META-Learning Eligibility Traces for More Sample Efficient Temporal Difference Learning

Mingde Zhao, School of Computer Science

McGill University, Montréal

Apr, 2020

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of

Master of Computer Science

Abstract

Temporal-Difference (TD) learning is a standard and very successful reinforcement learning approach, at the core of both algorithms that learn the value of a given policy, as well as algorithms which learn how to improve policies. TD-learning with eligibility traces provides a way to do temporal credit assignment, *i.e.* decide which portion of a reward should be assigned to predecessor states that occurred at different previous times, controlled by a parameter λ. However, tuning this parameter can be time-consuming, and not tuning it can lead to inefficient learning. To improve the sample efficiency of TD-learning, we propose a meta-learning method for adjusting the eligibility trace parameter, in a state-dependent manner. The adaptation is achieved with the help of auxiliary learners that learn distributional information about the update targets online, incurring roughly the same computational complexity per step as the usual value learner. Our approach can be used both in on-policy and off-policy learning. We prove that, under some assumptions, the proposed method improves the overall quality of the update targets, by minimizing the overall target error. This method can be viewed as a plugin which can also be used to assist prediction with function approximation by meta-learning feature (observation)-based λ online, or even in the control case to assist policy improvement. Our empirical evaluation demonstrates significant performance improvements, as well as improved robustness of the proposed algorithm to learning rate variation.

Abrégé

L'apprentissage par différence temporelle (TD) est une approche d'apprentissage par renforcement standard et très réussie, au cœur des deux algorithmes qui apprennent la valeur d'une politique donnée, ainsi que des algorithmes qui apprennent à améliorer les politiques. L'apprentissage TD avec des traces d'éligibilité fournit un moyen de faire une attribution de crédit temporelle, c'est-à-dire décide quelle portion d'une récompense doit être affectée aux états prédécesseurs qui se sont produits à différents moments précédents, contrôlé par un paramètre λ. Cependant, le réglage de ce paramètre peut prendre du temps et ne pas le régler peut conduire à un apprentissage inefficace. Pour améliorer l'efficacité de l'échantillon d'apprentissage TD, nous proposons une méthode de méta-apprentissage pour ajuster le paramètre de trace d'éligibilité, d'une manière dépendante de l'état. L'adaptation est réalisée avec l'aide d'apprenants auxiliaires qui apprennent en ligne les informations de distribution sur les cibles de mise à jour, entraînant à peu près la même complexité de calcul par étape que l'apprenant de valeur habituelle. Notre approche peut être utilisée à la fois dans l'apprentissage sur les politiques et hors politique. Nous prouvons que, sous certaines hypothèses, la méthode proposée améliore la qualité globale des cibles de mise à jour, en minimisant l'erreur cible globale. Cette méthode peut être considérée comme un plugin qui peut également être utilisé pour aider à la prédiction avec l'approximation des fonctions par la fonction de méta-apprentissage (observation) basée sur λ en ligne, ou même dans le cas de contrôle pour aider à l'amélioration des politiques. Notre évaluation empirique démontre des améliorations significatives des

performances, ainsi qu'une meilleure robustesse de l'algorithme proposé à la variation du taux d'apprentissage.

Acknowledgements

I want to thank Prof. Doina Precup and Prof. Xiao-Wen Chang for funding and supervising me during my master's studies. I also want to thank my colleagues and friends for supporting my studies and my life, especially Sitao Luan and Ian Porada, who are not only good collaborators but also cheerful companions. I would also like to express my sincere gratitude to the people that shared their ideas with me and inspired me to go beyond!

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Temporal-difference learning is an important approach which enables an agent interacting with its environment to learn how to assign credit to different states it encounters and actions it takes. Eligibility trace-based policy evaluation (prediction) methods, *e.g.*, TD(λ), use geometric sequences, controlled by a "trace-decay" parameter λ, to solve the temporal credit assignment problem. Eligibility traces weight multi-step returns and assemble compound update targets. The sample complexity (speed and accuracy of convergence given the number of samples) is in practice sensitive to the choice of λ.

To address this, in this thesis, we propose meta-learning as an approach for adapting the learning *state-based* λs. First, we propose the methodology of improving sample efficiency by improving the quality of the update targets during temporal difference learning. Then, we derive the method of achieving improvement on the overall update targets for each state. Finally, we propose an approximate way to implement the method in an online learning setting, with the help of auxiliary learners and trust region-style updates.

The thesis is structured as followed: In Chapter 2, we introduce the fundamentals of reinforcement learning that are *directly related* to this thesis. The content is fully re-written for the state-based decay and discount settings that we consider in this thesis. The content of the thesis is topologically sorted so that all mentioned methods and settings could be found in this chapter. Then, in Chapter 3, a comprehensive review of the relevant

literature, which focuses on the adaptation of trace-based temporal difference learning, is provided. Chapter 4 and 5 contain our main contribution: a meta-learning approach is proposed, discussed and then validated through experiments. Chapter 6 concludes the thesis and discusses avenues for future work.

Chapter 2

Basics of Reinforcement Learning

A learning paradigm of learning by trial and error. Interacting with an environment to map situations to actions in a way that some notion of cumulative reward is maximized.

2.1 What is Reinforcement Learning?

Learning from interaction is one of the essential and fundamental ideas of intelligence and learning theories [20]. Reinforcement Learning (RL) was introduced by A. Harry Klopf as a computational approach that focused on learning by interacting with the environment without explicit supervision.

In the RL setting, there is an *agent* (the algorithm or the method), an environment (essentially a task that we are facing, a probabilistic system). The environment is an ensemble of a reward function (a scalar feedback for the decisions) and state dynamics (transition probabilities of states by actions). According to the state, the agent computes a series of action to be taken and in this way interacts with the environment.

From a "dataset"-label perspective, we could say that in RL problems, the data samples are dynamically collected by the agents' decisions series. This also means the quality of the "dataset" is also determined by the quality of the decisioning processes of the agents. Therefore, RL is, like many problems, a exploration-exploitation tradeoff prob-

lem with no static datasets and thus it stands out from the classical machine learning paradigms. The learner is not told which actions to take but instead must discover which actions yield the most reward by trying them.

Another distinct feature of RL is that RL agents learn what to do - how to map situations to actions - all by the interactions after its deployment in the environment, *i.e.* it does not need domain specific knowledge.

2.2 Basics

Now, we formally identify the main subelements of a RL system: a policy, a reward signal, a value function and optionally a model of the environment.

A *policy* defines the learning agent's way of behaving for a given state at a given time. Roughly speaking, a policy is a mapping from perceived states of the environment to actions to be taken when in those states.

A *reward* signal defines the goal of a RL problem. On each timestep, the environment sends to the RL agent a scalar feedback called the *reward*. The agent's sole objective is to maximize the total reward it receives over the long run. The reward signal thus defines what are the good and bad events for the agent.

A *value function* specifies what is good in the long run, whereas the reward signal indicates what is good in an immediate sense. Roughly speaking, the value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state. Whereas rewards determine the immediate, intrinsic desirability of environmental states, values indicate the long-term desirability of states after taking into account the states that are likely to follow and the rewards available in those states. Rewards are in a sense primary, whereas values, as predictions of rewards, are secondary. Without rewards there could be no values, and the only purpose of estimating values is to achieve more reward.

The fourth and final element of some reinforcement learning systems is a model of the environment. This is something that mimics the behavior of the environment, or more generally, that allows inferences to be made about how the environment will behave. For example, given a state and action, the model might predict the resultant next state and next reward. Models are used for planning, by which we mean any way of deciding on a course of action by considering possible future situations before they are actually experienced. Methods for solving reinforcement learning problems that use models and planning are called model-based methods, as opposed to simpler model-free methods that are explicitly trial-and-error learners—viewed as almost the opposite of planning.

2.3   Scope of Reinforcement Learning

RL relies heavily on the concept of *state* — as input to the policy and value function, and as both input to and output from the model. We can think of the state as a signal conveying to the agent some sense of "how the environment is" at a particular time. The formal definition of state as we use it here is given by the framework of Markov decision processes.

2.3.1   Markov Decision Processes

Markov Decision Processes (MDPs), a mathematically idealized form of the RL problem for which precise theoretical statements can be made, are a classical formalization of sequential decision making. In MDPs, actions influence not just immediate rewards, which are essentially some feedback from the environment, but also subsequent situations, or states, and through the future rewards. Thus MDPs involve delayed reward and the need to tradeoff immediate and delayed reward.

The learner and decision maker is called the *agent*. The thing it interacts with, comprising everything outside the agent, is called the *environment*. They interact continually, the agent selecting actions and the environment responding to these actions and present-

ing new situations to the agent. The environment also gives *rewards*, special numerical values which the agent seeks to maximize over time through its choice of actions.

As illustrated in Figure 2.1, in an MDP, the agent and the environment interact at each of a sequence of discrete timesteps $t \in \{0, 1, 2, \dots\}$[1]. At each timestep t, the agent receives some representation of the environment's state, $S_t \in \mathcal{S}$ and on that basis selects an action, $A_t \in \mathcal{A}$. One timestep later, in part as a consequence of its action, the agent receives a numerical reward, $R_{t+1} \in \mathcal{R} \in \mathbb{R}$ and finds itself in a new state S_{t+1}. In a finite MDP, the state set $\mathcal{S}$, the action set $\mathcal{A}$ and the reward set $\mathcal{R}$ are all finite. In this case, the random variables R_t and S_t have well defined discrete probability distributions that only depend on the preceding state and action. That is, for particular values of these random variables, $s' \in \mathcal{S}$ and $r \in \mathcal{R}$, there is a probability of those values occurring at timestep t, given particular preceding state and action:

$$p(s', r | s, a) \equiv \mathbb{P}\{S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a\}, \forall s, s' \in \mathcal{S}, \forall r \in \mathcal{R}, \forall a \in \mathcal{A}$$

The function $p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ defines the *dynamics* of the MDP, and is often recognized as the *transition probability function* or simply *transition function*.

The MDP and the agent together thereby give rise to a *trajectory* like:

$$S_0, A_0, R_1, S_1, A_1, R_2, \dots$$

Note that we have used the uppercase letters to denote the random variables since we have not yet observed the states, the rewards or the actions. Yet if we have already, we would use lowercase to denote their specific instantiation. For example, at timestep t, the agent took action a_t based on state s_t and transitioned to the state s_{t+1} while receiving the reward r_{t+1}.

The states that the agent start from in a finite MDP can be described using a probability distribution.

---

[1]The ideas of the discrete time case can be extended to the continuous-time case, *e.g.* [3,8].

Definition 1 (Initial State Distribution). *For a finite Markov decision process, the distribution* $d_0 : \mathcal{S} \to [0, 1]$ *of the first state* S_0, *from which the agent-environment interactions start, is called the starting state distribution or initial state distribution.*

For an RL agent, the distribution d_0 is part of the unknown environment and can be only learnt through interactions.

Markov Property

In an MDP, the probabilities given by p completely characterize the environment's dynamics. That is, the probability of each possible value for S_t and R_t depends only on the immediately preceding state and action S_{t-1} and A_{t-1} not at all on earlier states and actions. This is best viewed a restriction not on the decision process but on the state, which means the state must include information about all aspects of the past agent-environment interaction that make a difference for the future.

The 4-argument deterministic transition function is actually the most general form of a transition function defined in a MDP. There are also alternative forms of the transition function, which rely on either additional assumptions of the environment or exist as marginalized expectations.

Marginalizing over rewards yields the 3-argument state-transition probability function:

$$p(s'|s, a) \equiv \mathbb{P}\{S_t = s'|S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r|s, a)$$

where p is overloaded. However, if we assume that the state transition and the reward are jointly determined, *i.e.* a fixed transition from one state to another always generates the same reward, then the 4-argument transition function collapse into the 3-argument version $p : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to \times[0, 1]$.

Many other useful expected statistics can be derived from the general 4-argument p by marginalizing. These include:
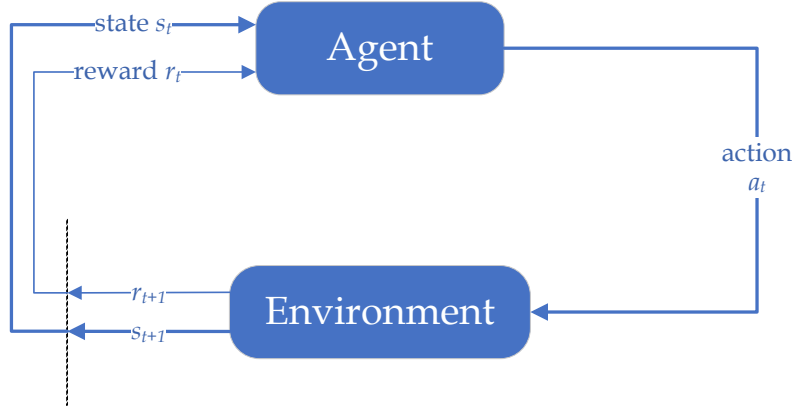
Figure 2.1: Agent-Environment Interaction in a Markov Decision Process

Expected rewards for state-action pairs as a 2-argument function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$:

$$r(s,a) \equiv \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s',r|s,a)$$

Since the only way that the agent could interact with the environment is through the action, there is no way for the agent to optimize the transition and reward by any other means, this 2-argument expected reward function should be an appropriate choice when the agent tries to model the reward function for decisioning, through agent-environment interactions.

