Master's Thesis

Learning Temporally-Extended Actions with Uncertainty-Aware Q-learning

불확실성을 고려한 반복 행동 정책 학습

Feburary 2023

Graduate School of Data Science
SEOUL NATIONAL UNIVERSITY
Data Science Major

JOONGKYU LEE

Learning Temporally-Extended Action with Uncertainty-Aware Q-learning

Adviser Min-Hwan Oh

Submitting a master's thesis of
Data Science

February 2023

Graduate School of Data Science
Seoul National University
Data Science Major

Joongkyu Lee

Confirming the master's thesis written by
Joongkyu Lee
February 2023

Chair	Wen-Syan Li	(Seal)
Vice Chair	Hyung-Sin Kim	(Seal)
Examiner	Min-Hwan Oh	(Seal)

Abstract

In reinforcement learning, temporal abstraction in action space is a common approach to simplifying the learning process of policies through temporally-extended courses of action. In recent work, temporal abstractions are often modeled as repeating the chosen action for a certain duration. A major drawback of the prior work on action repetition is that repetitions of suboptimal actions may lead to significant deterioration in performance. Hence, the degradation in performance that action repetition causes can be larger than the gains it provides. We propose a novel algorithm named *Uncertainty-aware Temporal Extension* (`UTE`), which leverages ensemble methods to estimate uncertainty when extending an action. Our uncertainty-aware learning framework can allow policies to be exploration-favor or uncertainty-averse. We empirically demonstrate the efficacy of `UTE` on both gridworld and Atari 2600 environments, exhibiting superior performances over alternative algorithms.

Keywords: Reinforcement Learning, Temporal Abstraction, Action Repeat, Uncertainty, Exploration

Student ID: 2021-27322

Contents

Abstract **i**

Contents **ii**

1 Introduction **1**

2 Related Work **4**

3 Preliminaries and Notations **7**

4 Method: Uncertainty-aware Temporal Extension **10**

4.1 Temporally-Extended Q-Learning 10

4.2 Option Decomposition 11

4.3 Ensemble-based Risk-Sensitive Action Repetition 12

4.4 n-step Q-Learning 14

5 Experiments **16**

5.1 Chain MDP 16

5.2 Gridworlds 19

5.3 Atari 2600: Arcade Learning Environment 22

6 Conclusion **26**

7 Appendix **27**

7.1 Details of Baselines 27

7.1.1 Fixed Repeat 27

7.1.2 Temporally-Extended ϵ-Greedy 27

7.1.3 Dynamic Action Repetition 28
7.1.4 TempoRL 28
7.1.5 Bootstrapped DQN 29
7.2 Implementation Details: `UTE` 29
7.2.1 Bootstrap with random initialization for option-value functions 29
7.2.2 Multi-step target for both action- and option-value functions 30
7.2.3 The Same Target for both action- and option- value functions 31
7.3 Experiments Details 33
7.3.1 Chain MDP experiment 33
7.3.2 Gridworlds experiment 34
7.3.3 Atari experiment 35
7.4 Limitations 37
7.5 Further Experimental Results 38
7.5.1 Chain MDP 38
7.5.2 Gridworlds 40
7.5.3 Atari 2600 46

Bibliography **57**

초 록 **63**

감사의 글 **65**

1 Introduction

Temporal abstraction is a promising approach to solving complex tasks in reinforcement learning (RL) with complex structures and long horizons Fikes et al. 1972; Dayan and Hinton 1992; Parr and Russell 1997; Sutton et al. 1999; Precup 2000; Bacon et al. 2017; Barreto et al. 2019; Machado et al. 2021. 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 Machado et al. 2021. One of the most prominent approaches for HRL is the *option* framework Sutton et al. 1999; Precup 2000, 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 fewer number of high-level decisions when solving a problem Stolle and Precup 2002; Biedenkapp et al. 2021. From a cognitive perspective, such observations are coherent with how humans learn, generalize from experiences, and perform abstraction over tasks Xia and Collins 2021.

There has been a line of works that propose repetition of action for an extended period as a specialized form of temporal abstraction Lakshminarayanan et al. 2017; Sharma et al. 2017; Dabney et al. 2020; Metelli et al. 2020;

Biedenkapp et al. 2021; Park et al. 2021.[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 Dabney et al. 2020; Biedenkapp et al. 2021. The extension length, the interaction steps to repeat the same action, is learned by an agent along with what action to execute Sharma et al. 2017; Biedenkapp et al. 2021. As shown by the improved empirical performances Dabney et al. 2020; Biedenkapp et al. 2021, these action repetition approaches can be well justified by the *commitment* to action for deriving a persistent and 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

[1] In fact, action repetition for a fixed number of steps was one of the strategies deployed in solving Atari 2600 games Mnih et al. 2015; Machado et al. 2018. 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.

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 significantly enhances the performances of the deep reinforcement learning agent in various environments. In comparison with the benchmarks, we show that our proposed method achieves the state-of-the-art performances, consistently outperforming the existing action repetition methods. Our contributions are:

- We present a novel framework that allows the agent to repeat an action 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.
- Our method yields a salient insight that it is beneficial to consider environment-inherent uncertainty preference. Some environments are uncertainty-favor (5.1), and some are uncertainty-averse (5.2).
- In a set of testing environments, we show `UTE` consistently outperforms all of the existing action-repetition baselines, such as `DAR`, ϵz-`Greedy`, `DQN`, `B-DQN`, in terms of final evaluation scores, learning speed, and coverage of state-spaces.

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 Precup 2000. The option Sutton et al. 1999; Precup 2000; Bacon et al. 2017 framework 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 Sutton et al. 1999. 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 steps Schoknecht and Riedmiller 2002. Action repetition has been widely explored in various literature Lakshminarayanan et al. 2017; Sharma et al. 2017; Dabney et al. 2020; Metelli et al. 2020; Biedenkapp et al. 2021; Park et al. 2021. Action repetition has empirically shown to induce deeper exploration Dabney et al. 2020 and lead to efficient learning by reducing the granularity of control Lakshminarayanan et al. 2017; Sharma et al. 2017; Metelli et al. 2020; Biedenkapp et al. 2021.

Action repetition can be implemented by deciding the extension length of an action which is either sampled from a distribution Dabney et al. 2020 or returned by a policy Lakshminarayanan et al. 2017; Sharma et al. 2017. However, these approaches to action repetition have limitations. The chosen action cannot be stopped during the execution of a predetermined extension length. This could lead to catastrophic failure when an agent enters a "risky" area which we will describe in Section 5.2. 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 Osband et al. 2016; Bellemare et al. 2016; Badia et al. 2020; Lee et al. 2022. There are mainly 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. Since we only cover deterministic environments such as Gridworlds and Atari games in this work, aleatoric uncertainty does not exist. As a result, the term "uncertainty" used in this paper mostly refers to epistemic uncertainty.

One mainstream of estimating the uncertainty in deep RL relies on bootstrapping. Osband et al. 2016 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. Kalweit and Boedecker 2017 substituted the vanilla DQNs with bootstrapped DQNs in model-based DDPG and achieved much higher rewards on continuous control tasks. Lee et al. 2022 merged the actor's and critic's bootstrapped

uncertainty estimations with Conservative Q Learning Kumar et al. 2020. Anschel et al. 2017; Peer et al. 2021 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. As in Bootstrapped DQN Osband et al. 2016, we use multiple randomly-initialized bootstrapped heads which stretch out from a shared network, simultaneously returning multiple estimates of the *option*-value function. The variance between these estimates is then used as a metric of uncertainty. Estimating the uncertainty using randomly-initialized heads is a simple but effective approach Da Silva et al. 2020; Bai et al. 2021. It can easily be adapted across different value-based learning algorithms Da Silva et al. 2020.

3 Preliminaries and Notations

In reinforcement learning, an agent interacts with an environment whose underlying dynamics is modeled by a Markov Decision Process (MDP) Puterman 2014. 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 Processes (SMDPs) model to incorporate the option framework Sutton et al. 1999; Precup 2000. A 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 $\mathcal{P}(\mathcal{X})$ denotes 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 each option conditioned on a given state. Our goal is to optimize the discounted return expected over all the trajectories when following π_ω starting from a given state s_0; then, define the value functions $V^{\pi_\omega}(s_0) = \mathbb{E}_{\pi_\omega}[\sum_{t=0}^{\infty} \gamma^t R_t \mid s = s_0]$, the action-value functions $Q^{\pi_\omega}(s_0, a) = \mathbb{E}_{\pi_\omega}[\sum_{t=0}^{\infty} \gamma^t R_t \mid s = s_0, a]$, or the option-value functions $\tilde{Q}^{\pi_\omega}(s_0, \omega) = \mathbb{E}_{\pi_\omega}[\sum_{t=0}^{\infty} \gamma^t R_t \mid s = s_0, \omega]$.

In general, options depend on the entire *history* between time step t when they were initiated and the current time step $t+k$, $h_{t:t+k} := s_t a_t s_{t+1} ... a_{t+k-1} s_{t+k}$.

