the last 4 frames of this image are stacked together. This will be our input throughout the experiment.

DDQN agent uses the same architecture for DQN of [23] with the target network [38]. This architecture has 3 convolutional layers of 32, 64 and 64 feature planes with kernel sizes of 8,4 and 3, and strides of 4,2, and 1, respectively. These are followed by a fully connected network with 512 hidden units followed by another fully connected layer to the Q-Values for each action.
ϵz-**Greedy** agent uses the exact same architecture as Mnih et al. [23]. The only difference with DDQN is that ϵz-Greedy repeats an exploratory action, which is sampled from uniform random distribution. And the extension length j is sampled from zeta distribution. The hyper-parameter μ for zeta distribution is set depending on the experiments: 1.25 for Chain-MDP and Gridwolrds, and the best one for each game in Atari experiment.
DAR agent selects action and extension length based on $2 \times |A|$ Q-values. Therefore, the output of the last layer is duplicated and the duplicate outputs corresponding to a different extension length, r_1 and r_2. The hyper-parameters, r_1 and r_2, are set to 1 and 10 respectively.
TempoRL agent uses the shared architecture, the structure of which is the same as one described in Biedenkapp et al. [10]. On top of DQN architecture [23], an additional output stream for the extension length is incorporated. The extension length is embedded into a 10-dimensional vector and then concatenated with the output of the last convolutional layer of the network. The features then pass through two fully connected hidden layers, each with 512 units.
B-DQN has one torso network of 3 convolutional layers, which is the same as that of DQN [23]. However, it has 10 heads branching off independently [25]. Each head consists of two fully connected hidden layers, each with 512 units. Therefore the agent returns 10 Q-values from each head.
UTE uses the similar architecture as that of TempoRL [10]. The main difference compared to TempoRL is that UTE uses an ensemble method for the output stream of extension length. After concatenating a 10-dimensional extension length vector and the output of the last convolutional layer, the concatenated vector pass through 10 heads branching off independently. Each 10 head consists of fully connected layer with 512 hidden units followed by a fully connected layer to the Q-Values for each extension length.

Hyper-parameter	Value
Discount rate	0.99
Gradient Clip	40.0
Target update frequency	500
Learning starts	10 000
Initial ϵ	1.0
Final ϵ	0.01
Evaluation ϵ	0.001
ϵ time-steps	200 000
Train frequency	4
Loss Function	Huber Loss
Optimizer	Adam
Learning rate	0.0001
Batch Size	32
Extension Batch Size	32
Replay buffer size	5×10^4
Extension replay buffer size	5×10^4
Number of ensemble heads	10
Max extension length (J)	10
Uncertainty parameter (λ)	-1.5, -1.0, -0.5, -0.2, 0.0, 0.2, 0.5, 1.0

Table B.1: Hyper-parameters used for the Atari experiments

C Additional Experimental Results on Atari 2600

Per-game Best Parameter. We applied various kinds of uncertainty parameters to our proposed model from +1.0 to -1.5. As shown in Table C.2, the optimal uncertainty parameter varies from environment to environment. In most games, such as *Beam Rider, Centipede, Crazy Climber, Freeway, Qbert, Road Runner, Up n Down*, the uncertainty-averse strategy (negative λ) exhibits an improvement

in averaged rewards over the last 100,000 time steps. Meanwhile, on *Kangaroo*, the exploration-favor strategy (positive λ) shows better performance.
Table C.1 describes that optimal hyperparameter μ for ϵz-Greedy also varies from environment to environment. For fair comparison with our algorithm, we used the per-game best μ for ϵz-Greedy.

	ϵz-Greedy (μ)				
Environment	1.5	1.75	2.0	2.25	2.5
Beam Rider	331.6	272.9	**409.1**	261.4	328.9
Centipede	1271.8	1222.6	1080.2	1316.0	**1431.7**
Crazy Climber	5026.1	4420.0	3128.0	**5295.1**	4690.9
Freeway	**30.8**	30.7	25.6	20.5	25.6
Kangaroo	**609.1**	518.8	604.0	360.0	396.4
Ms Pacman	580.0	551.3	584.5	514.9	**597.2**
Pong	19.7	18.4	19.8	19.4	**19.9**
Qbert	**392.8**	388.6	345.2	264.7	270.6
Riverraid	810.4	835.5	**945.6**	695.5	738.2
Road Runner	2943.0	2215.2	**3733.8**	3131.2	848.5
Sea Quest	116.1	**214.5**	172.6	123.8	119.6
Up n Down	700.6	**823.1**	669.0	794.8	653.2

Table C.1: Average rewards for ϵz-Greedy varying values for hyperparameter μ in the Atari 2600 environments. (7 random seeds)