Expected rewards for state-action-next-state triples as a 3-argument function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$:

$$r(s,a,s') \equiv \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s',r|s,a)}{p(s'|s,a)}$$

where $p(s'|s,a)$ is the 3-argument transition function we derived earlier. This function can be estimated to predict the reward incurred by some certain transition, which is often used in model-based RL.

Implicit Pause-able Environment Assumption

MDP implicitly assumes that the environment only changes when the agent is taking actions, whereas this assumption seems inappropriate in many decision problems and in the cases where the decisioning process takes in-negligible time, *e.g.* when the agent is planning with a complex model.

Very recently, the notion of realtime reinforcement learning has been introduced in [15] to address such problem.

2.3.2 Returns & Episodes

Undiscounted Episodic Setting

In general, RL seeks to maximize the expected return G_t, which, in the simplest case, is defined as sum of the rewards:

$$G_t \equiv R_{t+1} + R_{t+2} + \cdots + R_T \tag{2.1}$$

where T is a final timestep. The notion of *episodes* is naturally formed as the interaction sequence from the starting timestep 0 until the terminal time T, a random variable that normally varies, for example when playing a game repeatedly.

Each episode ends in a *terminal state*. In this setting, the episodes are assumed to be finite-length, *i.e.* T is finite, and independent with each other, *i.e.* one episode does not affect the environment dynamics of the next.

In episodic tasks, it is sometimes necessary to distinguish the set of all nonterminal states, denoted $\mathcal{S}$ from the set of all states plus the terminal states $\mathcal{S}^+$.

Discounted Episodic Setting

In many cases however, the agent-environment interaction does not break naturally into identifiable episodes, but goes on continually without limit. The previous formulation of

return is problematic in these *continuing tasks* because not only the final step would be $T = \infty$, as well as that the return, which we seek to optimize, goes easily to infinite.

To fix this, an additional concept of *discounting* is needed. According to this approach, the agent tries to select actions so that the sum of the discounted rewards it receives over the future is maximized. In particular, it chooses to maximize the *expected discounted return*:

$$G_t \equiv R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.2}$$

where $\gamma \in [0, 1]$ is the *discount parameter*, also recognized as the *discount rate*. Note that here the discount parameter is constant throughout the states and the episodes. While other more complicated discount settings are also possible. For example, in this thesis, we will adopt the *per-state discount function*, which can be interpreted as discounting the future rewards by some degree after **entering** some state:

$$G_t \equiv R_{t+1} + \gamma_{t+1} R_{t+2} + \gamma_{t+1}\gamma_{t+2} R_{t+3} + \cdots = \sum_{k=0}^{\infty} R_{t+k+1} \cdot \prod_{m=1}^{k} \gamma_{t+m} \tag{2.3}$$

where $\gamma_{t+1} \equiv \gamma(S_{t+1})$, $\gamma : \mathcal{S} \to [0, 1]$.

The discount rate determines the present value of the future rewards: a reward received k timesteps in the future is worth only $\prod_{m=1}^{k-1} \gamma_{t+m}$ times what it would be worth if it were received immediately.

If $\gamma : \mathcal{S} \to [0, 1)$, the infinite sums in 2.3 will have finite values as long as the reward sequence $\{R_k\}$ is bounded.

Returns at successive timesteps are recursively related:

$$\begin{aligned} G_t &\equiv R_{t+1} + \gamma_{t+1} R_{t+2} + \gamma_{t+1}\gamma_{t+2} R_{t+3} + \cdots \\ &= R_{t+1} + \gamma_{t+1} \left(R_{t+2} + \gamma_{t+2} R_{t+3} + \cdots \right) \\ &= R_{t+1} + \gamma_{t+1} G_{t+1} \end{aligned} \tag{2.4}$$

This relation is generalizable for all timesteps $t < T$.

The fact that the discount parameters can be set as 1 gives unified formulation for the episodic tasks as well as the continuing tasks. Ideally, the discount factors should come from the task itself, as its value reflects the goal. However, for complicated tasks with Deep Reinforcement Learning (DRL), which is essentially using artificial neural networks for reinforcement learning, it is generally observed that lowering the discount factor yields significantly more stable performance rather than using $\gamma = 1$, even if the goal tells that there should be no discounting [13]. These blur the line how we should see the discount factor, which classically should be seen as some kind of built-in characteristics of the environment yet now a parameter that could be set or learnt for some purposes.

In the following parts of this thesis, we will focus on the episodic setting with state-based discount functions.

2.3.3 Policies & Value Functions

A *policy* is a function used by an agent to decide what to do given the state it is in. Formally,

Definition 2 (policy). *In an MDP, a policy π is a mapping from states to probabilities of selecting each possible action, i.e. $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$.*

If the agent is following policy π at timestep t, then $\pi(a|s)$ is the probability that $A_t = a$ if $S_t = s$. Note that it is also quite common to define state-based action sets, however we will stick loyal to the setting of [20] for this thesis. The policies in RL is essentially stationary decision rules defined for more general Markov chains [14], where "stationary" means that the decision rules are consistent for every possible states.

The *value function* of a state s under a policy π, denoted $v_\pi(s)$, is the expected (discounted) return when starting in s and following π thereafter. Formally,

Definition 3 (state-value function). *In an MDP, the state-value function for policy π or simply value function $v_\pi(s)$, given discount function $\gamma(s)$ and policy $\pi(s)$, is defined as*

$$v_\pi(s) \equiv \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} R_{t+k+1} \cdot \prod_{m=1}^{k} \gamma_{t+m} | S_t = s \right]$$

where $\mathbb{E}_\pi$ denotes the expected value of random variable given that the agent follows policy π and the values of the terminal states are defined as 0.

There is also the state-action value function, which is more useful when searching for policies.

Definition 4 (state-action-value function). *In an MDP, the state-action-value for policy π $q_\pi(s)$, given discount function γ and policy $\pi(s)$, is defined as the expected return starting from s, taking the action a and thereafter following π:*

$$q_\pi(s, a) \equiv \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} R_{t+k+1} \cdot \prod_{m=1}^{k} \gamma_{t+m} | S_t = s, A_t = a \right]$$

One of the key subroutines of RL is to estimate v_π or q_π from experience, as V_π or Q_π, which is often recognized as *policy evaluation* or *prediction*.

With the recursive relationship derived in 2.4, we can obtain the following equation for the state-value function v_π, given a policy π:

$$
\begin{aligned}
v_\pi(s) &\equiv \mathbb{E}_\pi[G_t | S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma_{t+1} G_{t+1}] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma(s') \cdot \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s'] \right] \qquad (2.5) \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma(s') \cdot v_\pi(s') \right]
\end{aligned}
$$

where $\gamma_{t+1} \equiv \gamma(S_{t+1})$ is a conventional abbreviation, which will be frequently used hereafter. The equation is essentially one basic form of the *Bellman equation* for v_π. Serving

as the core equation of RL, it expresses the relationship between the value of one state and its successor states.

Optimal Policies & Optimal Value Functions

Value functions define a partial ordering over policies in an MDP with certain discounting function γ.

Definition 5 (partial order of policies). *In an MDP with certain discounting function γ, a policy π is defined to be better than or equal to a policy π' if its expected return is greater than or equal to that of π' for all states, i.e. $\pi \geq \pi'$ **iff** $\forall s \in \mathcal{S}, v_\pi(s) \geq v_{\pi'}(s)$.*

There always exists at least one policy that is better than or equal to all other policies, which is identified as the *optimal policy π_**. More specifically,

Fact 1 (Existence of Optimal Deterministic Markovian Policies). *Given a finite MDP, there always exist some deterministic policy π_* that achieves the optimal return, which is independent of the transition history, i.e. Markovian.*

This fact is the reason why we could have confidently focus on the world of Markovian policies, since an optimal policy always exist inside. This also justifies the value-based methods that uses ϵ-greedy as policies since deterministic policies can achieve optimal return. There can be more than one optimal policies but we denote all of them by π_*. The optimal policies share the same state-value function, which is called the *optimal state-value function v_**, which is defined as:

$$v_*(s) \equiv \max_\pi v_\pi(s), \forall s \in \mathcal{S}$$

where the $\max$ operator is defined upon the policy partial orders.

Optimal policies also share the same optimal action-value function q_*, which is defined as

$$q_*(s, a) \equiv \max_\pi q_\pi(s, a), \forall s \in \mathcal{S}, \forall a \in \mathcal{A}$$

The function gives the expected return for taking action a in state s and thereafter following an optimal policy. Thus,

$$q_*(s, a) = \mathbb{E}\left[R_{t+1} + \gamma(S_{t+1}) \cdot v_*(S_{t+1})|S_t = s, A_t = a\right]$$

There are also Bellman equations for optimal policies, which are recognized as *Bellman optimality equations*.

The methods for finding better policies, are recognized as *policy improvement* methods. A representative family of such methods, which are called policy gradient methods, will be introduced in Section 2.9.

2.4   Dynamic Programming

Dynamic Programming (DP) refers to a family of algorithms that can be used for solving the value function given a policy, or finding better, even the optimal policies given a perfect model of the environment in the form of an MDP.

Though guaranteed with optimality, DP algorithms are limited for practical use for the their need of a perfect environment model as well as their expensive computational cost. However, they are still widely used to calculate the ground truth values of relatively small environments for the analyses of new RL methods.

2.4.1   DP for Policy Evaluation

First, we consider the method of computing the state-value function v_π for an arbitrary policy π, which is called *policy evaluation* or *prediction*.

Fact 2 (Existence & Uniqueness of State-Value Function)**.** *The existence and the uniqueness of state-value function v_π are guaranteed as long as either $\gamma(\cdot) < 1$ or termination will be reached from any state following π.*

From the Bellman equation (2.5), we have

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \left[r + \gamma(s') \cdot v_\pi(s')\right]$$

Suppose for a specific MDP, we have a fixed indexing method for all the possible states. When the environment dynamics (4-argument p) is known, the Bellman equation gives a system of $|\mathcal{S}|$ linear equations with $|\mathcal{S}|$ unknowns, *i.e.* solvable. It is desirable to transform it to matrix form and then compute the true values. After some algebraic manipulation, we end up in the following system.

$$\boldsymbol{v}_\pi = \boldsymbol{r}_\pi + P_\pi \boldsymbol{v}_\pi \Gamma$$

where $\boldsymbol{r}_\pi$ is a $|\mathcal{S}| \times 1$ vector in which $\boldsymbol{r}_\pi[i] = \sum_a \pi(a|s_i) \sum_{s_j,r} r \cdot p(s_j,r|s_i,a)$, P is a $|\mathcal{S}| \times |\mathcal{S}|$ matrix in which $P_\pi[i,j] = \sum_a \pi(a|s_i) p(s_j,r|s_i,a)$ and Γ is a diagonal matrix whose $\Gamma(i,i) \equiv \gamma(s_i)$. With the system, the rest is just to solve it, whose complexity is no lower than $\mathcal{O}(|\mathcal{S}|^3)$. The method is elaborated in Algorithm 1.

Algorithm 1: Policy Evaluation in Matrix Form

Input: $p(s',r|s,a)$ (transition probability function), π (policy to be evaluated), γ (discount function)

Output: $\boldsymbol{v}_\pi$ (the state value vector for policy π, where the components correspond to indexed states in $\mathcal{S}^+$)

Form policy-conditioned transition matrix P_π, where
$P_\pi[i,j] = \sum_a \pi(a|s_i) p(s_j,r|s_i,a)$
Form policy-conditioned reward vector $\boldsymbol{r}_\pi$, where
$\boldsymbol{r}_\pi[i] = \sum_a \pi(a|s_i) \sum_{s_j,r} r \cdot p(s_j,r|s_i,a)$
Form a diagonal matrix Γ, where $\Gamma(i,i) \equiv \gamma(s_i)$
$\boldsymbol{v}_\pi = (I - P_\pi\Gamma)\backslash\boldsymbol{r}_\pi$ // or $\boldsymbol{v}_\pi = (I - P_\pi\Gamma)^{-1}\boldsymbol{r}_\pi$

The $\mathcal{O}(|\mathcal{S}|^3)$ complexity is a nightmare for problems with large state sets. Thus, it is desirable to change this method into methods with lower computational complexity. The tool for this conversion is the matrix splitting methods.

Matrix splitting is a method for converting the problems of solving linear equations into iterative methods with presumably lower computational complexity.

Fact 3 (Matrix Splitting). *To solve linear equation* $A\boldsymbol{x} = \boldsymbol{b}$, *where* A *is non-singular, we can split matrix* A *as* $A = M - N$, *where* M *and* N *are greater or equal to* 0 *element-wise. Then, the formula*

$$\boldsymbol{x}_{t+1} = M^{-1}\boldsymbol{b} + M^{-1}N\boldsymbol{x}_t$$

leads to the solution if the spectral radius $\rho(M^{-1}N)$, *i.e. the largest absolute value of the eigenvalues of* $M^{-1}N$, *is less than* 1 *and the spectral radius is also the convergence rate of the iteration.*

With this, when we set $M = I$ and $N = P_\pi\Gamma$ and $\boldsymbol{b} = \boldsymbol{r}_\pi$, we achieve the *iterative policy evaluation method*. It is simple and powerful, turning Bellman equation into an iterative formula achieves the convergence to the true values. One may also prove the convergence of iterative policy evaluation using Banach's fixed point theorem, by showing that the Bellman operator is a contraction.