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, K **do**
  Obtain initial state s from environment
  repeat
    $a \leftarrow \epsilon$-greedy $\text{argmax}_{a'}\ Q^{\pi_\omega}$
    Calculate $\hat{\mu}_{\pi_\omega}(s, \omega_{aj})$, $\hat{\sigma}^2_{\pi_\omega}(s, \omega_{aj})$ ▷ Eq.(4.3), (4.4)
    $j \leftarrow \text{argmax}_{j'} \{\hat{\mu}_{\pi_\omega}(s, \omega_{aj'}) + \lambda \hat{\sigma}_{\pi_\omega}(s, \omega_{aj'})\}$
    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

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 for otherwise. This action repeating option takes action a for j times and then terminates. Thus, the transition dynamics P_o and the reward function R_o can be defined as follows:

$$P_o(s'_{(j)} \mid s, \omega_{aj}) = \begin{cases} \prod_{k=0}^{j-1} P^a_{s'_{(k)}, s'_{(k+1)}} & \text{if reachable} \\ 0 & \text{otherwise} \end{cases}$$

$$R_o(s, \omega_{aj}) = \begin{cases} \sum_{k=0}^{j-1} \gamma^k R^a_{s'_{(k)}} & \text{if reachable} \\ 0 & \text{otherwise} \end{cases}$$

where $s'_{(k)}$ refers to a state after k steps from state s, "state $s'_{(j)}$ is reachable" means that the episode does not terminate while repeating action a for j times from state s, and $P^a_{s'_{(k)},s'_{(k+1)}} = P(s'_{(k+1)} \mid s_k, a)$ and $R^a_{s'_{(k)}} = R(s'_{(k)}, a)$ are the transition and reward model of $\mathcal{M}$. Note that $\mathcal{M}_o$ generalizes the original $\mathcal{M}$ where with extension length j of length 1, the transition and reward models of $\mathcal{M}_o$ reduce to the original transition function $P_o(s'_{(1)} \mid s, \omega_{a,1}) = P(s'_{(1)} \mid s, a)$ as well as the original reward function $R_o(s, \omega_{a,1}) = R(s, a)$.

One interesting point of this approach is that we can observe smaller extensions of semi-MDP transitions ($\tilde{j} \leq j$) Biedenkapp et al. 2021. Specifically, when repeating the action for j times from state s, we can also experience $(s \rightarrow s'_{(1)}), (s \rightarrow s'_{(2)}), \ldots, (s'_{(1)} \rightarrow s'_{(2)}), \ldots, (s'_{(j-1)} \rightarrow s'_{(j)})$, in total $\frac{j \cdot (j+1)}{2}$ transitions. We make use of these transitions for efficient learning.

4 Method: 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 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 Watkins and Dayan 1992, 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

$$Q^{\pi^*_\omega}(s,a) = \mathbb{E}_{s'_{(1)}\sim P}\Big[R(s,a) + \gamma \max_{a'} Q^{\pi^*_\omega}(s'_{(1)}, a')\Big], \tag{4.1}$$

$$\tilde{Q}^{\pi^*_\omega}(s,\omega_{aj}) = \mathbb{E}_{s'_{(j)}\sim P_o}\Big[R_o(s,\omega_{aj}) + \gamma^j \max_{\omega'} \tilde{Q}^{\pi^*_\omega}(s'_{(j)}, \omega')\Big], \tag{4.2}$$

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.

4.2 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 Thrun and Schwartz 1993; Zahavy et al. 2018. Therefore, we consider decomposed policy over option Biedenkapp et al. 2021, $\pi_\omega(\omega_{aj}|s) := \pi_a(a|s) \cdot \pi_e(j|s, a)$, in which an action policy $\pi_a(a|s) : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ assigns some probability to each action conditioned on a given state, and then an extension policy $\pi_e(j|s, a) : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{J})$ 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 $\text{argmax}_{a'}\ Q^{\pi_\omega}(s, a)$). 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 $|\mathcal{A}|+|\mathcal{J}|$, inducing stable learning. We provide empirical results

showing that decomposing option can stabilize the Q-learning in Figure 7.7a. 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 `UTE` can mitigate this issue by inducing risk sensitivity to the extension policy π_e.

Note that the equation $\max_a Q^{\pi^*_\omega}(s, a) = \max_\omega \tilde{Q}^{\pi^*_\omega}(s, \omega)$ holds trivially since the optimal policy over option π^*_ω incorporates the optimal action policy π^*_a (Refer Appendix 7.2.3). Thus, target value for the option selection of repeated actions is 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.(4.2). This can stabilize the learning process by sharing the same target.

4.3 Ensemble-based Risk-Sensitive Action Repetition

In the previous action repetition methods Lakshminarayanan et al. 2017; Sharma et al. 2017; Dabney et al. 2020; Biedenkapp et al. 2021, 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 a risk-sensitive manner. `UTE` is a risk-sensitive version of the `TempoRL` Biedenkapp et al. 2021, which not only selects an action in a state but also for how long to commit to that action.

The main idea of `TempoRL` is to explore deeper by skipping unimportant states. However, our main intuition is that simply repeating the chosen action is not enough. We may encounter undesirable states while repeating the action. Thus, 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.

In order to estimate uncertainty in our estimated option-value functions, we utilize the ensemble method, becoming prevalent in RL recently Osband et al. 2016; Da Silva et al. 2020; Bai et al. 2021. We use a network consisting of a shared architecture with $B \in \mathbb{Z}^+$ independent "heads" 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 `B-DQN` Osband et al. 2016 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:

$$\hat{\mu}_{\pi_\omega}(s, \omega_{aj}) := \frac{1}{B} \sum_{b=1}^{B} \tilde{Q}^{\pi_\omega}_{(b)}(s, \omega_{aj}) \tag{4.3}$$

$$\hat{\sigma}^2_{\pi_\omega}(s, \omega_{aj}) := \frac{1}{B} \sum_{b=1}^{B} (\tilde{Q}^{\pi_\omega}_{(b)}(s, \omega_{aj}))^2 - (\hat{\mu}_{\pi_\omega}(s, \omega_{aj}))^2 \tag{4.4}$$

Then, we define *risk-sensitive* extension policy π_e, which takes extension length j deterministically and risk-sensitively 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 risk sensitivity. The positive λ induces more aggressive exploration, and the negative one causes risk-averse exploration.

4.4 n-step Q-Learning

We make use of n-step Q-learning Sutton 1988 to learn both Q^{π_ω} and $\tilde{Q}^{\pi_\omega}$, whereas `TempoRL` Biedenkapp et al. 2021 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 Harutyunyan et al. 2016, 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.(4.1), `UTE` uses n-step Q-Learning to update Q^{π_ω}:

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

where $\bar{\theta}$ are the delayed parameters of action-value function Q^{π_ω} and $j \sim \pi_e(j_t | s_t, a_t)$. In general, n-step returns can be used to propagate rewards faster Watkins 1989; Peng and Williams 1994. It mitigates the overestimation problem Thrun and Schwartz 1993 in Q-learning as well Meng et al. 2021. We empirically

illustrate that n-step learning leads to faster learning in Figure 7.7b. 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 present three main experimental results of `UTE`: Chain MDP, Gridworlds, and Atari 2600 games Machado et al. 2018. First, we validate our reasoning to adjust uncertainty parameter λ depending on the degree of riskiness of an environment (Chain MDP, Gridworlds). Second, we show how well-tuned λ contribute to performance significantly in more complicated environments (Atari 2600). Note that n-step Q-learning for extension policy was not applied to the experiments of Chain MDP and gridworlds to compare the effect of risk-sensitive action repetition.

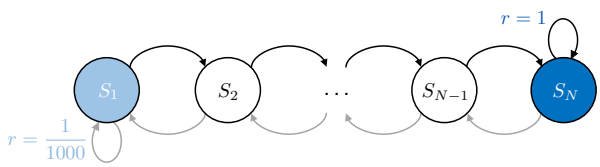


Figure 5.1 Chain MDP (Length: N)

5.1 Chain MDP

We experimented in the Chain MDP environment as described in Figure 5.1 Osband et al. 2016. In this toy environment, we test the ability of algorithms to perform deep exploration. Also, we evaluated the effect of the uncertainty parameter (λ) in our algorithm. First, the length of the chain is set to N. Each episode of interaction lasts $N + 8$ starting from the initial state (s_2). There are

two possible actions {left, right}. The agent needs to reach the right end (s_n) of the chain and perform the right action to receive a reward of 1. A deceptive small reward, 0.001, is given when the agent reaches the left end (s_1) of the chain and performs a left action. The reward is sparse and we need "deep" exploration in order to obtain a big reward. This problem becomes harder as the chain length N increases. We compared our method with other algorithms, ϵz-`Greedy` Dabney et al. 2020 and `TempoRL` Biedenkapp et al. 2021.

Setup. Each algorithm is trained for a total of 1,000 episodes with a fixed horizon length, $N + 8$, where N is the chain length. Therefore, at each episode, the agent can get a reward from zero to 10. We limited the maximum extension length as 10 for `TempoRL` and `UTE`. All policies use ϵ-greedy as an exploration strategy where the exploration rate, ϵ, is linearly decayed from 1.0 to 0.001 over $N \times 10$ steps. Additionally, for the hyperparameter μ of ϵz-`Greedy`, we set it to 1.25, which showed the best performance among the set of $\{1.25, 1.5, 2.0, 2.5, 3.0\}$. And for `UTE`, we use the best uncertainty parameters $\lambda = +2.0$ among the set of $\{-2.0, -1.0, 0.0, +1.0, +2.0\}$ (See Table 7.4 in Appendix for more details about hyperparameters).