	UTE (uncertainty parameter: λ)							
Environment	+1.0	+0.5	+0.2	+0.0	-0.2	-0.5	-1.0	-1.5
Beam Rider	384.3 (137.8)	431.6 (162.9)	401.1 (143.3)	425.2 (153.5)	403.1 (127.9)	417.1 (115.4)	**439.5** (163.0)	414.2 (112.3)
Centipede	1898.2 (889.2)	1327.8 (760.6)	1581.4 (1008.2)	2125.6 (1356.4)	**2190.1** (1073.0)	1893.6 (954.9)	1605.9 (1167.7)	1377.5 (875.2)
Crazy Climber	5093.2 (4011.6)	4426.2 (3987.4)	6163.6 (5025.2)	5033.9 (2653.9)	6484.8 (4797.2)	**8175.6** (5790.4)	5198.6 (4049.5)	5220.2 (4751.1)
Freeway	28.6 (3.7)	29.9 (2.0)	27.9 (5.4)	30.5 (1.5)	**30.7** (1.9)	24.1 (3.9)	25.1 (4.7)	25.0 (6.8)
Kangaroo	555.2 (650.7)	493.3 (515.7)	543.0 (450.3)	661.0 (630.4)	623.3 (630.6)	577.6 (622.6)	718.2 (859.1)	**728.5** (832.6)
Ms Pacman	446.5 (229.7)	509.6 (256.3)	551.0 (237.5)	545.5 (218.3)	**551.6** (225.5)	544.1 (249.1)	511.2 (182.2)	540.4 (228.3)
Pong	17.5 (5.9)	18.0 (5.1)	17.1 (5.9)	16.9 (4.5)	**19.1** (2.5)	16.8 (6.2)	18.4 (4.7)	16.4 (4.6)
Qbert	387.4 (415.8)	400.5 (490.3)	457.4 (461.3)	399.2 (461.9)	297.2 (302.7)	417.1 (441.0)	**582.5** (558.7)	459.1 (499.7)
Riverraid	**938.0** (388.8)	828.6 (339.6)	823.3 (363.0)	922.1 (418.0)	880.3 (341.0)	909.8 (350.1)	863.1 (354.8)	795.8 (310.5)
Road Runner	10051.5 (4107.8)	10853.3 (5789.4)	10712.7 (3290.1)	9788.2 (4054.3)	9019.5 (4469.1)	**12323.2** (4177.1)	6638.6 (3602.0)	5763.6 (4681.0)
Sea Quest	260.3 (126.7)	301.0 (162.6)	290.4 (154.4)	308.5 (150.8)	**313.4** (141.1)	282.4 (164.7)	250.9 (162.0)	226.8 (125.4)
Up n Down	1012.7 (613.6)	999.6 (504.3)	865.8 (451.2)	912.1 (611.0)	990.3 (532.0)	972.5 (530.2)	**1072.8** (664.0)	1039.6 (573.0)

Table C.2: Average rewards and standard deviations (numbers in bracket) over the last 100,000 time steps for different uncertainty parameter of our proposed method. (7 random seeds)

Environment	DDQN	Fixed-j	ϵz-Greedy	DAR	TempoRL	B-DQN	UTE 1-step	UTE n-step	UTE Adaptive λ
Beam Rider	290.1	277.9	409.9	177.7	431.9	315.6	414.4	**439.5**	423.9
	(101.5)	(109.9)	(124.0)	(73.3)	(140.4)	(131.1)	(141.1)	(163.0)	(158.5)
Centipede	1574.7	1222.8	1431.7	1410.3	1958.0	1285.2	1437.1	**2190.1**	1829.9
	(1044.6)	(840.8)	(1169.1)	(982.8)	(1166.4)	(867.2)	(887.2)	(1073.0)	(969.6)
Crazy Climber	5265.8	3731.1	5295.1	2059.1	4885.5	2961.6	6761.9	**8175.6**	7046.3
	(4063.4)	(2997.2)	(3609.7)	(1225.1)	(3378.3)	(3080.0)	(5061.9)	(5790.4)	(5350.0)
Freeway	27.1	25.2	30.8	22.5	**32.1**	31.7	31.7	30.7	30.8
	(4.6)	(3.1)	(1.3)	(2.9)	(1.0)	(1.1)	(1.2)	(1.9)	(1.7)
Kangaroo	547.2	222.2	609.1	305.5	424.2	534.8	586.4	**728.5**	661.0
	(764.9)	(247.9)	(880.1)	(340.6)	(282.0)	(467.4)	(753.5)	(832.6)	(613.5)
Ms Pacman	509.8	388.8	597.2	371.3	495.6	516.1	**645.6**	551.6	537.6
	(204.6)	(201.6)	(261.5)	(166.5)	(245.1)	(206.5)	(267.1)	(225.5)	(263.8)
Pong	19.2	-20.3	**19.9**	-20.6	15.6	18.3	19.4	19.1	19.5
	(4.8)	(0.9)	(1.8)	(0.6)	(7.7)	(3.6)	(1.8)	(2.5)	(2.3)
Qbert	278.3	133.8	392.8	104.7	299.7	470.8	387.3	**582.5**	581.4
	(312.9)	(267.9)	(438.3)	(70.8)	(349.8)	(489.5)	(423.1)	(558.7)	(602.5)
Riverraid	740.5	360.9	**945.6**	102.8	807.7	843.8	942.2	938.0	890.5
	(291.9)	(231.6)	(457.9)	(66.0)	(354.7)	(407.2)	(319.3)	(388.8)	(330.7)
Road Runner	3277.0	1230.3	3733.8	845.5	8131.5	4976.8	4935.6	**12323.2**	10353.3
	(4470.3)	(1640.9)	(4716.5)	(791.9)	(4099.3)	(6032.6)	(5206.4)	(4177.1)	(3283.3)
Sea Quest	207.6	47.0	214.5	42.8	128.2	145.1	206.9	313.4	**320.3**
	(124.5)	(26.2)	(85.6)	(33.9)	(55.5)	(64.5)	(92.4)	(141.1)	(159.4)
Up n Down	536.4	594.8	823.1	348.7	641.5	383.2	911.5	**1072.8**	990.4
	(361.5)	(324.6)	(320.0)	(227.0)	(428.6)	(242.8)	(476.7)	(664.0)	(707.5)