Definition 6 (Bellman Operator). *Given an MDP with its dynamics* p, *a policy* π *and discount function* γ, *the Bellman operator* $\mathcal{B}_\pi : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$ *is defined by*

$$(\mathcal{B}_\pi v)(s) \equiv \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma(s')v(s')] \tag{2.6}$$

Or equivalently in matrix form,

$$\mathcal{B}_\pi \boldsymbol{V} \equiv \boldsymbol{r}_\pi + P_\pi\Gamma\boldsymbol{V} \tag{2.7}$$

where $\boldsymbol{r}_\pi$ *is a* $|\mathcal{S}| \times 1$ *vector in which* $\boldsymbol{r}_\pi[i] = \sum_a \pi(a|s_i) \sum_{s_j,r} r \cdot p(s_j,r|s_i,a)$, P *is a* $|\mathcal{S}| \times |\mathcal{S}|$ *matrix in which* $P_\pi[i,j] = \sum_a \pi(a|s_i)p(s_j,r|s_i,a)$ *and* Γ *is a diagonal matrix whose* $\Gamma(i,i) \equiv \gamma(s_i)$.

Definition 7 (Contraction). *Let* $\langle X, d \rangle$ *be a complete metric space. Then a map* $\mathcal{T} : X \to X$ *is called a* ***contraction mapping*** *on* X *if there exists* $q \in [0,1)$ *s.t.*

$$\forall x, y \in X, d(\mathcal{T}(x), \mathcal{T}(y)) \leq q \cdot d(x,y)$$

Theorem 1 (Banach–Caccioppoli). *Let $\langle X, d \rangle$ be a non-empty complete metric space with a contraction mapping $\mathcal{T} : X \to X$, then $\mathcal{T}$ admits a **unique** fixed point $x^* \in X$, i.e. $\mathcal{T}(x^*) = x^*$. Moreover, x^* can be found as follows: start with an arbitrary element $x_0 \in X$ and define a sequence $\{x_n\}$ by $x_n = \mathcal{T}(x_{n-1})$ for $n \geq 1$, then $x_n \to x^*$.*

We can prove that the Bellman operator, which is essentially turning the Bellman equation into an iterative formula, on the *estimated value function* or simply *value estimate* is a contraction. The unique fixed point must be the true value because that is when the Bellman equation holds. The vanilla version is provided in Algorithm 2.

Algorithm 2: Iterative Policy Evaluation (Matrix)

Input: $p(s', r | s, a)$ (transition probability function), π (policy to be evaluated), γ (discount factor), θ (accuracy threshold)

Output: $\boldsymbol{v}_\pi$ (the state value vector for policy π, where the components correspond to indexed states in $\mathcal{S}^+$)

Initialize $\boldsymbol{v}_{|\mathcal{S}| \times 1} = \boldsymbol{0}$

Form policy-conditioned transition matrix P_π, where
$\quad P_\pi[i, j] = \sum_a \pi(a | s_i) p(s_j, r | s_i, a)$

Form policy-conditioned reward vector $\boldsymbol{r}_\pi$, where
$\quad \boldsymbol{r}_\pi[i] = \sum_a \pi(a | s_i) \sum_{s_j, r} r \cdot p(s_j, r | s_i, a)$

$\Delta = \infty$

Form a diagonal matrix Γ, where $\Gamma(i, i) \equiv \gamma(s_i)$

while $\Delta \geq \theta$ **do**
$\quad \boldsymbol{v}' = \boldsymbol{r}_\pi + P_\pi \Gamma \boldsymbol{v}$
$\quad \Delta = \|\boldsymbol{v}' - \boldsymbol{v}\|_\infty$
$\quad \boldsymbol{v} = \boldsymbol{v}'$

2.4.2 DP for State Distribution

Given a policy, the expected frequency of states that the agent meet in the environment, which is recognized as the *state frequency*, or the *on-policy distribution*, can also be computed using DP. Note that, given a fixed policy, this frequency is neither dependent on discount function nor the reward function.

In RL literature, the state on-policy distribution is often assumed to be equal to the "stationary distribution"[2] of the Markov chain induced by the MDP and the policy π. However, this assumption is inappropriate because they are never the same in the episodic setting.

One can use an iterative method to solve the true on-policy distribution, given initial state distribution d_0 and dynamics P_π. The idea is to calculate the averaged number of state visits for agents within their lifetime, as shown in Algorithm 3.

Algorithm 3: Iterative DP for State Frequencies in Matrix Form

---

Input: $p(s', r|s, a)$ (transition probability function), π (policy to be evaluated), θ (accuracy threshold), d_0 (initial distribution of states, represented with an indexed vector whose components are the discrete probabilities)

Output: d_π (expected frequencies of states under policy π, where the components correspond to indexed states in $\mathcal{S}^+$)

Initialize $d_\pi = d_0$

Form policy-conditioned transition matrix P_π, where
   $P_\pi[i, j] = \sum_a \pi(a|s_i) p(s_j, r|s_i, a)$

$d' = d_0$

for $i \in \{1, \ldots, |\mathcal{S}|\}$ **do**

    if s_i *is terminal* **then**

        $d'(i) = 0$ // terminal states can only be visited once

while $\|d'\|_1 \geq \theta$ **do**

    $d' = P_\pi^T d'$

    $d_\pi = d_\pi + d'$

    for $i \in \{1, \ldots, |\mathcal{S}|\}$ **do**

        if s_i *is terminal* **then**

            $d'(i) = 0$

$d_\pi = d_\pi / \|d_\pi\|_1$ //normalize

To achieve a more compact form of this algorithm, we need the following proposition:

Theorem 2. *In an MDP, the expected state frequency of an agent during one lifetime (from the beginning of an episode to the end) is the same as the expected state frequency of an agent that is redeployed to the MDP after termination.*

---

[2]This "stationary distribution" is actually not a stationary distribution, as the terminal states of MDP eliminates the ergodicity. The "stationary distribution" d in the episodic setting means only the solution of $d^T P_\pi = d^T$.

Also, the state-frequency $\boldsymbol{d}_\pi$ satisfies:

$$\boldsymbol{d}_\pi^T \tilde{P}_\pi = \boldsymbol{d}_\pi^T \tag{2.8}$$

where $\tilde{P}_\pi$ is the matrix that replaces the rows corresponding to the terminal states in P_π with $\boldsymbol{d}_0^T$.

Proof. Redeploying a terminated agent back to the MDP is upon the initial state distribution. Thus the portion of terminated agents will follow exactly the one-life state frequencies in each of their following lives. □

With this we have a compact matrix-form algorithm and can spot the distribution mismatch more easily. Having spotted the problem, in this thesis, we stick to the more accurate definition of on-policy distribution.

Note that to use this augmented $\tilde{P}_\pi$ for policy evaluation, we need to replace the core operation $\boldsymbol{V} = \boldsymbol{r}_\pi + P_\pi \Gamma \boldsymbol{V}$ with $\boldsymbol{V} = \boldsymbol{r}_\pi + \tilde{P}_\pi \tilde{\Gamma} \boldsymbol{V}$, where $\tilde{\Gamma}$ satisfies that the discount for terminal states are 0[3], as suggested in [27].

Also, in the linear case for projected Bellman operator, which we will discuss in future sections, the contraction of policy evaluation heavily relies on the assumption that the "stationary distribution" is the on-policy distribution, which is critical for a required lemma[4]. We will try to re-prove it with the correct setting[5]. Please refer to [2] theorem 6.3.1 for more details.

2.4.3 DP for Policy Improvement

DP can also be used to find better policies, which ultimately leads to the optimal policies. Finding better policies requires *policy iteration*, which is essentially alternating policy evaluation and policy improvement. The related details will not be covered since they are too distantly related to this thesis.

---

[3] We think it is more appropriate to call P_π a "value" transition matrix, and $\tilde{P}_\pi$ the "state" transition matrix.

[4] $\|P_\pi \boldsymbol{z}\|_{\boldsymbol{d}} \leq \|\boldsymbol{z}\|_{\boldsymbol{d}}$ if $\boldsymbol{d}^T P_\pi = \boldsymbol{d}^T$

[5] $\|P_\pi \boldsymbol{z}\|_{\boldsymbol{d}} \leq \|\boldsymbol{z}\|_{\boldsymbol{d}}$ if $\boldsymbol{d}^T \tilde{P}_\pi = \boldsymbol{d}^T$

2.5 Methods without Perfect Models

DP can solve the value function perfectly and improve the policy only when the perfect model of the environment is given. When not, other methods are needed. Learning from actual experience is striking because it requires no prior knowledge of the environment's dynamics, yet can still attain optimal behavior.

2.5.1 Monte-Carlo Methods for Episodic Tasks

For policy evaluation, the simplest strategy would be to repeatedly utilize the policy in the environment and observe its average performance. Such strategy leads us to the the the simplest policy evaluation method named *Monte-Carlo* or simply *MC*, whose effectiveness is backed by the law of large numbers.

Monte Carlo methods are ways of solving the RL problem based on averaging sample returns. MC is well-defined for episodic setting.

Algorithm 4: Episodic First-visit Monte-Carlo Prediction

Input: π (policy to be evaluated), γ (discount function), N (maximum number of episodes)
Output: $V(s), \forall s \in \mathcal{S}^+$ (state values for policy π)
Initialize $V(s)$ arbitrarily, $\forall s \in \mathcal{S}$
Returns$(s) = $ an emptylist, $\forall s \in \mathcal{S}$
for $n \in \{1, \ldots, N\}$ **do**
    generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G = 0$
    for $t \in \{T-1, T-2, \ldots, 0\}$ **do**
        $G = \gamma S_{t+1} G + R_{t+1}$
        if $S_t \notin \{S_0, \ldots, S_{t-1}\}$ **then**
            // visit every state only once, prefer the longer sum if many
            Append G to Returns(S_t)

for $s \in \mathcal{S}$ **do**
    $V(s) = $ mean(Returns(s))

The state-action values can also be predicted using a very similar method. Policy improvement can also be achieved using generalized policy improvement. The details are omitted.

2.5.2 Temporal Difference Learning

Temporal Difference (TD) learning is the most central idea and methodology of RL, which is a combination of MC and DP. Like MC, TD methods can learn directly from raw experience without a model of the environment's dynamics. Like DP, TD methods *bootstrap*: they update estimates based in part on other learned estimates, without waiting for a final outcome.

Whereas MC must wait until the end of the episode to determine the increment to $V(S_t)$ (only when G_t is known), TD methods need to wait only until the next timestep. At time $t + 1$ they immediately make a useful update using the observed reward R_{t+1} and the estimate $V(S_{t+1})$. The simplest TD method makes the update

$$V(S_t) = V(S_t) + \alpha \left[R_{t+1} + \gamma(S_{t+1})V(S_{t+1}) - V(S_t) \right] \tag{2.9}$$

immediately on transition to S_{t+1} and receiving R_{t+1}. The update rule 2.9 is called the 1-*step TD update*, where we recognize $R_{t+1} + \gamma(S_{t+1})V(S_{t+1}) - V(S_t)$ as the *(1-step) TD error* and $R_{t+1} + \gamma(S_{t+1})V(S_{t+1})$ as the *update target*. Every 1-step TD update can be understood as: walk towards the update target $R_{t+1} + \gamma(S_{t+1})V(S_{t+1})$ from the current (estimated) value $V(S_t)$ with a step length of $\alpha \left[R_{t+1} + \gamma(S_{t+1})V(S_{t+1}) - V(S_t) \right]$ (decreasing the distance by ratio α). Note that the update target $R_{t+1} + \gamma(S_{t+1})V(S_{t+1})$ is also a random variable. The 1-step update yields the simplest TD method, which is named TD(0) and presented in Algorithm 5.

Because TD(0) bases its update in part on an existing estimate, we say that it is a *bootstrapping* method, like DP.

Algorithm 5: Tabular 1-step TD Policy Evaluation (TD(0))

Input: π (policy to be evaluated), γ (discount function), $\alpha \in (0, 1]$ (learning rate),
       N (maximum number of episodes)
Output: $V(s), \forall s \in \mathcal{S}^+$ (state values for policy π)
Initialize $V(s) = 0, \forall s \in \mathcal{S}$
for $n \in \{1, \ldots, N\}$ **do**
    Initialize S
    while S *is not terminal* **do**
        $A = $ action given by$\pi(\cdot|s)$
        Take action A, observe R, S'
        $V(S_t) = V(S_t) + \alpha\left[R_{t+1} + \gamma(S_{t+1})V(S_{t+1}) - V(S_t)\right]$
        $S = S' \ G = \gamma S_{t+1} G + R_{t+1}$

Fact 4. *Under the episodic setting, given an MDP and a policy π, either discounted or not, TD(0) achieves convergence to v_π asymptotically.*

2.5.3   Multi-Step TD

Besides the 1-step TD target, the update target could be many things as long as the convergence to the true value can be guaranteed. In this subsection, we introduce n-step TD methods that generalize both TD and MC methods so that one can shift from one to the other smoothly as needed to meet the demands of a particular task. n-step methods span a spectrum with MC methods at one end and 1-step TD methods at the other.