Chain Length	10	30	50	70
ϵz-`Greedy`	0.654	0.427	0.434	0.131
`TempoRL`	0.904	0.740	0.246	0.052
`UTE` (ours)	**0.919**	**0.758**	**0.560**	**0.191**

Table 5.1 Normalized AUC on Chain MDP over 20 runs

Exploration-Favor. Table 5.1 summarizes the results on various levels of chain length in terms of normalized area under the reward curve (AUC), comparing `UTE` with the best uncertainty parameter (+2.0, the most optimistic λ) to ϵz-`Greedy` and `TempoRL`. A reward AUC value closer to 1.0 indicates

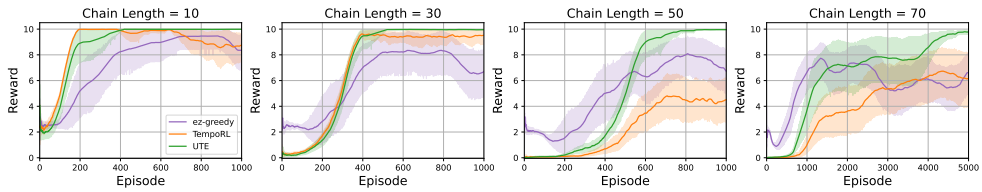
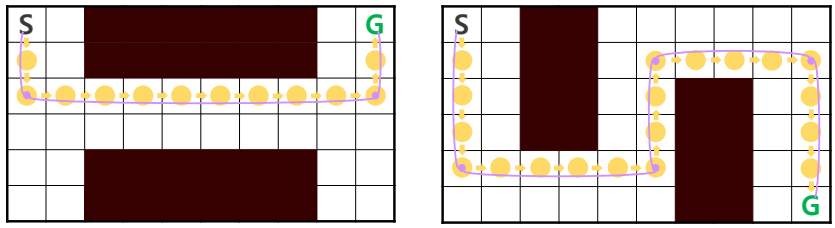


Figure 5.2 Learning curves for ϵz-`Greedy`, `TempoRL`, and `UTE` in Chain MDP environment with maximum extension length 10. The rewards are averaged over 20 runs with different seeds. `UTE` achieves the highest reward and stable learning throughout different chain lengths. Learning curve for $N = 70$ is presented with 5,000 training episodes to fully demonstrate the learning process.

that the agent was able to find the optimal policy faster. The total training episodes for calculating AUC was set to 1,000 across all chain lengths. The results in the table show that `UTE` outperforms the other two baselines notably throughout different chain lengths. In Figure 5.2, we depict learning curves for training episodes. The figure shows that `UTE` has a better exploration strategy which leads to higher reward even in the difficult settings (longer chain length) given an appropriate uncertainty parameter $\lambda = +2.0$. We also point out that `UTE` has a smaller variance than `TempoRL` after it has reached the optimal reward of 10. This is mainly because `UTE` can collect more diverse samples by exploratory extension policy π_e, which may lead to better generalization and more accurate approximation to the optimal option-value function.

More importantly, we can encode the exploration-favor strategy by adjusting the uncertainty parameter, λ. In Appendix Table 7.4, we verify that more positive λ achieves higher AUC scores. When the agent selects a random action by the ϵ-greedy action policy, it can explore deeper by being more optimistic, which leads to faster convergence to the optimal solution. An aggressive exploration strategy is beneficial because the environment has no risky area where the game terminates while repeating the action.



(a) Bridge **(b)** ZigZag

Figure 5.3 6×10 Gridworlds. Agents have to reach a fixed goal state (G) from a fixed start state (S) detouring the lava. Dots represent decision steps of policies with and without temporally-extended actions.

5.2 Gridworlds

In this section, we analyze the empirical behavior of the various algorithms in the gridworlds environment: *Lava* (Figure 5.3). It is a 6×10 grid with discrete states and actions. An agent starts in the top-left corner and must reach the goal to receive a positive reward (+1) while avoiding stepping into the lava (-1 reward) on its way. In contrast to the chain MDP environment, since we have a risky area "lava", an uncertainty-averse strategy must be preferred. We compare our method against vanilla `DDQN` Van Hasselt et al. 2016 ϵz-`Greedy` Dabney et al. 2020 and `TempoRL` Biedenkapp et al. 2021.

Setup. We trained all agents for a total of 3.0×10^3 episodes using the ϵ-greedy method with 3 different types of exploration rate schedule: linearly decaying from 1.0 to 0.0 over all episodes, logarithmically decaying, and fixed $\epsilon = 0.1$. We limited the maximum extension length to be 7. Though it is a tabular environment, we use neural networks to learn Q-value functions instead of tabular Q-learning to identify the impact of epistemic uncertainty more explicitly. For ϵz-`Greedy`, we fix μ to 1.25 which was the best among the

set of $\{1.25, 1.5, 2.0, 2.5, 3.0\}$, and for `UTE`, we use the best $\lambda = -1.5$ among the set of $\{-1.5, -1.0, 0.0\}$ (Refer Table 7.5 in Appendix for more details).

Env	ϵ decay	`DDQN`	`TempoRL`	ϵz-`Greedy`	`UTE`
Bridge	Linear	0.61	0.44	0.76	**0.86**
	Log	0.54	0.32	0.92	**0.92**
	Fixed	0.57	0.41	0.59	**0.83**
Zigzag	Linear	0.38	0.14	0.62	**0.84**
	Log	0.46	0.12	0.76	**0.89**
	Fixed	0.34	0.19	0.36	**0.76**

Table 5.2 Normalized AUC for reward across different ϵ exploration schedules over 20 random seeds.

Uncertainty-Averse. We compare our `UTE` to the other two baselines in terms of normalized area under the reward curve for three different ϵ-greedy schedules (see Table 5.2). A bigger AUC value implies that the agent can find the optimal policy quickly. Across all ϵ exploration strategies, `UTE` outperforms other methods while showing better performance as uncertainty parameter λ becomes smaller (large negative λ). Figure 5.4a shows that `UTE` learns faster than others. This result supports our argument that a pessimistic strategy is preferred in environments with unsafe regions.

Interestingly, the performance of `TempoRL` is a lot worse than the one described in the original paper Biedenkapp et al. 2021. This is because we use function approximation to estimate Q-values, rather than tabular Q-learning. Generally, uncontrolled or undesirable overestimation bias can be caused when using function approximation Moskovitz et al. 2021. Therefore, simply selecting extension length with the highest value leads to a catastrophic result in this case. Table 5.2 verifies the fact that it is beneficial to use a pessimistic extension policy in these environments, especially in function approximation. Moreover, in Appendix Table 7.5, we empirically show that more negative λ achieves higher AUC scores.

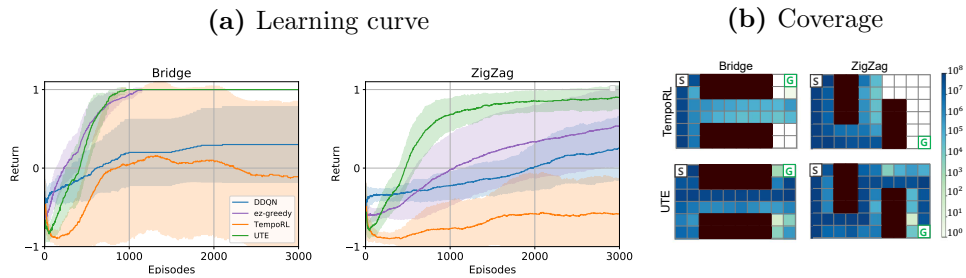


Figure 5.4 Learning curve (left) and coverage plots (right) on ZigZag environments, comparing `UTE` to `DDQN`, ϵz`-Greedy` and `TempoRL` on a logarithmically decaying ϵ-strategy. In coverage plots, blue represents states visited more often and white states rarely or never seen. See Appendix 7.5 for the expanded version of the figures.

Coverage. In Figure 5.4b, we present coverage plots comparing `UTE` and `TempoRL` on two types of Lava environments. We specify λ of `UTE` as -1.5, which exhibited good performance overall. The results show that `UTE` provides significantly better coverage over the state space. We can induce our algorithm to repeat sub-optimal action less by using a pessimistic extension policy. Owing to this, our agent can survive for a longer time, leading to better coverage.

Distribution of extension length. Figure 5.5 depicts the extension length distributions of `TempoRL` and `UTE` on Birdge and ZigZag with logarithmically decaying ϵ exploration schedule More red represents more repetitions. It shows that `UTE` prefers fewer repetitions than `TempoRL`. Also, we can figure out that a more pessimistic extension policy ($\lambda < 0$) leads to fewer repetitions. In a pessimistic extension policy, the agent tends to refrain from repeating the chosen action many times because the value of a distant state could be much more uncertain than that of a neighbor one.

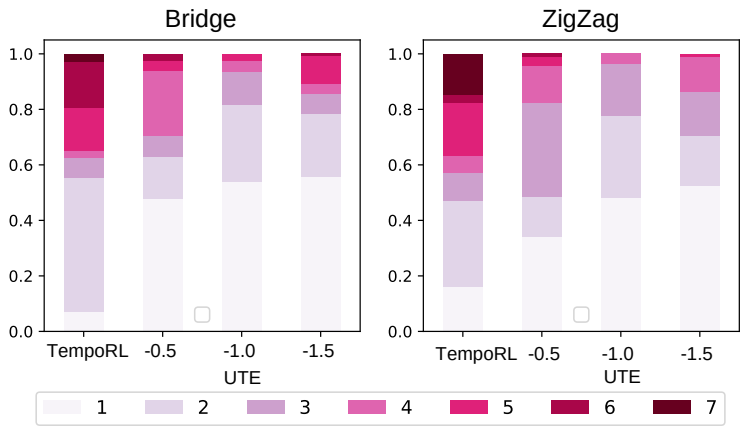


Figure 5.5 Distributions of extension length in Gridworlds.

5.3 Atari 2600: Arcade Learning Environment

In this subsection, we evaluate the performance of `UTE` on the Atari benchmark, comparing the following six baseline algorithms: i) vanilla `DDQN` Van Hasselt et al. 2016, ii) Fixed Repeat ($j = 4$), iii) ϵz-`Greedy` Dabney et al. 2020, iv) `DAR` (Dynamic Action Repetition Lakshminarayanan et al. 2017), v) `TempoRL` Biedenkapp et al. 2021 vi) `B-DQN` Osband et al. 2016. The Fixed Repeat is an algorithm that naively repeats the action a fixed amount of times. Note that action policies of all agents except for `B-DQN` have the same architecture as vanilla `DDQN` with a standard ϵ-greedy exploration scheme.

Setup. For direct comparison, we trained our agents on 12 different Atari environments including 5 games experimented in Biedenkapp et al. 2021 as well. Each algorithm is trained for a total of 2.5×10^6 training steps, which is only 10 million frames. All algorithms except `B-DQN` use a linearly decaying ϵ-greedy exploration schedule over the first 200,000 time-steps with a final ϵ fixed to 0.01. We evaluated all agents every 10,000 training steps and evaluated for 3 episodes with a very small ϵ exploration rate (0.001). We used *OpenAi*

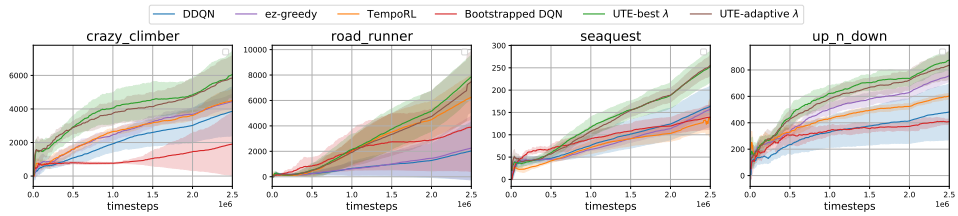


Figure 5.6 Learning curves of `UTE` with best λ, `UTE` with adaptive λ and other baseline algorithms on Atari environments. The shaded area represents the standard deviation over 7 random seeds.