Table C.3: Average rewards and standard deviations (numbers in bracket) over the last 100,000 time steps over Atari environments.

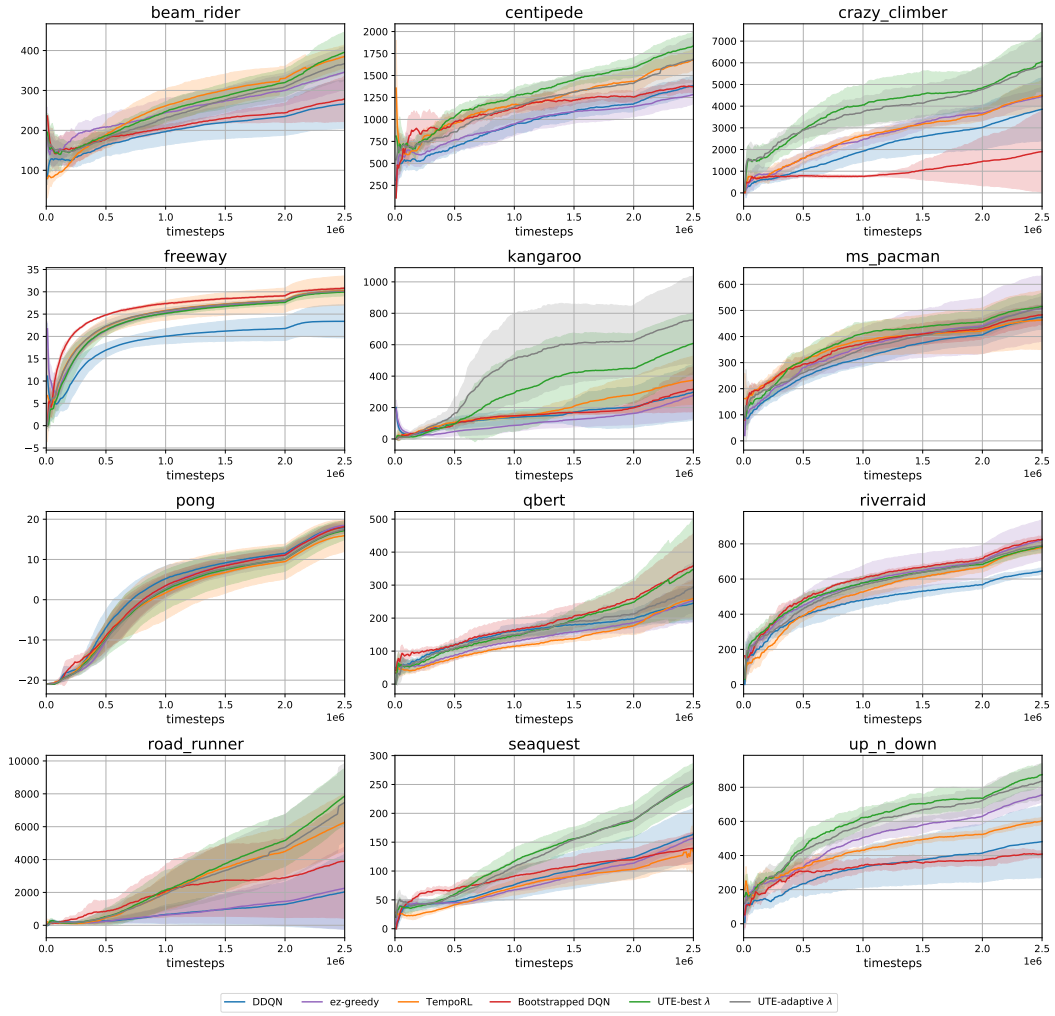


Figure C.1: Per-game Atari learning curves for ϵz-`Greedy` with the best μ for zeta distribution, `UTE` with the best uncertainty parameter, `UTE` with adaptive uncertainty parameter and the other baselines.(7 random seeds)