Definition 8 (n-step return). *The n-step return is defined as:*

$$G_{t:t+n} = R_{t+1} + \gamma(S_{t+1})R_{t+2} + \cdots + \prod_{k=1}^{n}\gamma(S_{t+k})V_{t+n-1}(S_{t+n}), \forall n \geq 1, \forall 0 \leq t \leq T - n \quad (2.10)$$

The n-step return serves as the update target for an *n-step TD update.*

The n-step return uses the value function V_{t+n-1} to correct for the missing rewards beyond R_{t+n}. The error reduction property of n-step returns lead to its convergence under appropriate technical conditions [20].

We notice that, when n is set to be the timestep difference between the current timestep and the timestep for the end of the episode, the n-step return becomes the MC return. And

---

Algorithm 6: Tabular n-step TD Policy Evaluation (TD(n))

Input: π (policy to be evaluated), γ (discount function), α (learning rate), M
       (maximum number of episodes), n (step parameter)
Output: $V(s), \forall s \in \mathcal{S}^+$ (state values for policy π)
Initialize $V(s) = 0, \forall s \in \mathcal{S}$
for $m \in \{1, \ldots, M\}$ **do**
    Initialize and store non-terminal S_0
    $T = \infty$
    for $t \in \{0, 1, 2, \ldots\}$ **do**
        if $t < T$ **then**
            Take action according to $\pi(\cdot|s)$, observe and store R_{t+1}, S_{t+1}
            if S_{t+1} *is terminal* **then**
                $T = t + 1$
        $\tau = t - n + 1$
        if $\tau \geq 0$ **then**
            $G = \sum_{i=\tau+1}^{min(\tau+n, T)} \left(\prod_{k=1}^{i} \gamma(S_k) \right) \cdot R_i$
            if $\tau + n < T$ **then**
                $G = G + \prod_{k=1}^{n} \gamma(S_{\tau+k}) V(S_{\tau+n})$
            $V(S_\tau) = V(S_\tau) + \alpha[G - V(S_\tau)]$
        if $\tau = T - 1$ **then**
            break

---

when $n = 1$, the method collapses to TD(0). This is to say that n-step returns, as update targets, generalizes the TD and MC and yields all the shades between them.

2.6 Off-Policy Learning Using Importance Sampling

How can an agent estimate the values of one policy when acting upon another? Let us call the policy to learn about the *target policy*, and the policy used to generate behavior the *behavior policy*. In this case we say that learning is from data "off" the *target policy*, and the overall process is termed off-policy learning.

2.6.1 Importance Sampling

Importance sampling is a technique to use a sample of examples from a different distribution to estimate the expectation of some target distribution. It requires the knowledge to explicitly compute the probability of each sample under the two distributions. Let the target distribution be π and the distribution that generated the sample be b, we have the following.

Definition 9 (Importance Sampling Estimator). *Suppose that π and b are probability density (mass) functions that satisfy $b(x) = 0 \implies \pi(x) = 0$, i.e. π is absolutely continuous w.r.t. b, we define the importance sampling estimator $\hat{\mu}_{IS}$ of $\mathbb{E}[f(x)]$ as:*

$$\hat{\mu}_{IS} \equiv \frac{1}{n} \sum_{i=1}^{n} f(x_i)\rho(x_i) \tag{2.11}$$

where $\rho(x_i) \equiv \frac{\pi(x_i)}{b(x_i)}, x_i \sim b$ is recognized as an importance weight function and its value is recognized as an importance sampling ratio.

The importance sampling estimator has some properties that we need to know.

Theorem 3. *Let $\mu \equiv \mathbb{E}_{\pi}[f(x)]$,*

$$\mathbb{E}_b[\hat{\mu}_{IS}] = \mu, Var_b[\mu_{IS}] = \int \frac{(f(x)\pi(x) - \mu b(x))^2}{b(x)} dx$$

The proof is straightforward algebra. The unbiasedness shows that importance sampling ratios can be used to estimate the statistics of data even if they are generated using from different sources. However, the variance will bring trouble, if π and b are different.

2.6.2 Importance Sampling based Off-Policy Learning

Now we plug the theory of importance sampling into RL. Let the target policy be π and the behavior policy be b. Given a starting state S_t, the probability of the subsequent state-

action trajectory, $A_t, S_{t+1}, \ldots, S_T$, occurring under π is:

$$\mathbb{P}\{A_t, S_{t+1}, \ldots, S_T | S_t, A_{t:T-1} \sim \pi\}$$

$$= \pi(A_t|S_t)p(S_{t+1}|S_t, A_t)\pi(A_{t+1}|S_{t+1}) \cdots p(S_T|S_{T-1}, A_{T-1}) \quad = \prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)$$

where p is the 3-argument transition function. The relative probability (importance sampling ratio) of the trajectory under the policies π and b is:

$$\rho_{t:T-1} \equiv \frac{\prod_{k=t}^{T-1} \pi(A_k|S_k)p(S_{k+1}|S_k, A_k)}{\prod_{k=t}^{T-1} b(A_k|S_k)p(S_{k+1}|S_k, A_k)} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$$

The canceling of the terms show that importance sampling ratio of trajectories does not depend on the MDP's dynamics. With this we have

$$\mathbb{E}_b[\rho_{t:T-1}G_t|S_t = s] = v_\pi(s)$$

This means that Monte-Carlo method can learn the target values as long as we have the computational access to the importance sampling ratios. We will not cover the details of off-policy MC.

2.6.3 Per-Decision Importance Sampling

The off-policy MC estimator, the unbiased one with high-variance, of return is:

$$\begin{aligned}
\rho_{t:T-1}G_t &= \rho_{t:T-1}\left(R_{t+1} + \gamma_{t+1}R_{t+2} + \cdots + \prod_{k=t+1}^{T-1}\gamma_k R_T\right) \\
&= \rho_{t:T-1}R_{t+1} + \gamma_{t+1}\rho_{t:T-1}R_{t+2} + \cdots + \prod_{k=t+1}^{T-1}\gamma_k\rho_{t:T-1}R_T
\end{aligned} \quad (2.12)$$

Each sub-term is a product of a random reward and a random importance sampling ratio. For example, the first sub-term is:

$$\rho_{t:T-1}R_{t+1} = \frac{\pi(A_t|S_t)}{b(A_t|S_t)}\frac{\pi(A_{t+1}|S_{t+1})}{b(A_{t+1}|S_{t+1})}\frac{\pi(A_{t+2}|S_{t+2})}{b(A_{t+2}|S_{t+2})}\cdots\frac{\pi(A_{T-1}|S_{T-1})}{b(A_{T-1}|S_{T-1})}R_{t+1}$$

For this term, it is intuitive to see that only $\frac{\pi(A_t|S_t)}{b(A_t|S_t)}$ and R_{t+1} are related, as one can easily show:

$$\mathbb{E}_b\left[\frac{\pi(A_k|S_k)}{b(A_k|S_k)}\right] \equiv \sum_{a\sim b}\frac{\pi(a|S_k)}{b(a|S_k)} = \sum_a b(a|S_k)\cdot\frac{\pi(a|S_k)}{b(a|S_k)} = \sum_a \pi(a|S_k) = 1$$

Thus,

$$\mathbb{E}_b[\rho_{t:T-1}R_{t+1}] = \mathbb{E}_b[\rho_{t:t}R_{t+1}]$$

and also

$$\mathbb{E}_b[\rho_{t:T-1}R_{t+k}] = \mathbb{E}_b[\rho_{t:t+k-1}R_{t+k}]$$

With this we can get another unbiased return estimator, which is recognized as the *per-decision* importance sampling estimator $\tilde{G}_t$ for return:

$$\mathbb{E}_b[\rho_{t:T-1}G_t] = \mathbb{E}_b[\tilde{G}_t]$$

and

$$
\begin{aligned}
\tilde{G}_t &\equiv \rho_{t:t}R_{t+1} + \gamma_{t+1}\rho_{t:t+1}R_{t+2} + \cdots + \prod_{k=t+1}^{T-1}\gamma_k\rho_{t:T-1}R_T \\
&= \rho_t R_{t+1} + \gamma_{t+1}\rho_t\rho_{t+1}R_{t+2} + \cdots + \prod_{k=t+1}^{T-1}\gamma_k\cdot\prod_{j=t}^{T-1}\rho_j\cdot R_T \quad (\rho_j \equiv \frac{\pi(A_j|S_j)}{b(A_j|S_j)}) \quad\quad (2.13) \\
&= \rho_t\left(R_{t+1} + \gamma_{t+1}\rho_{t+1}\left(R_{t+2} + \gamma_{t+2}\rho_{t+2}\left(\cdots\right)\right)\right)
\end{aligned}
$$

Per-decision importance sampling enables off-policy bootstrapping. The change of the algorithm is just to multiply the learning rates of the TD updates by the per-decision importance sampling ratio.

2.7 Function Approximation

The notion of state that we have discussed before are recognized as *tabular*, in a sense that we can list a table for all the states and their corresponding properties. An agent, at a particular time, can only be in exactly one state, and these states do not influence each other. However, this setting is problematic for the cases in which the state space is too large to be discretized as tables, *e.g.* when the state space is continuous. In this case, which is recognized as the *function approximation case*, we have to use the approximate value function, which is not represented as a table but as a parameterized functional form with some corresponding weight vector $\boldsymbol{w}$.

In the tabular case a continuous measure of prediction quality was not necessary because the learned value function could converge to the true value function exactly. Moreover, the learned values at each state were decoupled —an update at one state affected no other. But with function approximation, an update at one state affects many others, and it is not possible to get the values of all states exactly correct. By assumption we have far more states than weights, so making one state's estimate more accurate invariably means making others' less accurate. However, this could also mean that making one state more accurate will make some similar states also more accurate. This is often recognized as the dilemma of generalization: on one hand it introduces the forgetting problem; On the other hand it could significantly accelerate learning for its updates to the similar states.

Since there could be many states, generally the states' importance are weighted using the state frequency distribution d_π. With this we obtain a natural objective function, the state-value error, which is essentially the weighted mean squared value error between the true value and the value estimate.

Definition 10 (State-Value Error). *Given an MDP, for a state s, let its true value under target policy π be $v_\pi(s)$. Given an estimated value $V_\pi(s)$ of the state s, the state value error of the estimate $V_\pi(s)$ is defined as:*

$$J(s) \equiv 1/2 \cdot (V_\pi(s) - v_\pi(s))^2$$

The state value error of a value estimate is its squared distance to the true value. Weighting the state value error by their state frequency d_π yields the following:

Definition 11 (Overall Value Error). *Given an MDP and a particular fixed indexing of its states, let its true state-values under target policy π be $\boldsymbol{v}_\pi$, where each element of the vector corresponds to the true value of an indexed state and an value estimate. Given an estimate $\boldsymbol{V}_\pi$ of all the states, the **overall value error** of the estimate $\boldsymbol{V}_\pi$ is defined as:*

$$J(\boldsymbol{V}_\pi) \equiv 1/2 \cdot \left\| D_\pi^{1/2} \cdot (\boldsymbol{V}_\pi - \boldsymbol{v}_\pi) \right\|_2^2$$

where D_π is the diagonalized state frequencies under π, i.e.

$$D_\pi \equiv diag(d_\pi(s_1), d_\pi(s_2), \cdots, d_\pi(s_{|S|})) \tag{2.14}$$

This criterion can be used with any form of value estimator, either tabular or with function approximators. The weights D_π favor the states that will be met with higher frequency. The overall value error is often used to evaluate the performance of policy evaluation [19]. When a perfect model of the environment MDP is known, the $\boldsymbol{v}_\pi$ and D_π can be exactly solved using DP, as discussed in Section 2.4. Thus, DP-solvable MDPs are the first-choices of testing the policy evaluation algorithms.

An ideal goal in terms of policy evaluation would be to find a global optimum, a weight vector $\boldsymbol{w}^*$ for which $J(\boldsymbol{V}_\pi(\boldsymbol{w}^*)) \leq J(\boldsymbol{V}_\pi(\boldsymbol{w}))$ for all possible $\boldsymbol{w}$. Reaching this goal is sometimes possible for simple function approximators such as linear function approximators, which are to be introduced later, but is rarely possible for complex function approximators, *e.g.* artificial neural networks.

Usually, we will use differentiable value estimate functions $V(s; \boldsymbol{w})$ parameterized by a weight vector $\boldsymbol{w}$ to enable stochastic gradient-descent methods for approaching the update targets.

$\boldsymbol{w}$ will be updated at each of a series of discrete timesteps as before, $t \in \{1, 2, \dots\}$, trying to minimize the state-value error. Stochastic gradient-descent (SGD) methods do this by adjusting the weight vector after each example by a small amount in the direction that would most reduce the error on that example:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{1}{2}\alpha\nabla\left[v_\pi(S_t) - V_\pi(S_t, \boldsymbol{w}_t)\right]^2 = \boldsymbol{w}_t + \alpha\left[v_\pi(S_t) - V_\pi(S_t, \boldsymbol{w}_t)\right]\nabla V_\pi(S_t, \boldsymbol{w}_t) \quad (2.15)$$

where α is the learning rate, a positive step-size parameter.