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

Gym's Atari environment with 4 frame-skips following Bellemare et al. 2013. For maximal extension length, we set it to 10. To be as fair as possible, we used the per-game best hyperparameters μ and λ for ϵz-`Greedy` and `UTE` respectively (Refer Table 7.7 and 7.8 in Appendix).

Uncertainty-Awareness. In Figure 5.6, we give representative examples of per-game performance for `UTE` and other algorithms (see Figure 7.8 for other environments). And Table 5.3 summarizes the the results of the games in terms of average rewards over the last 100,000 time steps (Refer Table 7.9 for other environments). Overall, `UTE` achieves higher final rewards than other agents (Refer Table 7.6 for DQN-normalized version; DQN-normalized score is defined in Appendix 7.5.3). These results make a strong point that if λ is tuned suitably according to the environment, our method shows significantly improved performance than existing action repetition methods (`DAR`, ϵz-`Greedy`

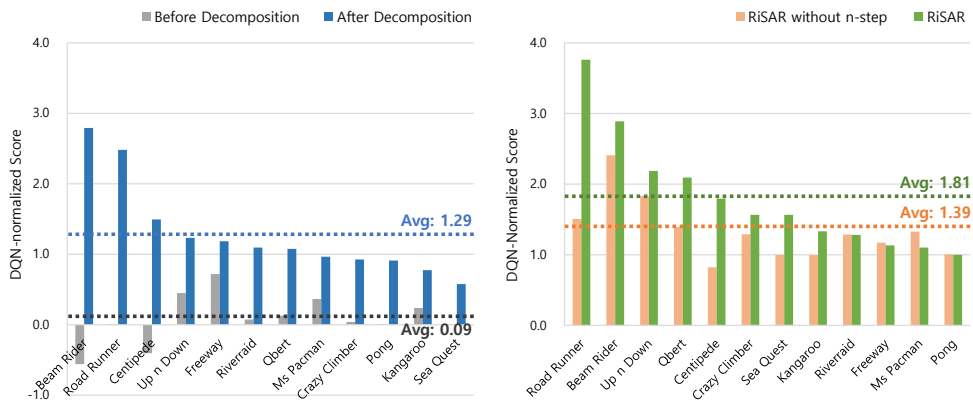
and TempoRL) as well as a deep exploration algorithm (B-DQN). On top of that, we found that the Fixed Repeat algorithm fails at learning in most games. Hence, it is crucial to learn a good extension policy for higher performance.

Adaptive Uncertainty Parameter λ. Instead of fixing λ for the whole learning process, we propose a method to adaptively choose λ using a non-stationary multi-arm bandit algorithm Badia et al. 2020. Let Λ be the pre-specified set of uncertainty parameters. Then, at the beginning of each episode k, the bandit chooses an arm $\lambda_k \in \Lambda$ and receives feedback of episode returns $R_k(\lambda_k)$. Since the reward signal $R_k(\lambda_k)$ is non-stationary, we use a sliding-window UCB with ϵ_{ucb}-greedy exploration (details of the algorithms are provided in Appendix 7.5.3). Figure 5.6 and Table 5.3 demonstrate that UTE with adaptive λ still out performs other baselines. These results are promising since we don't have to specify the λ prior, relieving the burden of tuning hyperparameters.

Effect of Decomposition. Our method formulates joint optimization of the action and the extension length as a two-level optimization problem. The action is selected based on Q^{π_ω} and then the extension length is selected based on $\tilde{Q}^{\pi_\omega}$ sequentially. Figure 7.7a shows the effect of the decomposition. Without decomposition, the size of search space is $|\mathcal{A}| \times |J|$, which leads to a catastrophic performance. In some environments such as *Beam Rider* and *Road Runner*, the DQN-normalized scores are negative, which means the agent is worse than a random policy. Overall, We can see that decomposition of action and extension length selection improves the performance significantly.

Effect of n-step Learning. We conducted an ablation study to analysis n-step Q-learning for the action-value function Q^{π_ω}. Figure 7.7b shows the effect of an n-step target for Q^{π_ω} in terms of the DQN-normalized score. This

result illustrates that applying n-step learning is beneficial in most games, which shows a 30.2% improvement (from 1.39 to 1.81) after it has been applied. Especially in games with relatively sparse rewards such as *Road Runner* and *Centipede*, it dramatically enhanced the performance. This is because rewards can be propagated faster using n-step returns. These results are promising since we don't have to specify the λ prior, relieving the burden of tuning hyperparameters.



(a) Decomposition Effect **(b)** Multi-step Target Effect for Q^{π_ω}

Figure 5.7 Ablation studies for decomposition effect (left) and n-step learning effect for action-value function Q^{π_ω}. The negative score indicates that the policy is worse than a random policy. (5 random seeds)

6 Conclusion

We presented `UTE` that learns to repeat actions while explicitly considering the uncertainty over the Q-value estimates of the states reached under the repeated-action option. We empirically showed that it is crucial to consider environment-inherent uncertainty, and a well-suited uncertainty parameter λ significantly outperforms other existing action repetition algorithms such as `DAR`, ϵz-`Greedy` and `TempoRL` as well as a traditional deep exploration algorithm such as `B-DQN` in various environments. 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 actions.

7 Appendix

7.1 Details of Baselines

7.1.1 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 (see 7.6 for performance of other js). The reason for evaluating this naive method is to confirm that this approach fails so that learning extension length is crucial.

7.1.2 Temporally-Extended ϵ-Greedy

ϵz-`Greedy` Dabney et al. 2020 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 Dabney et al. 2020 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.

7.1.3 Dynamic Action Repetition

`DAR` Lakshminarayanan et al. 2017 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.

7.1.4 TempoRL

`TempoRL` Biedenkapp et al. 2021 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.

7.1.5 Bootstrapped DQN

`B-DQN` Osband et al. 2016 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. 2016 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.

7.2 Implementation Details: `UTE`

7.2.1 Bootstrap with random initialization for option-value functions

Formally, we consider an ensemble of B option-value functions, $\{\tilde{Q}^{\pi_\omega}_{(b)}\}^{B}_{b=1}$, where π_ω denotes the policy over option. To train the ensemble of option-value functions $\tilde{Q}^{\pi_\omega}_{(b)}$, we use two mechanisms to enforce diversity between

these Q-functions Efron 1982; Osband et al. 2016: The first mechanism is random-initialization of model parameters for each option-value functions to induce initial diversity in the models. The second mechanism is to train each Q-function with different samples. Specifically, in each timestep t, each b^{th} Q-function is trained by multiplying binary mask $m_{t,b}$ to each objective function, where the binary mask $m_{t,b}$ is sampled from the Bernoulli distribution (p) with parameter $\beta \in (0, 1]$. In our experiments, we use $p = 0.5$ for the parameter of the Bernoulli distribution.

7.2.2 Multi-step target for both action- and option-value functions

We learn parameterized estimates of Q-value functions, an action-value function $Q^{\pi_\omega}(s, a; \theta) \approx Q^{\pi^*_\omega}(s, a)$ and an option-value function $\tilde{Q}^{\pi_\omega}(s, \omega_{aj}; \phi) \approx \tilde{Q}^{\pi^*_\omega}(s, \omega_{aj})$, using neural networks. We use two different neural network function approximators parameterized by θ and ϕ respectively. For stability, we integrate double Q-learning Van Hasselt et al. 2016 technique. We use multi-step Q-learning to update both action-value function Q^{π_ω} and option-value function $\tilde{Q}^{\pi_\omega}$. The following equations represent Bellman residual errors of action- and option-value functions respectively:

$$\mathcal{L}_{Q^{\pi_\omega}}(\theta) = \mathbb{E}_{\tau_t \sim \mathcal{R}}\left[(Q^{\pi_\omega}(s_t, a_t; \theta) - \sum_{k=0}^{n-1} \gamma^k r_t - \gamma^n \max_{a'} Q^{\pi^*_\omega}(s_{t+n}, a'; \bar{\theta}))^2 \,\middle|\, n \leq j \sim \pi_e \right] \tag{7.1}$$

$$\mathcal{L}_{\tilde{Q}^{\pi_\omega}}(\phi) = \mathbb{E}_{\tau_t \sim \mathcal{R}}\left[\sum_{b=1}^{B} \left[m_{t,b} (\tilde{Q}^{\pi_\omega}_{(b)}(s_t, \omega_{aj}; \phi) - \sum_{k=0}^{j-1} \gamma^k r_t - \gamma^j \max_{a'} Q^{\pi^*_\omega}(s_{t+j}, a'; \bar{\theta}))^2 \right] \right] \tag{7.2}$$

where τ_t is a multi-step transition trajectory sampled from a replay buffer $\mathcal{R}$, $m_{t,b}$ is a binary bootstrap mask, and $\bar{\theta}$ are the delayed parameters of

action-value function Q^{π_ω}. The delayed parameters are the parameters of the target network for action-value function Q^{π_ω}. The target network with the delayed parameters is the same as the online network except that its parameters are copied every τ step from the online network, and kept fixed on all other steps Mnih et al. 2015. Note that n for n-step learning in Eq.(7.1) has to follow the current extension policy π_e. Therefore, in order to update action-value function Q^{π_ω} in Eq.(7.1), we only use trajectory samples in which the extension length is smaller than or equal to the output of current extension policy, i.e. $n \leq j \sim \pi_e(j_t \mid s_t, a_t)$.

One interesting point of above equations is that we use the same target value for both Q-functions: Q^{π_ω} in Eq.(7.1) and $\tilde{Q}^{\pi_\omega}$ Eq.(7.2). Trivially, we can demonstrate that the target value for the option selection of repeated actions is the same as one for single-step action selection within the option, i.e. $\max_{a'} Q^{\pi^*_\omega}(s_{t+j}, a') = \max_{\omega'} Q^{\pi^*_\omega}(s_{t+j}, \omega'_{aj})$. By using the same target value, we can stabilize the learning process.

7.2.3 The Same Target for both action- and option- value functions

In Subsection 4.1, we argue that we can use the same target for both types of Q values. The following proposition formalizes the statement. Though it is a trivial result, we simply present the proof of it for better comprehension.

Proposition 7.1 *In a Semi-Markov Decision Processes (SMDPs), 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. 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 Sutton et al. 1999, which returns an action a when executing an option ω_{aj} at state s. Then, option-value functions can be written as:

$$\begin{aligned}
\tilde{Q}^{\pi_\omega}(s, \omega_{aj}) &= \sum_{a'} \pi_U(a'|s, \omega_{aj}) Q_U(s, \omega_{aj}, a') \\
&= \sum_{a'} \mathbb{1}_a Q_U(s, \omega_{aj}, a') \\
&= Q_U(s, \omega_{aj}, a), \qquad (7.3)
\end{aligned}$$

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

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

where the second equality holds since ω_{aj} is determined by an action a and extension length j, the third equality is by Eq.(7.3), and the last equality holds since π^*_ω is the optimal policy. This concludes the proof. □

7.3 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. Our Chain MDP environment and `B-DQN` baseline implementation is based on code from Touati et al. 2020. The license for this asset is Attribution-NonCommercial 4.0 International.

Gridworlds and Atari environment settings along with `TempoRL` baseline implementation is from Biedenkapp et al. 2021. This asset is licensed under Apache License 2.0. The Arcade Learning Environment (ALE) Bellemare et al. 2013 for Atari games is licensed under the GNU General Public License Version 2. Baseline codes can be found at `https://github.com/facebookresearch/RandomizedValueFunctions` and `https://github.com/automl/TempoRL` respectively.

7.3.1 Chain MDP experiment

Network Architecture. In Chain MDP experiment, all agents, i.e. ϵz-`Greedy`, `TempoRL`, and `UTE`, use simple DQN architecture Mnih et al. 2015 for their action policy. The network consists of 3 dense layers with a ReLU activation function. The number of hidden nodes is set to 16 for all dense layers.

`TempoRL` has another output stream that combines a hidden layer with 10 units together with the output of the second fully connected layer. It is followed by

Hyper-parameter	Value
Discount rate	0.999
Target update frequency	500
Initial ϵ	1.0
Final ϵ	0.001
ϵ time-steps	N $\times$ 100
Loss Function	Huber Loss
Optimizer	Adam
Learning rate	0.0005
Batch Size	64
Replay buffer size	5×10^4
Extension replay buffer size	5×10^4
Number of ensemble heads	10
Max extension length (J)	10, 15, 20
Uncertainty parameter (λ)	-2, -1, 0, 1, 2

Table 7.1 Hyper-parameters used for the Chain MDP experiments

a fully connected layer that outputs predicted Q-values, extension lengths.

In order to implement extension policy π_e of `UTE`, the agent has another ensemble network of 10 identical neural networks. Each of these 10 ensemble networks is a 3-layer neural network with fully connected layers with 26 hidden units, where the input is a concatenation of the state and the chosen action.

7.3.2 Gridworlds experiment

Network Architecture. In the Gridworlds experiment, all agents, i.e. `DDQN`, `TempoRL` and `UTE`, were trained using deep Q-network Mnih et al. 2015 with 3 dense layers. The number of each hidden node is 50 and ReLU activation function was used for non-linearity. All of the agents were implemented using double DQN Van Hasselt et al. 2016. Both `TempoRL` and `UTE` agents have separate network for extension policy.

For extension policy, `TempoRL` uses a single 3-layer neural network with fully connected layers of 50, 50, and 50 units, whereas our `UTE` uses 10 duplicated networks of a 3-layer neural network with fully connected layers of 50, 50, and 50 units. The input of extension policy is a concatenation of the state and the chosen action, which is the same as the Chain MDP experiment.

Hyper-parameter	Value
Discount rate	0.99
Initial ϵ	1.0
Final ϵ	0.0
ϵ time-steps	50
Loss Function	MSE Loss
Optimizer	Adam
Learning rate	0.001
Batch Size	64
Replay buffer size	10^6
Extension replay buffer size	10^6
Number of ensemble heads	10
Max extension length (J)	7
Uncertainty parameter (λ)	-1.5, -1.0, -0.5

Table 7.2 Hyper-parameters used for the Gridworlds experiments

7.3.3 Atari experiment

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 Mnih et al. 2015 with the target network Van Hasselt et al. 2016. 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. 2015. 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. 2021. On top of DQN architecture Mnih et al. 2015, 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 Mnih et al. 2015. However, it has 10 heads branching off independently Osband et al. 2016. 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` Biedenkapp et al. 2021. 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 7.3 Hyper-parameters used for the Atari experiments

7.4 Limitations

One of the limitations of our work is the total training frames in the Atari games. We have trained our agents for only 10 million frames which is the same as `TempoRL` paper Biedenkapp et al. 2021. However, 10 million frames may not be enough to fully train the agent for some games. We acknowledge

that the final performance of the fully trained agent may differ from our result. This issue stems from the limited computation resources we have. To train the agent on the same game for multiple runs to obtain reliable results, we have set the total training frames to be 10 million.

7.5 Further Experimental Results

7.5.1 Chain MDP

Uncertainty Parameter. Table 7.4 shows the effect of the uncertainty parameter on normalized AUC score for 1,000 training episodes. We can see that an exploration-favoring high uncertainty parameter is beneficial in the Chain MDP environment. The longer the chain length, the more sensitive it becomes sensitive to the uncertainty parameter. In the chain length of 70, `UTE` with uncertainty parameter +2 is a lot better than the one with uncertainty parameter -2. This result indicates that optimistically repeating the chosen action could lead to good performance if there is no risky area in the environment. The Table also shows that for ϵz-`Greedy` with $\mu = 1.25$ performed the best, and we used the value for the experiments.

Maximum Repeat J. We can also predispose the agent to repeat actions in larger numbers by increasing another parameter, the maximum extension length J. However, increasing the maximum extension length is not always a good solution as it increases the size of the set of extension lengths, $|\mathcal{J}|$, slowing down the learning process. As exhibited in Figure 7.1, the small maximum extension length is detrimental to the agent's performance. Note that the degree of exploration of ϵz-`Greedy` is affected by the hyperparameter μ for zeta distribution, and the value is fixed to 1.25 throughout the chain MDP

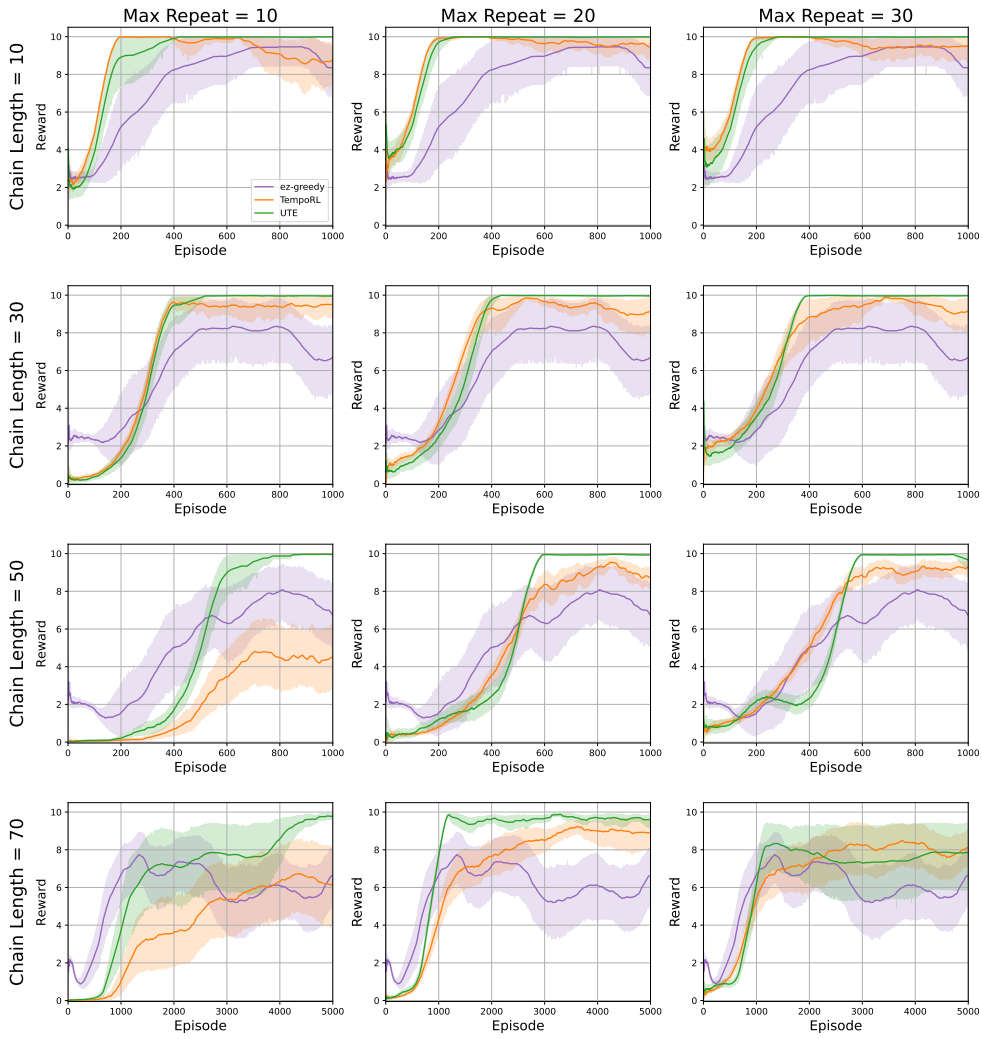


Figure 7.1 Training reward for ϵz-`Greedy`, `TempoRL`, and `UTE` in the Chain MDP environment. This is an expanded version of Figure 5.2 where the horizontal axis represents the maximum extension length of 10, 15, and 20 from left to right, respectively. The vertical axis represents the chain length of 10, 30, 50, and 70 from top to bottom. Learning curve for $N = 70$ is presented with 5,000 training episodes. (20 random seeds)

experiments. While TempoRL is sensitive to extension length especially when the chain length is long, our UTE is quite robust to changes in extension length. We can see that UTE agent reaches the highest final performance compared to other agents.