Gradient descent methods are called "stochastic" when the update is done on only a single example, selected stochastically. Over many steps, the overall effect is to minimize an average performance measure such as the overall value error.

Obviously, we cannot use (2.15) to do update because the true value $v_\pi(S_t)$ is unknown. Thus, we must replace the update target $v_\pi(S_t)$ with an estimate U_t. If U_t is unbiased, *i.e.* $\mathbb{E}[U_t|S_t = s] = v_\pi(S_t), \forall t$, then $\boldsymbol{w}_t$ is guaranteed to converge to a local optimum under the usual SGD conditions with decreasing α. One simplest instance of this kind of method is to use the MC returns as the update targets, which leads to the gradient-MC method.

Such unbiasedness cannot be achieved with TD updates, which are essentially using bootstrapping estimates as targets. Bootstrapping targets or DP target depend on the current value of the value estimate and the parameter $\boldsymbol{w}_t$ for the value estimate. This implies that they are biased and will not produce a true gradient method. It has been proved that bootstrapping methods are not in fact instances of true gradient descent [1], as they take into account the effect of changing the weight $\boldsymbol{w}_t$ on the estimate but ignore

its effect on the target. They are recognized as *semi-gradient* methods because they only take into consideration a part of the gradient.

Although semi-gradient bootstrapping methods do not converge as robustly as gradient methods, they do converge reliably in important cases such as the linear case. Algorithm 7 shows the simplest instance, the semi-gradient TD(0), which uses 1-step target as the update target.

Algorithm 7: Semi-Gradient TD(0) for Policy Evaluation

Input: π (policy to be evaluated), γ (discount function), $\alpha \in (0, 1]$ (learning rate),
      N (maximum number of episodes)
Output: $V(s), \forall s \in \mathcal{S}^+$ (state values for policy π)
Initialize $V(s) = 0, \forall s \in \mathcal{S}$
for $n \in \{1, \ldots, N\}$ **do**
    Initialize S
    while S *is not terminal* **do**
        $A =$ action given by$\pi(\cdot|s)$
        Take action A, observe R, S'
        $\boldsymbol{w} = \boldsymbol{w} + \alpha \left[R + \gamma(S') \cdot V(S'; \boldsymbol{w}) - V(S; \boldsymbol{w}) \right] \nabla_{\boldsymbol{w}} V(S; \boldsymbol{w})$
        $S = S'$

2.7.1 Linear Methods

One of the simplest and most important special cases of function approximation is the *linear function*, where $V(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x}$, is a linear function of the weight vector $\boldsymbol{w}$, and $\boldsymbol{x}$ is some real-valued feature vector corresponding to some state s. The linear case brings some important properties:

First, the gradient of the parameter $\boldsymbol{w}$ has a special form that is independent of the parameter $\boldsymbol{w}$ - the gradient of the approximate value function with respect to $\boldsymbol{w}$ is $\nabla_{\boldsymbol{w}} V(s; \boldsymbol{w}) = \boldsymbol{x}(s)$. This means that a once a gradient of yielded by some feature is calculated, it remains valid, *i.e.* remains a true gradient, forever. This is a very special property that empowers eligibility traces, a fundamental RL method for policy evaluation.

Second, in particular, in the linear case there is only one optimum (or, in degenerate cases, one set of equally good optima), and thus any method that is guaranteed to con-

verge to or near a local optimum is automatically guaranteed to converge to or near the global optimum.

Note that the convergence of linear semi-gradient TD(0) algorithm presented in Algorithm 7 does not follow from general results on SGD but a separate theorem. The weight vector converged to is also not the optimum, but rather a point nearby. The update for linear semi-gradient TD(0) is

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha(R_{t+1} + \gamma(\boldsymbol{x}_{t+1})\boldsymbol{w}_t^T\boldsymbol{x}_{t+1} - \boldsymbol{w}_t^T\boldsymbol{x}_t)\boldsymbol{x}_t$$
$$= \boldsymbol{w}_t + \alpha(R_{t+1}\boldsymbol{x}_t - \boldsymbol{x}_t(\boldsymbol{x}_t - \gamma(\boldsymbol{x}_{t+1})\boldsymbol{x}_{t+1})^T\boldsymbol{w}_t)$$

where $\boldsymbol{x}_t \equiv \boldsymbol{x}(S_t)$.

If we use onehot encoding for tabular states, which is to use a binary vector with length $|\mathcal{S}|$ and mark the corresponding state with 1, we can see that tabular TD(0) is a special case of semi-gradient TD(0).

2.7.2 Tile Coding

Tile coding uses overlapping tilings to generate binary features for multi-dimensional continuous spaces, which are beneficial for generalization.

In tile coding, multiple tilings are used. Each *tiling* is a grid partition of the state space and each element of the partition is called a *tile*. The tilings are put on the state space, each offset by a fraction of a tile width. A simple case with 4 tilings is shown on the right side of Figure 2.2. Every state, such as that indicated by the white spot, falls in exactly one tile in each of the 4 tilings. These 4 tiles correspond to 4 features that become active. Specifically, the feature vector $\boldsymbol{x}(s)$ has one component for each tile in each tiling. In this example there are $4 \times 4 \times 4 = 64$ components, all of which will be 0 except for the 4 corresponding to the tiles that s is within.

There are several advantages of using tile coding for feature construction. 1) Since the overall number of activated tiles are always the same for any state, the learning rate

Tiling 1
Tiling 2
Tiling 3
Tiling 4

Continuous
2D state
space

Point in
state space
to be
represented

Four active
tiles/features
overlap the point
and are used to
represent it
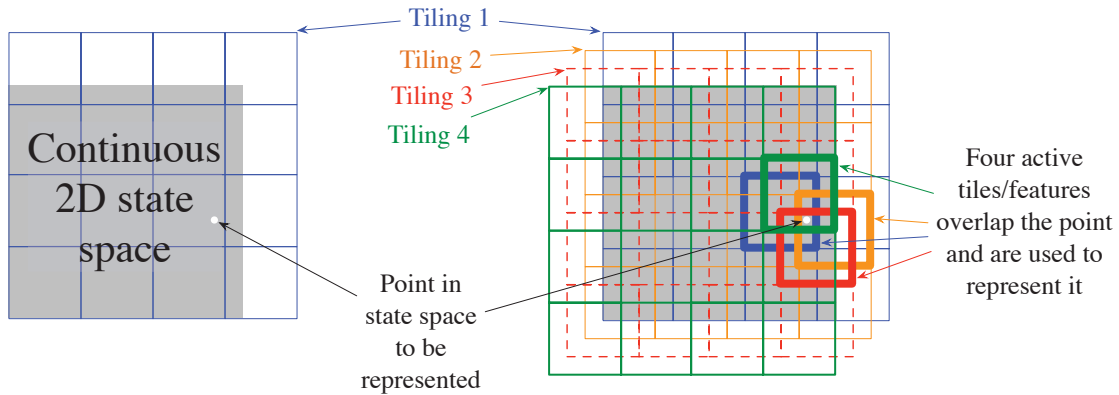
Figure 2.2: Multiple overlapping grid-tilings on a 2D box space with uniform offset in each dimension. Figure from [20].

parameter can be set intuitively and easily; 2) Since the features are always binary, efficient binary-float multiplication can be implemented for calculation; 3) Since there is no constraint on the shape of the tilings as well as the offsets, significant degrees of freedom can be utilized to design effective feature constructors.

Tile coding will be used as one of the feature construction methods in the experimental studies section.

2.7.3 Off-policy Methods with Function Approximation

When learning off-policy with function approximation, new troubles emerge for semi-gradient methods. First, the update targets need to be fixed with importance sampling ratios; Second and most importantly, the state distribution will no longer match the target policy. This means that the cumulative effect of gradient or semi-gradient updates does not optimize the overall value error.

Let us first look into why off-policy learning is much more difficult for the case of function approximation than the tabular case. In the function approximation case, generally the updates for one state affect all the similar states whereas in the tabular case, the updates for one state have no influence on others. This means that, in the off-policy case, the tabular updates do not have to care about the state-frequencies when doing updates

as long as the update targets are fixed using the importance sampling ratios. The blessing that the tabular case updates do not rely on any special distribution for stability has not been passed to the function approximation case. In the function approximation case, the semi-gradient methods that we have introduced before rely on the state-frequencies for updates. This means we either have to "reweight" the updates, *i.e.* to warp the update distribution back to the on-policy distribution using importance sampling methods, or we have to develop true gradient methods that do not rely on any special distribution for stability. In fact, the problem of the coexistence of bootstrapping, off-policy learning and function approximation is so troublesome that it is considered as "the deadly triad".

Here, we focus on the gradient methods.

Gradient Methods for Linear Case

Definition 12 (Bellman Error). *Given an MDP with its dynamics p, a policy π and discount function γ, let the corresponding Bellman operator be $\mathcal{B}_\pi$, the Bellman Error (BE) for a value estimate $V_{\boldsymbol{w}}$ is defined as the norm of the Bellman Error vector, i.e. the expected TD error vector, induced by the state-frequencies d_π.*

$$\overline{BE}(\boldsymbol{w}) \equiv \|\overline{\boldsymbol{\delta}}_{\boldsymbol{w}}\|^2_{d_\pi} \equiv \|D_\pi^{1/2} \cdot \overline{\boldsymbol{\delta}}_{\boldsymbol{w}}\|^2_2 \tag{2.16}$$

where D_π is the diagonalized $\boldsymbol{d}_\pi$ and the BE vector $\overline{\boldsymbol{\delta}}_{\boldsymbol{w}}$ is defined as:

$$\overline{\boldsymbol{\delta}}_{\boldsymbol{w}} \equiv \mathcal{B}_\pi V_{\boldsymbol{w}} - V_{\boldsymbol{w}} \tag{2.17}$$

It has been proved that, unfortunately, true gradient methods optimizing the Bellman error, named *residual methods*, cannot be realized for general RL settings unless the environment transitions are deterministic or if the transitions of the environment can be somehow reversed. Also, geometric analyses in the linear case show that the minimizers of the Bellman error may not be desirable. These lead to the gradient methods seeking to optimize the Mean Squared Projected Bellman Error (MSPBE).

Definition 13 (Projected Bellman Error Vector & Mean Squared Projected Bellman Error).
Given an MDP, target policy π and a parameterized value estimate $\boldsymbol{V_w}$, the Mean Squared Projected Bellman Error (MSPBE) is defined as the norm of the Projected Bellman Error vector (PBE), induced by the state-frequencies d_π.

$$\overline{PBE}(\boldsymbol{w}) \equiv \|\hat{\boldsymbol{\delta}}_{\boldsymbol{w}}\|^2_{d_\pi} \equiv \|D_\pi^{1/2} \cdot \hat{\boldsymbol{\delta}}_{\boldsymbol{w}}\|^2_2 \tag{2.18}$$

where D_π is the diagonalized $\boldsymbol{d}_\pi$ and the PBE vector $\hat{\boldsymbol{\delta}}_{\boldsymbol{w}}$ is defined as:

$$\hat{\boldsymbol{\delta}}_{\boldsymbol{w}} \equiv \Pi \overline{\boldsymbol{\delta}}_{\boldsymbol{w}}$$

where $\overline{\boldsymbol{\delta}}_{\boldsymbol{w}}$ is the Bellman error vector of $\boldsymbol{V_w}$ and Π is the projection operator that takes an arbitrary value function $\boldsymbol{V}'$ to the representable function that is closest in the weighted norm, which is:

$$\Pi \boldsymbol{V}' \equiv \boldsymbol{V_{w'}} \text{ and } \boldsymbol{w}' = \operatorname*{argmin}_{\boldsymbol{w}} \|\boldsymbol{V}' - \boldsymbol{V_w}\|^2_{d_\pi}$$

In the linear case, given a fixed state-to-feature mapping and a fixed policy π, the projection operator is a static linear transformation which does not depend on the parameter $\boldsymbol{w}$. This gives birth to the linear gradient-TD methods which achieve convergence by minimizing MSPBE under some conditions. In this thesis, we propose an assistive method for general function approximation based policy evaluation, but test on these linear methods with the back of the convergence guarantees.

2.8   From λ-Return to Eligibility Traces

For a given MDP, the update targets of different steps, *e.g.* 1-step target, 2-step target, *etc.*, yield different biases and variances, which lead to different qualities of the estimates. Naturally, one would like to combine them in a way *s.t.* we get better estimates, *e.g.* with

lower MSE towards the tru value, and without losing the properties of convergence. The following fact unlocks such possibility.

Fact 5 (Compound Targets & Compound Updates). *Given a MDP and appropriate learning rate, using convex combinations of multi-step returns as update targets for policy evaluation achieves convergence to fixed-points near the true value.*

Often, these targets are recognized as *compound targets*. The updates using compound targets are called *compound updates*. The convergence of compound updates relies on the fact that compound targets are composed of multi-step updates, and each of them has convergence guarantees.