Chain Length	TempoRL	ϵz-Greedy (μ)					UTE (λ)				
		1.25	1.5	2.0	2.5	3.0	-2.0	-1.0	0.0	1.0	2.0
10	0.90	**0.65**	0.61	0.48	0.46	0.46	0.88	0.91	0.90	0.92	**0.92**
30	0.74	**0.43**	0.42	0.42	0.38	0.27	0.45	0.55	0.62	0.73	**0.76**
50	0.25	**0.43**	0.40	0.23	0.17	0.05	0.07	0.05	0.37	0.47	**0.67**
70	0.05	**0.13**	0.12	0.03	0.04	0.1	0.01	0.01	0.01	0.07	**0.19**

Table 7.4 Normalized AUC for reward and its standard deviation inside the parenthesis across different agents over 20 random seeds. Maximum extension length for TempoRL and UTE is set to 10. The numbers in the brackets of UTE represent the uncertainty parameter, λ. We can see that greater uncertainty parameters show better performance, especially in the difficult settings where the chain length is long.

7.5.2 Gridworlds

In the following additional results, we supplement one more environment called Cliff in addition to Bridge and Zigzag. Same as others, the Cliff is discrete, deterministic, and has a 6×10 size with sparse rewards. Note that all agents are implemented by function approximation, not tabular setting.

Uncertainty Parameter Table 7.5 shows an uncertainty-averse (negative uncertainty parameter) is beneficial in the Gridworlds environment where there are risky area (Lava). Combined with the result of Figure 7.3, we empirically verified that more negative λ induce more pessimistic behavior, which can lead to better performance. The Table also shows that for ϵz-Greedy with $\mu = 1.25$ performed good in overall, and we used the value for the experiments.

Learning curve. In Figure 7.4, we plot all the learning curves of three

Environment		ϵz-Greedy (μ)					UTE (λ)		
		1.25	1.5	2.0	2.5	3.0	-0.5	-1.0	-1.5
Cliff	Linear	**0.80**	0.79	0.79	0.78	0.77	0.89	0.88	**0.90**
	Log	0.92	**0.93**	0.91	0.84	0.81	0.94	0.95	**0.96**
	Fixed	**0.65**	0.64	0.64	0.64	0.63	0.84	0.84	**0.85**
Bridge	Linear	0.75	0.75	**0.76**	0.74	0.73	0.83	0.84	**0.86**
	Log	**0.92**	0.92	0.91	0.90	0.89	0.85	0.88	**0.92**
	Fixed	**0.59**	0.57	0.58	0.55	0.55	0.72	0.82	**0.83**
Zigzag	Linear	0.62	**0.63**	0.61	0.61	0.62	0.73	0.82	**0.84**
	Log	**0.76**	0.72	0.63	0.61	0.52	0.66	0.86	**0.89**
	Fixed	0.36	0.40	0.41	0.42	**0.43**	0.62	0.70	**0.76**

Table 7.5 Normalized AUC of reward for varying hyperparameters of μ in ϵz-Greedy and λ in UTE on a logarithmically decaying ϵ-strategy in the Gridworlds environment. (20 random seeds)

agents across three different Lava environments and three different exploration schedules. In this result, we observe that UTE converges to the optimal solution faster than other baselines, showing low standard deviations. Also, our UTE is robust to varying exploration strategies even using sub-optimal ones such as Fixed ϵ. One more interesting point is that TempoRL performs worse than vanilla DDQN. These results contradict the ones of Biedenkapp et al. 2021, which experimented in a tabular setting instead of function approximation. Generally, when using function approximation to estimate Q-values, it is more likely to choose sub-optimal action a. Therefore, in this case, it is necessary to consider uncertainty in estimated values for safely repeating the chosen action. By inducing pessimism ($\lambda < 0$) to the extension policy π_e, the agent can repeat the chosen sub-optimal action a less, which leads to a safer learning.

Coverage. Figure 7.2 illustrates state visitation coverage of different agents for 3 different gridworlds environments. Both TempoRL and UTE repeat the chosen actions, which can lead to a better exploration. However, TempoRL is not any better than vanilla DDQN, whereas our UTE shows significantly better coverage. This implies that a pessimistic extension policy inducing safe exploration can result in better coverage of the state space.

Distribution of extension length. The full version of distributions of extension length is presented in Figure 7.3. It shows that the TempoRL selects large extension length, close to 7, more often than our UTE. We can see UTE maneuver at a smaller scale as the uncertainty parameter decreases to induce more pessimistic behavior. The portion of small extension lengths tends to increase as being more pessimistic.

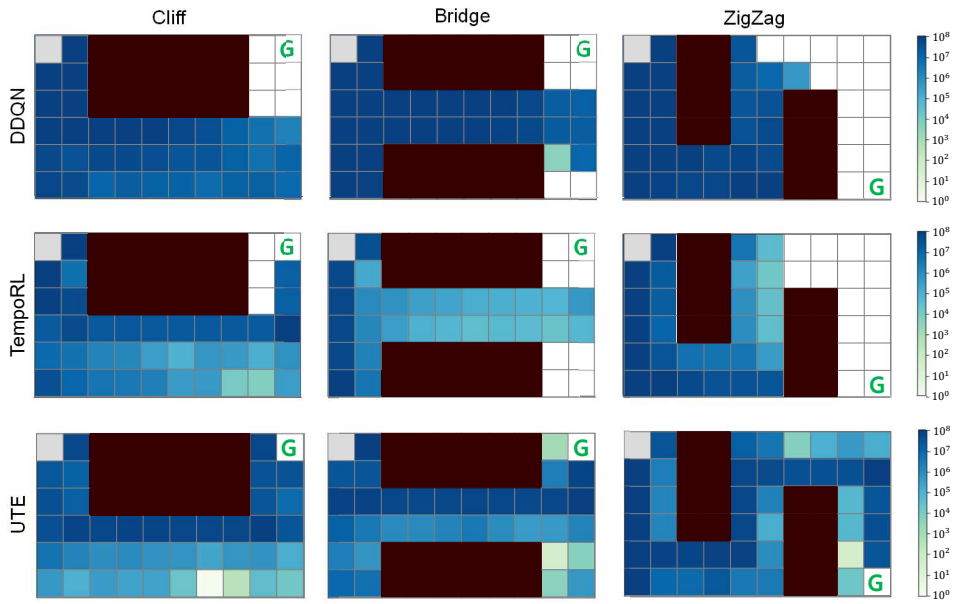


Figure 7.2 Coverage plots on all Lava environments, comparing `UTE` (λ=-1.5) to `DDQN` and `TempoRL` on logarithmically decaying ϵ-strategy (Blue represents states visited more often and white states rarely or never seen).

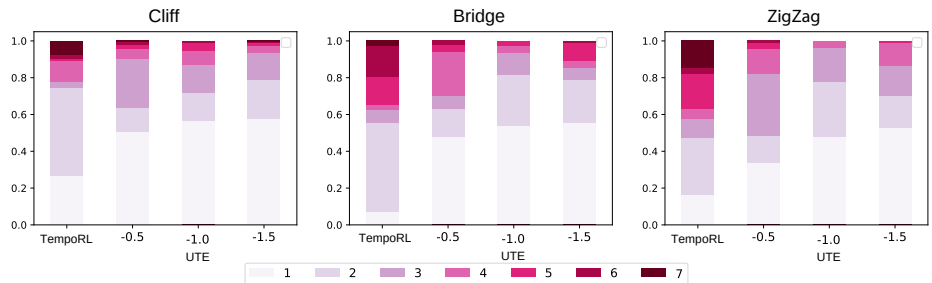


Figure 7.3 Distribution of extension length for three Lava environments with logarithmically decaying ϵ exploration schedule. More red represents more repetitions.

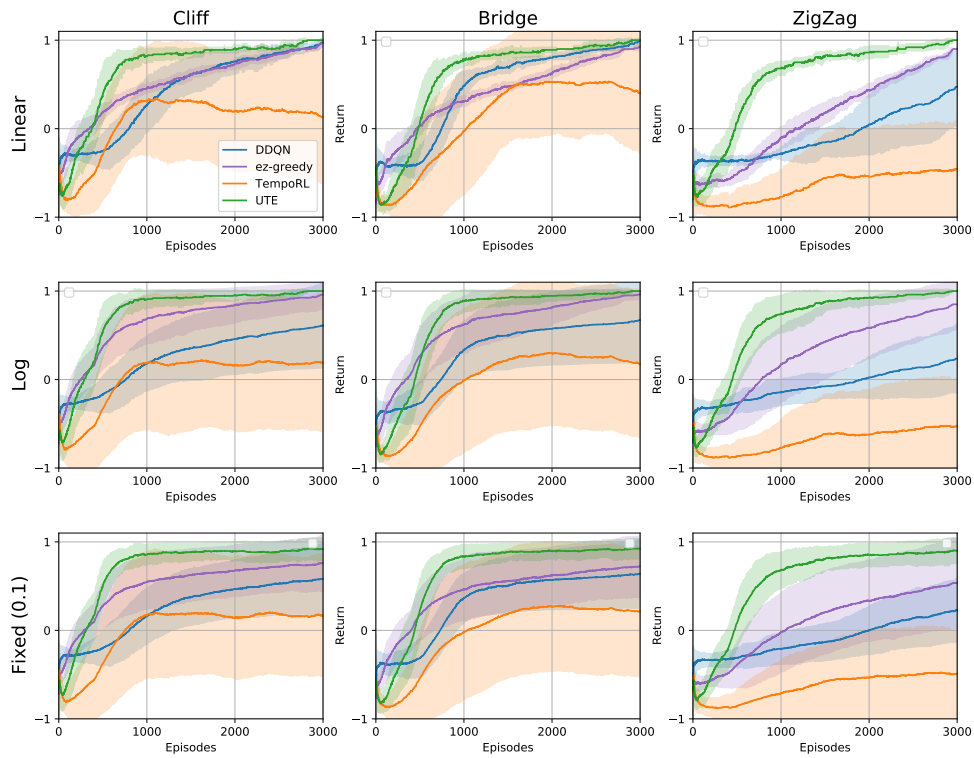


Figure 7.4 Learning curves across three Lava environments and three different ϵ-decaying exploration strategies, comparing `UTE` with `DDQN`, ϵz-`Greedy` and `TempoRL`. Shaded areas represent the standard deviations over 20 random seeds.

7.5.3 Atari 2600

DQN-normalized score. The DQN-normalized score is defined as

$$score = \frac{\text{agent - random}}{\text{DQN - random}}$$

where agent, random, and DQN are the per-game mean rewards over the last 100,000 time steps for the agent, a random policy, and a `DDQN` respectively. We used this DQN-normalized score to summarize the results across various games.

The results in Table 7.6 show mean DQN-normalized scores over the last 100,000 time steps of each game. The reason that we use this metric instead of the human-normalized score is that we have only trained the agent for 10 million frames due to limited resources. The agents were not fully trained to be compared with the human scores, so we normalized the score against `DDQN`. A score below 0 means that the performance is worse than that of random policy while the score greater than 1 indicates it achieves higher performance compared to that of `DDQN` agent. Overall, our `UTE` with the best uncertainty parameter performs best compared to other baselines (see Table 7.6 for more details). `UTE` achieves a score 81% higher than that of `DDQN` and 40% higher than `TempoRL`.

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 7.8, 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

Environment	DDQN	Fixed-j	ϵz-Greedy	DAR	TempoRL	B-DQN	UTE (1-step)	UTE (n-step)
Beam Rider	1.00	0.85	2.50	0.05	2.79	1.32	2.41	**2.89**
Centipede	1.00	0.55	0.82	0.79	1.49	0.63	0.82	**1.71**
Crazy Climber	1.00	0.70	1.01	0.38	0.93	0.55	1.29	**1.56**
Freeway	1.00	0.93	0.94	0.83	**1.18**	1.17	1.17	1.13
Kangaroo	1.00	0.41	1.11	0.56	0.78	0.98	1.07	1.21
Ms Pacman	1.00	0.71	**1.21**	0.67	0.97	1.02	**1.33**	1.10
Pong	1.00	0.02	**1.02**	0.01	0.91	0.98	1.00	1.00
Qbert	1.00	0.48	1.41	0.38	1.08	1.69	1.39	**2.09**
Riverraid	1.00	0.46	**1.29**	0.09	1.10	1.15	1.29	1.28
Road Runner	1.00	0.38	1.14	0.26	2.48	1.52	1.51	**3.76**
Sea Quest	1.00	0.14	1.04	0.12	0.58	0.67	1.00	**1.54**
Up n Down	1.00	1.13	1.63	0.58	1.23	0.66	1.83	**2.05**
Average	1.00	0.56	1.28	0.35	1.29	1.03	1.34	**1.81**

Table 7.6 DQN-normalized performance averaged over last 100,000 time steps for UTE with the best uncertainty parameter and other baselines. (7 random seeds)

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 7.7 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.

Multi-arm Bandit for Choosing λ. To choose λ adaptively, we used multi-armed bandit (MAB) algorithm Garivier and Moulines 2008 with sliding-window upper confidence bound (UCB) as described in Section 5.3. Therefore, here we describe the bandit algorithm in detail. The following method is the same as Appendix Section D in Badia et al. 2020.

At each episode $k \in [K]$, a N-armed bandit selects an arm A_k among the pre-defined set of arms $\mathcal{A} := \{0, \ldots, N-1\}$ by a policy π. The policy π depends

on the sequence of previous histories (actions and rewards). Then, it receives a reward $R_k(A_k) \in \mathbb{R}$ from the environment.

The objective of a MAB algorithm is to learn a policy π that minimizes the expected regret as follows:

$$\mathbb{E}_\pi \left[\sum_{k=0}^{K-1} \max_A R_k(A) - R_k(A_k) \right].$$

When reward distribution is stationary, i.e. $R_k(\cdot) = R(\cdot)$, the traditional UCB algorithm can be applied. Define the number of time episodes an arm $a \in \mathcal{A}$ has been selected in episode k as:

$$N_k(a) = \sum_{k'=0}^{k-1} \mathbb{1}(A_{k'} = a),$$

where $\mathbb{1}(A_k = a)$ is an indicator function. We can estimate the empirical mean reward of an arm a as:

$$\widehat{\mu}_k(a) = \frac{1}{N_k(a)} \sum_{k'=0}^{k=1} R_{k'}(a) \mathbb{1}(A_{k'} = a).$$

Then, we select an arm using the UCB algorithm as follows:

$$\begin{cases} \forall 0 \leq k \leq N-1, \quad A_k = k, \\ \forall N \leq k \leq K-1 \quad A_k = \text{argmax}_{a \in \mathcal{A}} \, \widehat{\mu}_{k-1}(a) + \beta \sqrt{\frac{\log(k-1)}{N_{k-1}(a)}}. \end{cases}$$

However, if reward distribution is non-stationary, the UCB algorithm cannot be directly applied due to the change in reward distribution. One of the common solutions to the non-stationary case is to use a sliding-window UCB. Let $\tau \in \mathbb{Z}^+$ be the size of window such that $\tau < K$. The number of time episodes an arm

$a \in \mathcal{A}$ has been played in episode k for a window size τ as:

$$N_k^\tau(a) = \sum_{k'=\max(0,k-\tau)}^{k-1} \mathbb{1}(A_{k'} = a).$$

Define the empirical mean reward of an arm a for a window size τ as:

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

Then, we select an arm using the sliding window UCB as follows:

$$\begin{cases} \forall 0 \leq k \leq N-1, & A_k = k, \\ \forall N \leq k \leq K-1 & A_k = \text{argmax}_{a \in \mathcal{A}}\, \widehat{\mu}_{k-1}^\tau(a) + \beta\sqrt{\frac{\log(k-1)}{N_{k-1}^\tau(a)}}. \end{cases}$$

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

$$\begin{cases} \forall 0 \leq k \leq N-1, & A_k = k, \\ \forall N \leq k \leq K-1 \text{and} U_k \geq \epsilon_{ucb} & A_k = \text{argmax}_{a \in \mathcal{A}}\, \widehat{\mu}_{k-1}^\tau(a) + \beta\sqrt{\frac{\log(k-1)}{N_{k-1}^\tau(a)}}, \\ \forall N \leq k \leq K-1 \text{and} U_k < \epsilon_{ucb} & A_k = Y_k, \end{cases}$$

where U_k is a random variable drawn uniformly from $[0, 1]$ and Y_k is a random action sampled uniformly form $\mathcal{A} = \{0, \ldots, N-1\}$.

Full Ensemble Model. We additionally evaluated another algorithm, called `Full Ensemble`. The `Full Ensemble` is a combination of `UTE` and `B-DQN`, which means both action-value functions and option-value functions are estimated by an ensemble method. Figure 7.5 demonstrates that the `Full`

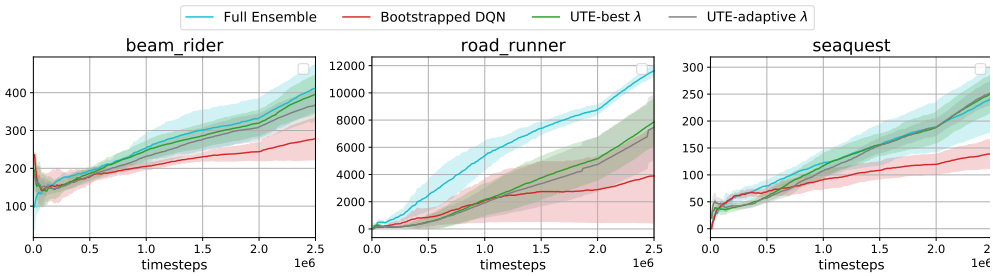


Figure 7.5 Learning curves of `Full Ensemble`, `B-DQN`, `UTE` with the best uncertainty parameter, and `UTE` with adaptive uncertainty parameter over 7 random seeds.

`Ensemble` performs similar to or better than others. In an environment where rewards are relatively sparse such as *Road Runner*, `Full Ensemble` notably outperforms other agents. We did not optimize the uncertainty parameter for `Full Ensemble` so that there is room for further improvements. The results imply that our method can apply to any base algorithm smoothly.