There are potentially many ways to mix the multi-step targets to achieve compound updates. Optimizing on the way of mixing should be beneficial for the sample efficiency of policy evaluation.

One of the most popular way is to mix the multi-step targets with a geometric weight sequence, which yields the famous λ-return:

Definition 14 (λ-return). *Given timestep t and the corresponding state S_t, the λ-return of S_t is defined as a convex combination of the multi-step targets of S_t, with the weights specified using a geometric sequence, controlled by a scalar parameter λ. Specifically, the λ-return G_t^λ combines all n-step returns $G_t^{(n)}$ using weight $(1 - \lambda)\lambda^{n-1}$:*

$$G_t^\lambda \equiv (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$$

where $\lambda \in [0, 1]$.

Methods that use λ-return as update target, given that parameter λ is set to be appropriate, often achieve significantly higher sample efficiency than using only some fixed-step returns as targets, for the fact that λ-return has a better bias-variance tradeoff.

Calculating this naïvely requires knowing all the multi-step targets up until the end of the trajectory. Thus, updates using λ-return directly can only be done offline, which

35

is recognized as the *offline λ-return algorithm*. Offline algorithms can be unsatisfactory for many reasons. Fortunately, there is a way to approximate the updates towards λ-return in an online fashion[6]. The compound update using λ-return can be approximated using the eligibility traces from a backward view, with the help of the eligibility trace vectors:

Proposition 1 (Trace Approximation). *Given an MDP and an infinitely long trajectory under policy π, the updates using λ-return as targets can be approximated with online updates. More specifically, the update rules are:*

$$\boldsymbol{z}_t = \gamma\lambda\boldsymbol{z}_{t-1} + \boldsymbol{\nabla}V(\boldsymbol{x}_t, \boldsymbol{w}_t)$$

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}_t + \alpha[R_{t+1} + \gamma V(\boldsymbol{x}_{t+1}, \boldsymbol{w}_t) - V(\boldsymbol{x}_t, \boldsymbol{w}_t)]\boldsymbol{z}_t$$

where $\boldsymbol{z}$ is the eligibility trace vector which is initialized as $\boldsymbol{0}$ at the beginning of the episode and α is some appropriate learning rate.

Proof.

$G_t^\lambda - V(S_t)$

$= -V(S_t) + (1-\lambda)\lambda^0(R_{t+1} + \gamma^1 V(S_{t+1})) + (1-\lambda)\lambda^1(R_{t+1} + \gamma^1 R_{t+2} + \gamma^2 V(S_{t+2})) + \cdots$

$= -V(S_t) + (1-\lambda)[\lambda^0\gamma^0(\lambda^0 + \lambda^1 + \cdots)R_{t+1} + \lambda^1\gamma^1(\lambda^0 + \lambda^1 + \cdots)R_{t+2} + \cdots + \gamma^1\lambda^0 V(S_{t+1}) + \gamma^2\lambda^1 V(S_{t+}$

$= -V(S_t) + (1-\lambda)[\frac{\lambda^0\gamma^0}{1-\lambda}R_{t+1} + \frac{\lambda^1\gamma^1}{1-\lambda}R_{t+2} + \cdots + \gamma^1\lambda^0 V(S_{t+1}) + \gamma^2\lambda^1 V(S_{t+2}) + \cdots]$

$= (\gamma\lambda)^0(R_{t+1} + \gamma V(S_{t+1}) - \gamma\lambda V(S_{t+1})) + (\gamma\lambda)^1(R_{t+2} + \gamma V(S_{t+2}) - \gamma\lambda V(S_{t+2})) + \cdots$

$= \sum_{k=t}^{\infty} (\gamma\lambda)^{k-t}(R_{k+1} + \gamma V(S_{k+1}) - V(S_k))$

This is exactly the form of accumulating trace[7]: □

---

[6]Actually, the online method that approximates the λ-return updates was discovered before the identification of λ-return as a compound target. However the introduction to the online method via the notion of compound targets are beneficial for understanding.

[7]People refer to "eligibility" trace as the family of the algorithms that employ trace vectors to incrementally approximate something that cannot be calculated directly. There are several variants of the eligibility traces. The accumulating traces are the original and the most classical one.

Note that the equality holds only when the trajectory is infinitely long. When it is not (as it will always be), there will be differences between this online approximation algorithm, which is named $TD(\lambda)$ and the offline λ-return algorithm. Interestingly, even if so, it is proved in [6] that the TD(λ), the algorithm that approximates with eligibility traces vectors, achieves convergence to a fixed-point near the true value almost surely.

Naturally, scalar parameter λ for all states can be generalized to state-based $\lambda(\cdot)$ [11, 20, 21, 23, 30]. If we index the states and concatenate all the $\lambda(s)\forall s \in \mathcal{S}$, we arrive at the generalized $\boldsymbol{\lambda}$-return[8]:

Definition 15 ($\boldsymbol{\lambda}$-return)**.** *The generalized state-based $\boldsymbol{\lambda}$-return $G_t^{\boldsymbol{\lambda}}$, where $\boldsymbol{\lambda} \equiv [\lambda_1, \ldots, \lambda_i \equiv \lambda(s_i), \ldots, \lambda_{|\mathcal{S}|}]^T$, for state S_t in a trajectory τ is recursively defined as*

$$G_t^{\boldsymbol{\lambda}} = R_{t+1} + \gamma_{t+1}[(1 - \lambda_{t+1})V(S_{t+1}) + \lambda_{t+1}G_{t+1}^{\boldsymbol{\lambda}}]$$

where $G_t^{\boldsymbol{\lambda}} = 0$ for $t \geq |\tau|$.

Convergence of method using $\boldsymbol{\lambda}$-return as the update target can be shown based on the fact that the generalized Bellman operator with state-dependent discount and trace decay is a contraction [24]. On the other hand, time-dependent decay may not be equipped with well-defined fixed points.

Similarly, online approximations using traces exist for offline updates with λ-return.

Proposition 2 (Generalized Trace Approximation)**.** *Given an MDP, an infinitely long trajectory under policy π, the state-based discount function $\gamma : \mathcal{S} \to [0, 1]$ and the state-based trace-decay function $\lambda : \mathcal{S} \to [0, 1]$, the updates using $\boldsymbol{\lambda}$-return as targets can be approximated with online updates. More specifically, the update rules are:*

$$\boldsymbol{z}_t = \gamma(\boldsymbol{x}_t)\lambda(\boldsymbol{x}_t)\boldsymbol{z}_{t-1} + \boldsymbol{\nabla}V(\boldsymbol{x}_t, \boldsymbol{w}_t)$$

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}_t + \alpha[R_{t+1} + \gamma V(\boldsymbol{x}_{t+1}, \boldsymbol{w}_t) - V(\boldsymbol{x}_t, \boldsymbol{w}_t)]\boldsymbol{z}_t$$

---

[8]$\boldsymbol{\lambda}$ here is in bold font to emphasize that it is a vector.

where α is some appropriate learning rate.

Proof. Let $\gamma_k \equiv \gamma(S_k)$ and $\lambda_k \equiv \lambda(S_k)$.

$G_t^{\boldsymbol{\lambda}} - V(\boldsymbol{x}_t)$

$= -V(\boldsymbol{x}_t) + R_{t+1} + \gamma_{t+1}(1 - \lambda_{t+1})V(\boldsymbol{x}_{t+1}) + \gamma_{t+1}\lambda_{t+1}G_{t+1}^{\boldsymbol{\lambda}}$

$= -V(\boldsymbol{x}_t) + R_{t+1} + \gamma_{t+1}(1 - \lambda_{t+1})V(\boldsymbol{x}_{t+1}) + \gamma_{t+1}\lambda_{t+1}(R_{t+2} + \gamma_{t+2}(1 - \lambda^{(t+2)})V(\boldsymbol{x}_{t+2}) + \gamma_{t+2}\lambda^{(t+2)}G_{t+2}^{\boldsymbol{\lambda}})$

$= -V(\boldsymbol{x}_t) + R_{t+1} + \gamma_{t+1}(1 - \lambda_{t+1})V(\boldsymbol{x}_{t+1}) + \gamma_{t+1}\lambda_{t+1}R_{t+2} + \gamma_{t+1}\lambda_{t+1}\gamma_{t+2}(1 - \lambda^{(t+2)})V(\boldsymbol{x}_{t+2}) + \gamma_{t+1}\lambda_{t+1}\gamma_{t}$

$= \cdots$

$= \sum_{k=t}^{\infty} (\gamma_k\lambda^{(k)})^{k-t}(R_{k+1} + \gamma_k V(\boldsymbol{x}_{k+1}) - V(\boldsymbol{x}_k))$

$\square$

Generalization from scalar λ to state-based $\boldsymbol{\lambda}$ greatly increases the potential of trace-based policy evaluation in a sense that significantly more degrees of freedom are unlocked for the mixing of multi-step targets and potentially the achievement of update targets with significantly less bias and variance.

The complexity of the online trace updates stops anyone from optimizing it, unless the true equivalence can be established between the online and offline algorithms.

Recently, a new family of "true online" algorithms has been discovered, which achieves exact equivalence of the online approximation and the offline updates. The equivalence is achieved by maintaining extra traces that correct the updates towards a true convex combination of multi-step targets.

Fact 6 (True Online Equivalence). *The following incremental update rules achieve exact equivalence to updates using $\boldsymbol{\lambda}$-return targets using linear function approximations:*

$$\delta_t = R_{t+1} + \gamma_{t+1}\boldsymbol{x}_{t+1}^T w_{(t)} - \boldsymbol{x}_t^T w_{(t)}$$

$$\boldsymbol{z}_{(t)} = \gamma_t\lambda_{(t)}\boldsymbol{z}_{(t-1)} + \boldsymbol{x}_t - \alpha\gamma_{t+1}\lambda_{t+1}(\boldsymbol{z}_{(t)}^T\boldsymbol{x}_t)\boldsymbol{x}_t$$

$$\boldsymbol{w}_{(t+1)} = \boldsymbol{w}_{(t)} + \alpha \delta_t \boldsymbol{z}_{(t)} + \alpha(\boldsymbol{w}_{(t)}^T \boldsymbol{x}_t - \boldsymbol{w}_{(t-1)}^T \boldsymbol{x}_t)(\boldsymbol{z}_{(t)} - \boldsymbol{x}_t)$$

where α is the step-size hyperparameter that is consistent from both the forward view and the backward view.

With the true online algorithms, we can enjoy the efficiency of online updates and the guarantee of convergence from the offline returns simultaneously. The equivalence also could serve as a bridge to optimize the online updates using the mathematical properties of the backward view.

Very recently, it is discovered that, it is possible to learn the statistics, *e.g.* variance, second-moment, of $\boldsymbol{\lambda}$-return online using eligibility traces. The online learning of these statistics as auxiliary tasks provides more information for online adaptation of the learning parameters. Here we introduce 2 relevant ones.

The first method VTD comes from [28], which seeks to learn the second moment of $\boldsymbol{\lambda}$-return. A Bellman operator is constructed for the squared $\boldsymbol{\lambda}$-return, which is the second moment of return. A fixed point objective Var-MSPBE, similar to MSPBE, is introduced. The recursive form for the squared return is:

$$
\begin{aligned}
\left(G_t^{\boldsymbol{\lambda}}\right)^2 &= \left(\rho_t R_{t+1} + \gamma_{t+1}[(1 - \lambda_{t+1})V(S_{t+1}) + \lambda_{t+1}G_{t+1}^{\boldsymbol{\lambda}}]\right)^2 \\
&= \overline{R}_{t+1} + \overline{\gamma}_{t+1}(G_{t+1}^{\boldsymbol{\lambda}})^2
\end{aligned} \tag{2.19}
$$

where for a given $\boldsymbol{\lambda} \in [0, 1]^{|\mathcal{S}|}$,

$$\overline{G}_t \equiv R_{t+1} + \gamma_{t+1}(1 - \lambda_{t+1})M(\boldsymbol{x}_{t+1}; \boldsymbol{w}_M) = R_{t+1} + \gamma_{t+1}(1 - \lambda_{t+1})\boldsymbol{w}_M^T \boldsymbol{x}_{t+1}$$

$$\overline{R}_{t+1} \equiv \rho_t^2 \overline{G}_t^2 + 2\rho_t^2 \gamma_{t+1}\lambda_{t+1}\overline{G}_t G_{t+1}^{\boldsymbol{\lambda}}$$

$$\overline{\gamma}_{t+1} \equiv \rho_t^2 \gamma_{t+1}^2 \lambda_{t+1}^2$$

Here M is the linear approximation of the expected squared return, parameterized by $\boldsymbol{w}_M$. It has been proved that even if $\overline{\gamma}_{t+1} < 1$ cannot be guaranteed, the Bellman operator for these new types of "reward" $\overline{R}_{t+1}$ and discount function $\overline{\gamma}_{t+1}$ is still a contraction un-

der appropriate conditions, which is when the second moment of return is finite. These tell us that the second moment of return can be learned in the same way as the first moment of return (which is the expected λ-return) using MC, (trace-enabled) TD or even GTDs. Notice that, using estimated first moment and second moment we can indirectly estimate the variance of λ-return by constructing $Var[G_t^\lambda] = \mathbb{E}[(G_t^\lambda)^2] - \mathbb{E}^2[G_t^\lambda]$. Interestingly, such estimation is proved to be possible if carried out directly, which leads to the following.