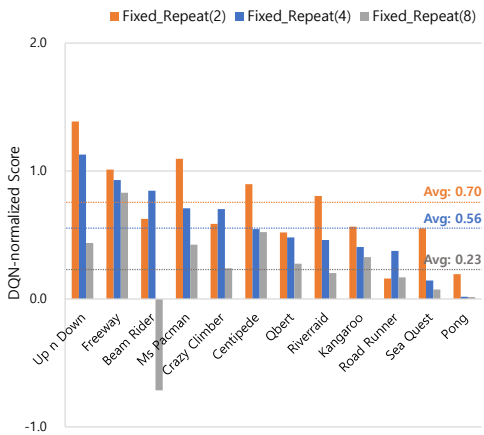
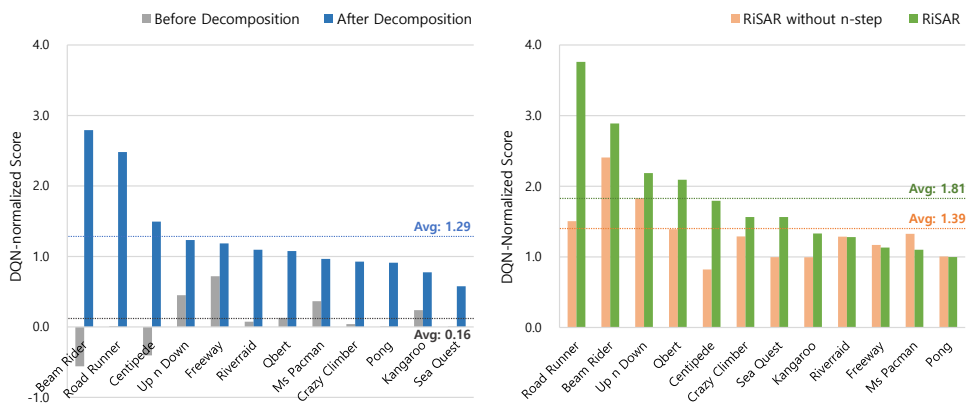


Figure 7.6 DQN-normalized score for Fixed Repeat with varying fixed j. (5 random seeds)

Different j for Fixed Repeat. Figure 7.6 shows the performance of Fixed Repeat agents with varying fixed extension length j. The fixed j affects the granularity of control. The negative score indicates that the performance is worse than a random policy. The result describes that naively repeating the chosen action could degrade the performance in Atari environments. And this tends to get worse as j is increased.

Ablation 1: Decomposition. Our method formulates joint optimization of the action and the extension length as a two-level optimization problem. The action is selected based on Q^{π_ω} and then the extension length is selected based on $\tilde{Q}^{\pi_\omega}$ sequentially. Left of Figure 7.7a shows the effect of the decomposition.



(a) Decomposition Effect **(b)** Multi-step Target Effect for Q^{π_ω}

Figure 7.7 Ablation studies for decomposition effect (left) and n-step learning effect for action-value function Q^{π_ω}. The negative score indicates that the policy is worse than a random policy. (5 random seeds)

Without decomposition, the size of search space is $|\mathcal{A}| \times |J|$, which leads to a catastrophic performance. In some environments such as *Beam Rider* and *Road Runner*, the DQN-normalized scores are negative, which means the agent is worse than a random policy. Overall, We can see that decomposition of action and extension length selection improves the performance significantly.

Ablation 2: Multi-step Target. Right of Figure 7.7b describes the effect of using an n-step target for Q^{π_ω}. We compare `UTE` with n-step Q-learning to the one without it. This result illustrates that applying n-step learning is beneficial in most games, which shows a 30.2% improvement (from 1.39 to 1.81) after it has been applied. Especially in games with relatively sparse rewards such as *Road Runner* and *Centipede*, it dramatically enhanced the performance. This is because rewards can be propagated faster using n-step returns.

	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 7.7 Average rewards for ϵz-`Greedy` varying values for hyperparameter μ in the Atari 2600 environments. (7 random seeds)

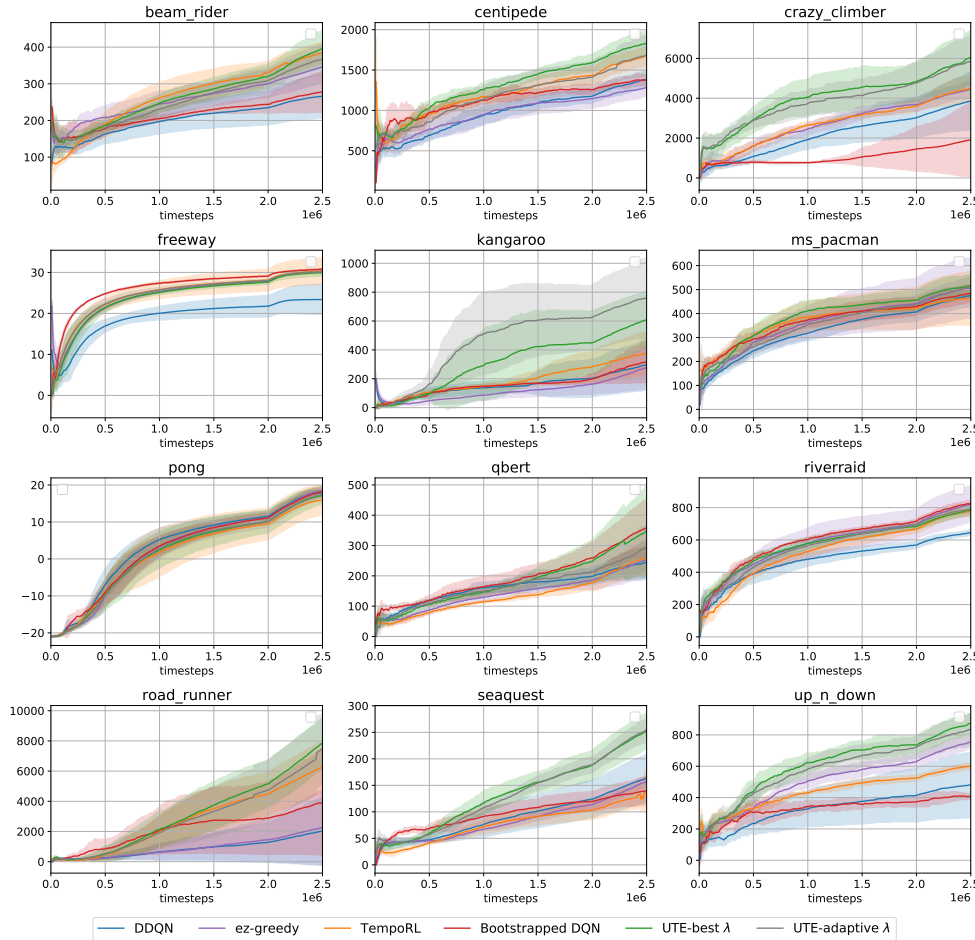


Figure 7.8 Per-game Atari learning curves for ϵz-`Greedy` with the best μ for zeta distribution, `UTE` with the best uncertainty parameter, `UTE` with adaptive uncertainty parameter and the other baselines.(7 random seeds)

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

Table 7.8 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 7.9 Average rewards and standard deviations (numbers in bracket) over the last 100,000 time steps over Atari environments.

Bibliography

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

Bacon, Pierre-Luc, Jean Harb, and Doina Precup, 2017: The option-critic architecture. *Proceedings of the AAAI Conference on Artificial Intelligence*. Volume 31. 1.

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

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

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

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

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

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

Da Silva, Felipe Leno, Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor, 2020: Uncertainty-aware action advising for deep reinforcement learning agents. *Proceedings of the AAAI conference on artificial intelligence*. Volume 34. 04, 5792–5799.

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

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

Efron, Bradley, 1982: *The jackknife, the bootstrap and other resampling plans*. SIAM.

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

Garivier, Aurélien, and Eric Moulines, 2008: On upper-confidence bound policies for non-stationary bandit problems. *arXiv preprint arXiv:0805.3415*.

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

Kalweit, Gabriel, and Joschka Boedecker, 2017: Uncertainty-driven imagination for continuous deep reinforcement learning. *Conference on Robot Learning*. PMLR, 195–206.

Kumar, Aviral, Aurick Zhou, George Tucker, and Sergey Levine, 2020: Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, **33**, 1179–1191.

Lakshminarayanan, Aravind, Sahil Sharma, and Balaraman Ravindran, 2017: Dynamic action repetition for deep reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*. Volume 31. 1.

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

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

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

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

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

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

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

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

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

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

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

Peng, Jing, and Ronald J Williams, 1994: Incremental multi-step q-learning. *Machine Learning Proceedings 1994*. Elsevier, 226–232.

Precup, Doina, 2000: *Temporal abstraction in reinforcement learning*. University of Massachusetts Amherst.

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

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

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

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

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

Sutton, Richard S, Doina Precup, and Satinder Singh, 1999: Between mdps and semi-mdps: a framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, **112**, 181–211.

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

Touati, Ahmed, Harsh Satija, Joshua Romoff, Joelle Pineau, and Pascal Vincent, 2020: Randomized value functions via multiplicative normalizing flows. *Uncertainty in Artificial Intelligence*. PMLR, 422–432.

Van Hasselt, Hado, Arthur Guez, and David Silver, 2016: Deep reinforcement learning with double q-learning. *Proceedings of the AAAI conference on artificial intelligence*. Volume 30. 1.

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

Watkins, Christopher John Cornish Hellaby, 1989: Learning from delayed rewards.

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

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

초 록

강화 학습에서, 행동의 추상화는 정책의 학습과정을 간소화하는 일반적인 접근 방식입니다. 최근, 행동의 추상화를 구현하는 방법론으로 단순히 행동을 일정 기간 동안 반복하는 것이 연구되고 있습니다. 그러나 기존의 행동 반복 연구들의 주요 단점은 차선의 행동을 불필요하게 많이 반복하여 성능이 저하될 수 있다는 문제점이 있습니다. 이러한 경우, 행동의 반복으로 탐색에 이점을 가지는 것보다 그로인한 성능 저하가 더 클 수 있습니다. 따라서, 앙상블 기법을 활용하여 불확실성을 측정하고, 그 불확실성을 고려한 행동 연장 알고리듬(Uncertainty-aware Temporal Extension, UTE)을 고안하였습니다 우리의 알고리듬은 불확실성을 제어하여 더 적극적인 탐색을 유도하거나, 불확실성을 회피하는 정책을 유도할 수 있습니다. 우리는 그리드 월드와 아타리 2600 환경을 비롯한 다양한 환경에서 성능을 평가하였고, 기존의 방법론들보다 우수한 성능을 보임을 확인하였습니다.

주요어: 강화학습, 행동 추상화, 행동 반복, 불확실성, 탐색

학 번: 2021-27322

감사의 글

한국어로 감사합니다!