The second method DVTD is more recent, published in [18], which seeks to learn the variance of the λ-return directly by constructing a Bellman operator for the variance of the λ-return. The recursive form for the variance of the λ-return is:

$$Var[G_t^\lambda] = \mathbb{E}\left[\delta_t^2 + (\gamma_{t+1}\lambda_{t+1})^2 Var[G_{t+1}^\lambda]|S_t = s\right] \tag{2.20}$$

It has been proved that the induced Bellman operator converges under appropriate assumptions. Similar to how we would apply VTD, this method can be used to learn the variance of λ-return flexibly if we treat reward as δ_t^2 and discount to be $(\gamma_{t+1}\lambda_{t+1})^2$.

Trace based true online methods will be used as baseline methods to validate the empirical performance of the contributed method in this thesis, in Chapter 5.

2.9 Policy Gradient Methods

Just like the value estimates, policies can also make use of the power of generalization by parameterization. In this section, we introduce the ways of dealing with parameterized policies that selects actions, independent of the value estimates[9].

We can use the notation $\boldsymbol{\theta}$ for the policy's parameter vector. Thus we write $\pi(a|s;\boldsymbol{\theta}) = \mathbb{P}\{A_t = a|S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}$ for the probability that action a is taken at time t given that the environment is in state s at time t with parameter $\boldsymbol{\theta}$.

---

[9]We focus on the type of parameterized policies disentangled from the value estimates.

We seek to maximize the value function of the starting states by optimizing $\boldsymbol{\theta}$. The representative method is to conduct gradient ascent, whose performance is guaranteed by the following[10]:

Theorem 4 (Episodic Discounted Policy Gradient). *Given a policy $\pi(\cdot; \boldsymbol{\theta})$, let the true state-action value function be $q_\pi(\cdot)$, the true state value function be $v_\pi(\cdot)$, the starting state distribution be d_0 and the state distribution be d_π. The gradient of $v_\pi(\cdot)$ w.r.t. $\boldsymbol{\theta}$ satisfies:*

$$\boldsymbol{\nabla} v_\pi(s_0) \propto \sum_s d_\pi(s) \cdot \gamma(s_0 \to s, \pi) \cdot \sum_a (q_\pi(s, a) - h(s)) \boldsymbol{\nabla} \pi(a|s; \boldsymbol{\theta}) \tag{2.21}$$

where $s_0 \sim d_0$, $s \sim d_\pi$, $h : \mathcal{S} \to \mathbb{R}$ is any random variable that is not dependent on a and $\gamma(s_0 \to s, \pi)$ is the expected cumulative product of discount factors transitioning from s_0 to s, specifically:

$$\gamma(s_0 \to s, \pi) \equiv \mathbb{E}_{\tau \sim \pi} \left[\sum_{k=0}^\infty \mathbb{P}\{s_0 \xrightarrow{\tau} s, k, \pi\} \left(\prod_{y=s_1}^s \gamma(y) \right) \right]$$

where τ is a trajectory sampled with π, $\mathbb{P}\{s \xrightarrow{\tau} x, k, \pi\}$ is the probability of transitioning according to trajectory τ from state s to state x in exactly k steps under policy π and $\prod_{y=s_1}^s \gamma(y)$ is the cumulative product of discount factors following the trajectory τ along $s_0, s_1, s_2, \ldots, s_{k-1}, s$.

---

[10]This is an original proof for the generalized state-based γ case.

Proof. The gradient of v_π can be written in terms of q_π as

$$\nabla v_\pi(s) = \nabla \left[\sum_a \pi(a|s) q_\pi(s,a) \right], \forall s \in \mathcal{S}$$

$$= \sum_a [q_\pi(s,a) \nabla \pi(a|s) + \pi(a|s) \nabla q_\pi(s,a)]$$

$$= \sum_a \left[q_\pi(s,a) \nabla \pi(a|s) + \pi(a|s) \nabla \sum_{s',r} p(s',r|s,a)(r + \gamma(s')v_\pi(s')) \right]$$

$$= \sum_a \left[q_\pi(s,a) \nabla \pi(a|s) + \pi(a|s) \sum_{s',r} p(s',r|s,a)\gamma(s') \nabla v_\pi(s') \right]$$

$$= \cdots (\text{keep unrolling } \nabla v_\pi(\cdot))$$

$$= \sum_{x \in \mathcal{S}} \sum_{\tau \sim \pi} \sum_{k=0}^{\infty} \left[\mathbb{P}\{ s \xrightarrow{\tau} x, k, \pi \} \cdot \left(\prod_{y=s'}^{x} \gamma(y) \right) \cdot \sum_a \nabla \pi(a|x) q_\pi(x,a) \right]$$

(2.22)

The value of the initial state is what we care about, thus

$$\nabla v_\pi(s_0)$$

$$= \sum_s (\sum_{\tau \sim \pi} \sum_{k=0}^{\infty} \mathbb{P}\{ s_0 \xrightarrow{\tau} s, k, \pi \} \left(\prod_{y=s_1}^{s} \gamma(y) \right)) \sum_a q_\pi(s,a) \cdot \left(\prod_{y=s_1}^{s} \gamma(y) \right) \cdot \nabla \pi(a|s)$$

$$= \sum_s \eta_\pi(s) \sum_a q_\pi(s,a) \cdot \gamma(s_0 \to s, \pi) \cdot \nabla \pi(a|s)$$

(2.23)

$\eta_\pi(s)$ is the expected number of visits to state s under π

$$\propto \sum_s d_\pi(s) \sum_a q_\pi(s,a)\gamma(s_0 \to s, \pi) \cdot \nabla \pi(a|s)$$

We notice that:

$$0 = \nabla 1$$

$$= \nabla \sum_a \pi(a|s)$$

$$= h(\cdot) \nabla \sum_a \pi(a|s)$$

$$= \sum_a h(\cdot) \nabla \pi(a|s) \quad \text{as long as } h \text{ has nothing to do with } a$$

(2.24)

Thus,

$$
\nabla v_\pi(s_0) = \nabla v_\pi(s_0) + 0
$$

$$
\propto \sum_s d_\pi(s) \cdot \gamma(s_0 \to s, \pi) \cdot \sum_a q_\pi(s, a) \nabla \pi(a|s) + \sum_a h(\cdot) \nabla \pi(a|s) \qquad (2.25)
$$

$$
\propto \sum_s d_\pi(s) \cdot \gamma(s_0 \to s, \pi) \cdot \sum_a \left(q_\pi(s, a) - h(s) \right) \nabla \pi(a|s)
$$

$\square$

When learning online, it is desirable, instead of to sum over a, use A_t to do a stochastic update.

$$
\nabla v_\pi(s_0)
$$

$$
= \mathbb{E}_\pi \left[\gamma(s_0 \to s, \pi) \cdot \sum_a q_\pi(S_t, a) \nabla \pi(a|S_t; \boldsymbol{\theta}) \right]
$$

$$
= \mathbb{E}_\pi \left[\prod_{X=S_1}^{S_t} \gamma(X) \cdot \sum_a \pi(a|S_t; \boldsymbol{\theta}) \cdot q_\pi(S_t, a) \cdot \frac{\nabla \pi(a|S_t; \boldsymbol{\theta})}{\pi(a|S_t; \boldsymbol{\theta})} \right]
$$

$$
= \mathbb{E}_\pi \left[\prod_{X=S_1}^{S_t} \gamma(X) \cdot q_\pi(S_t, a) \cdot \frac{\nabla \pi(a|S_t; \boldsymbol{\theta})}{\pi(a|S_t; \boldsymbol{\theta})} \right]
$$

$$
= \mathbb{E}_\pi \left[\prod_{X=S_1}^{S_t} \gamma(X) \cdot q_\pi(S_t, A_t) \cdot \frac{\nabla \pi(A_t|S_t; \boldsymbol{\theta})}{\pi(A_t|S_t; \boldsymbol{\theta})} \right] \qquad \mathbb{E}_\pi[A_t] = a
$$

$$
= \mathbb{E}_\pi \left[\prod_{X=S_1}^{S_t} \gamma(X) \cdot U_t \cdot \frac{\nabla \pi(A_t|S_t; \boldsymbol{\theta})}{\pi(A_t|S_t; \boldsymbol{\theta})} \right] \qquad \text{as long as } \mathbb{E}_\pi[U_t|S_t, A_t] = q_\pi(S_t, A_t)
$$

$$(2.26)$$

We realize that the corresponding online update rule of the policy gradient theorem has the same problem as that of the gradient of values (2.15) - $q_\pi(\cdot)$ is unknown and must be replaced with an estimator.

h, either a function or a random variable, is recognized as a *baseline*. In general, the baseline leaves the expected value of the gradient unchanged, yet having significant effect on its variance. The possibility of variance reduction is enabled by the idea recognized as *control variates*. One popular choice of the baseline is an estimate of the state value V.

Replacing $q_\pi(s, a) - h(s)$ with $G - V(s)$, where G is the MC return estimator, then we will arrive at the simplest policy gradient method, which is named *REINFORCE*. However, its details will not be discussed since they are not related to the contribution of this thesis.

2.9.1 Actor-Critic

Enabling bootstrapping in policy gradient methods is crucial, since the bias introduced through bootstrapping reduces the variance and boosts sample efficiency (makes learning faster and more accuracy). REINFORCE with baseline is unbiased and will converge asymptotically to a local minimum, but since it has no bootstrapping and updates only upon a high-variance target (MC return), it is problematic for online learning. Actor-Critic methods eliminate these inconveniences with TD and through the mixing of multi-step targets, we can flexibly determine the degree of bootstrapping.

The first and simplest instance of these methods is the 1-step actor-critic method, which is fully online and incremental, yet avoid the complexities of eligibility traces. Replacing the target U_t in (2.26) with 1-step target $R_{t+1} + \gamma(S_{t+1})V(S_{t+1})$, we have it as follows:

$$
\begin{aligned}
\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \alpha \left(G_{t:t+1} - V(S_t, \boldsymbol{w}) \right) \frac{\boldsymbol{\nabla}\pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)} \\
&= \boldsymbol{\theta}_t + \alpha \left(R_{t+1} + \gamma(S_{t+1})V(S_{t+1}, \boldsymbol{w}) - V(S_t, \boldsymbol{w}) \right) \frac{\boldsymbol{\nabla}\pi(A_t|S_t, \boldsymbol{\theta}_t)}{\pi(A_t|S_t, \boldsymbol{\theta}_t)}
\end{aligned}
\tag{2.27}
$$

With this, we have the 1-step Actor-Critic method for episodic tasks, as presented in Algorithm 8.

The generalizations to the forward view of n-step methods and then to a λ-return algorithm are straightforward. The episodic Actor-Critic method with eligibility traces is presented as follows, in Algorithm 9.

Algorithm 8: Episodic 1-step Actor-Critic for estimating π_*

Input: $\pi(a|s;\boldsymbol{\theta})$ (differentiable policy parameterization), $V(s;\boldsymbol{\theta})$ (differentiable state-value estimate parameterization), γ (discount function), $\alpha_{\boldsymbol{\theta}}, \alpha_{\boldsymbol{w}}$ (learning rates for actor and critic, respectively), N (maximum number of episodes)

Output: $\pi \approx \pi_*$ (an estimate of the optimal policy), $V(s), \forall s \in \mathcal{S}^+$ (estimated state-values for policy π_*)

Initialize weights $\boldsymbol{\theta}$ and $\boldsymbol{w}$, *e.g.* to $\boldsymbol{0}$

for $n \in \{1, \ldots, N\}$ **do**

    Initialize S // first state of episode

    $I = 1$ // cumulative product of discount factors

    while S *is not terminal* **do**

        $A \sim \pi(\cdot|S, \boldsymbol{\theta})$

        Take action A, observe R, S'

        $\delta = R + \gamma(S')V(S';\boldsymbol{w}) - V(S;\boldsymbol{w})$ // TD error, $V(S';\boldsymbol{w}) \equiv 0$ if S' is terminal

        $\boldsymbol{w} = \boldsymbol{w} + \alpha_{\boldsymbol{w}}\delta \cdot \boldsymbol{\nabla}_{\boldsymbol{w}}V(S;\boldsymbol{w})$// 1-step semi-gradient TD update for critic

        $\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}}I\delta \cdot \boldsymbol{\nabla}_{\boldsymbol{\theta}}ln(\pi(A|S;\boldsymbol{\theta}))$// 1-step update for actor

        $I = I \cdot \gamma(S')$

        $S = S'$

Algorithm 9: Episodic Actor-Critic with Eligibility Traces for estimating π_*

Input: $\pi(a|s; \boldsymbol{\theta})$ (differentiable policy parameterization), $V(s; \boldsymbol{\theta})$ (differentiable state-value estimate parameterization), γ (discount function), $\alpha_{\boldsymbol{\theta}}, \alpha_{\boldsymbol{w}}$ (learning rates for actor and critic, respectively), $\lambda_{\boldsymbol{\theta}}, \lambda_{\boldsymbol{w}}$ (trace-decay functions for actor and critic, respectively), N (maximum number of episodes)

Output: $\pi \approx \pi_*$ (an estimate of the optimal policy), $V(s), \forall s \in \mathcal{S}^+$ (estimated state-values for policy π_*)

Initialize weights $\boldsymbol{\theta}$ and $\boldsymbol{w}$, *e.g.* to $\boldsymbol{0}$

for $n \in \{1, \dots, N\}$ **do**

    Initialize S // first state of episode

    $I = 1$ // cumulative product of discount factors

    $\boldsymbol{z_\theta} = \boldsymbol{0}$ // eligibility trace for $\boldsymbol{\theta}$

    $\boldsymbol{z_w} = \boldsymbol{0}$ // eligibility trace for $\boldsymbol{w}$

    while S *is not terminal* **do**

        $A \sim \pi(\cdot|S, \boldsymbol{\theta})$

        Take action A, observe R, S'

        $\delta = R + \gamma(S')V(S'; \boldsymbol{w}) - V(S; \boldsymbol{w})$ // TD error, $V(S'; \boldsymbol{w}) \equiv 0$ if S' is terminal

        $\boldsymbol{z_w} = \gamma(S)\lambda_{\boldsymbol{w}}(S)\boldsymbol{z_w} + \boldsymbol{\nabla_w}V(S, \boldsymbol{w})$ // accumulating traces for $\boldsymbol{z_w}$

        $\boldsymbol{z_\theta} = \gamma(S)\lambda_{\boldsymbol{\theta}}(S)\boldsymbol{z_\theta} + I \cdot \boldsymbol{\nabla_\theta}ln(\pi(A|S; \boldsymbol{\theta}))$ // accumulating traces for $\boldsymbol{z_\theta}$

        $\boldsymbol{w} = \boldsymbol{w} + \alpha_{\boldsymbol{w}}\delta \cdot \boldsymbol{z_w}$

        $\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha_{\boldsymbol{\theta}}\delta \cdot \boldsymbol{z_\theta}$

        $I = I \cdot \gamma(S')$

        $S = S'$

Chapter 3

Sample Efficiency of Temporal Difference Learning

Learning faster and more accurately.

3.1   Sample Efficiency

Sample efficiency is widely regarded as one of the main bottlenecks of the existing RL methods, for both prediction and control. For example, on even very simple tasks, RL agents often require very large amount of agent-environment interactions in order to predict or control well. For prediction, it is observed from the average number of samples (interactions) to reach certain accuracy of the value estimate. For control, it is observed from the average number of samples to reach an optimal policy (or a good policy, if there is no guarantee to reach optimal). To our knowledge, there is no formal definition of sample efficiency. However, it can be naturally compared among different methods on identical tasks, based on their learning performance. Intuitively for prediction, having a higher sample efficiency, equivalent to having a lower sample efficiency, achieves higher learning speed as well as better accuracy. This thesis focuses on improving the sample efficiency of eligibility trace based methods, *e.g.* TD(λ), which are widely adopted.

3.2 Sample Efficiency & Tuning λ

Not only shown in [9, 19, 20], but also pervasively in RL practices that, given appropriate learning rates and amount of steps, TD(λ) with different λ performs very differently: they exhibit patterns of U-shaped curves when comparing the squared error of their value estimates against the true value, *i.e.* the sample efficiency of TD(λ) is sensitive to the value of λ. To deal with this [9, 19], the normal approach would be simply to run TD(λ) with different values of λ and pick whichever λ value that yields the best performance.

This brute-force search is a general strategy for hyperparameter search. In prediction tasks, where a fixed policy is given and the distribution of samples do not change with the values of λ, this parallel search requires no more samples from the environment and is effective but still very computationally expensive. In control tasks, where the quality of policy evaluation also determines the change of policies, this method is unsatisfactory.

The sensitivity naturally leads researchers to investigate into possible methods of adapting λ for better sample efficiency. Different interpretation of methods give rise to different approaches, which are currently few and mostly limited to special cases.

3.2.1 Counting Heuristics

One can think of λ-return constructed with a mix of current value estimate and the experienced reward transitions (MC return). Thus, from the view of credit assignment, it is appropriate to interpret λ as "confidence" to the current value estimates against the MC return: the more confident the method is to the current value estimate, the lower λ-values it should use. Thus, counting methods keep count of the visits of states (or neighborhoods of states) and adjust λ based on the counts to reflect the confidence.

3.2.2 Bypass Methods

Some methods seek to bypass the framework of TD(λ) and ameliorate the mechanisms of TD. [10] introduces TD$_\gamma$ as a method to remove the λ parameter altogether. Their ap-

proach, however, has not been extended to the off-policy setting and their full algorithm is computationally expensive for incremental estimation, while their incremental variant introduces a sensitive meta-parameter. [16] proposes a method to mitigate the propagation of errors by adaptively switching from 1-step TD updates to MC updates in each state.

3.2.3 Meta-Learning Methods

Meta-learning, recognized often as "learning to learn", is a methodology with which a meta-objective upon the problem objectives, in the RL case either optimizing errors of value estimates or improving policies, is proposed to enhance the problem solving process. Meta-learning algorithms adapt the parameters of the agent continuously, based on the stream of experience and some notion of the agent's own learning progress. Meta-learning has been heavily proposed as an approach for adapting the learning rates [5] however rarely for the trace-decay parameter λ.

Meta-learning λ dates back to [22], where adaptation of λ using an meta-learning algorithm is proposed, which however unfortunately requires access to the transition model of the MDP. [7] explores a Bayesian variant of TD learning by assuming the specific forms of the stochasticity of the environment. Yet, such method can only be used offline. [29] proposed approximate meta-gradients for directly optimizing the prediction error or the policy. However, the assumptions upon which the meta-gradients are derived are very strong and they are very likely unsatisfied for many environments. Thus the inaccuracy of the meta-gradients may lead to failed improvements of sample efficiency. Also, it is a offline λ-return method that does not use efficient updates of eligibility traces. Some methods have been proposed for online meta-learning, with high extra computational complexities that are intolerable for practical use [12]. [28] proposed an online meta-learning methods that achieves off-policy compatible adaptation of state-based λ's without introducing new meta-parameters by locally optimizing a bias-variance tradeoff of a surrogate

update target. However, the method optimizes a surrogate target which has huge space to be improved.

To summarize, the existing methods have trouble satisfying the following properties for various difficulties [9, 17]:

1. Achieve the optimization of sample efficiency provably.

2. Achieve adjustment of λ online.

3. Achieve compatibility with off-policy learning.

4. Incur low additional computational costs.

5. Does not change the mechanisms of TD(λ) *s.t.* they may be universally applied as a plugin for adapting λ.

6. Does not require access to MDP dynamics.

Although this long-history of prior work has helped develop our intuitions about λ, the available solutions are still far from the use cases outlined above. In this thesis, we build upon [28] to achieve a principled method for meta-learning state- or feature-based parametric λs [1] which aims directly at the sample efficiency. Under some assumptions, the method has the following properties:

1. Meta-learns online and uses only incremental computations, incurring the same computational complexity as usual value-based eligibility-trace algorithms, such as TD(λ).

2. Optimizes (approximately) the overall quality of the update targets.

3. Works in off-policy cases.

4. Works with function approximation.

5. Works with adaptive learning rate.

---

[1]For the tabular case, it is state-based and for the function approximation case it is feature-based.

3.2.4 Background Knowledge

Asymptotic convergence in the tabular case is not always useful in practice: we cannot afford infinite episodes nor can we apply tabular methods to environments with the number of states too large to be enumerated, *e.g.*, continuous state spaces. Yet, with limited episodes or function approximations, we lose most of the convergence guarantees.

Though TD(λ), under some cases, can be analyzed using λ-returns, the actual complicated trace updates make it almost unlikely to optimize the value error directly by optimizing λ. Thus in this paper, we develop our ideas upon the *optimization of update targets*. To our knowledge, this is the first paper that explicitly utilize this idea to improve sample efficiency in a principled way. Update targets have important connections to the quality of the learned value estimate [19], which we ultimately pursue in policy evaluation tasks.

Proposition 3. *Given suitable learning rates, value estimates using targets with lower overall target error achieve lower overall value error.*

Starting from the same initial candidate distribution for the learnable parameters, with the same value estimation method, the convergence rate is the same for all the instances towards their update targets. According to stochastic approximation theory, given the same number of update steps, the instances with better update targets yield value estimates with lower MSE compared to the ground truth. The conclusion is very powerful: we can make prediction more sample efficient by using better update targets, which in trace-based prediction means optimizing the difference between the update target and the true value function *w.r.t.* λ^2 [31]. This is the core idea for the λ-greedy algorithm which we are about to discuss as well as our proposed method.

How are we exactly going to learn faster and achieve better accuracy by changing λ? We shall first review a semi-principled approach proposed in [28]. The λ-greedy method is a landmark for online adaptation of λ: it achieves off-policy compatible meta-learning

---

[2]Note that with function approximation, updates are never conducted for the features of the terminal states in RL algorithms. Yet, the target error and value error for terminal states should always be set 0, since these states are always identifiable for they are accompanied by a terminal signal.

without introducing meta-parameters. The idea of this paper is going to be the foundation of the contributions of this thesis.

3.2.5 λ-Greedy [28]: An Existing Work

It is intuitively clear that using state-based values of λ provides more flexibility than using a constant for all states. Also, state-based λs are equipped with well-defined fixed points. Out of these reasons, TD(λ) with different λ values for different states has been proposed as a more general formulation of trace-based prediction methods. While preserving good mathematical properties such as convergence to fixed points, this generalization also unlocks significantly more degrees of freedom than only adapting a constant λ for every state.

[28] investigated the use of state-based λs, while outperforming constant λ values on some prediction tasks. The authors implicitly conveyed the idea that better update targets lead to better sample efficiency, *i.e.*, update targets with smaller Mean Squared Error (MSE) lead to smaller MSE in learned values. Their proposed online adaptation of λ is achieved via efficient incremental estimation of statistics about the update targets, gathered by some auxiliary learners. Yet, such method does not seek to improve the overall sample efficiency, because the meta-objectives does not align with the overall target quality.

The idea is to minimize the error between a pseudo target $\tilde{G}(s_t)$ and the true value $v(s_t)$, where the pseudo target is defined as:

$$\tilde{G}(s_t) \equiv \tilde{G}_t \equiv R_{t+1} + \gamma_{t+1}[(1 - \lambda_{t+1})V(s_{t+1}) + \lambda_{t+1}G_{t+1}]$$

where $\lambda_{t+1} \in [0, 1]$ and $\lambda_k = 1, \forall k \geq t + 2$.

With this we can find that $\tilde{J}(s_t) \equiv \mathbb{E}[(\tilde{G}_t - \mathbb{E}[G_t])^2]$ is a function of only λ_{t+1} (given the value estimate $V(s_{t+1})$). The greedy objective corresponds to minimizing the error of the pseudo target $\tilde{G}_t$:

Theorem 5 (Greedy Objective Minimizer). *Let t be the current timestep and s_t be the current state, the agent takes action at s_t s.t. it will transition into s_{t+1} at $t+1$. Given the pseudo update target $\tilde{G}_t$ of s_t, the minimizer λ^*_{t+1} of the target error of the state $\tilde{J}(s_t) \equiv \mathbb{E}[(\tilde{G}_t - \mathbb{E}[G_t])^2]$ w.r.t. λ_{t+1} is:*

$$\lambda^*_{t+1} = \frac{(V(s_{t+1}) - \mathbb{E}[G_{t+1}])^2}{\mathbb{E}^2[V(s_{t+1}) - G_{t+1}] + Var[G_{t+1}]} \tag{3.1}$$

where G_{t+1} is the MC return for state s_{t+1}.

The proof can be found in [28]. Adaptation based on (3.1) requires knowledge of $\mathbb{E}[G_{t+1}]$ and $Var[G_{t+1}]$: and since they are not known, they are approximated using auxiliary learners, that run in parallel with the value learner, for the additional distributional information needed, preferably in an incremental manner. The solutions for learning these have been contributed in [18, 28] and illustrated in (2.19) and (2.20). These methods learn the variance of λ-return in the same way TD methods learn the value function, however with different "rewards" and "discount factors" for each state, that can be easily obtained from the known information without incurring new interactions with the environment.

The λ-greedy method shows strong boost for sample efficiency in some prediction tasks. However, there are two reasons that λ-greedy has much space to be improved. The first is that the pseudo target $\tilde{G}_t$ used for optimization is not actually the update target used in TD(λ) algorithms: we will show that it is rather a compromise for a harder optimization problem; The second is that setting the λs to the minimizers does not help the overall quality of the update target: the update targets for every state is controlled by the whole λ, thus unbounded changes of λ for one state will inevitably affect the other states as well as the overall target error.

From the next chapter, we build upon the mindset provided in [28] to propose our method META.

Chapter 4

META

The contributed Meta Eligibility Trace Adaptation method[1]. In this chapter, we propose our method META, whose goal is to optimize the overall target error for sample efficiency.

4.1 On-policy Decomposition

We first investigate how the goal of optimizing overall target error can be achieved online. A key to solving this problem is to acknowledge the following points:

1. The states that the agent meets carrying out the policy π follows the "on-policy" distribution of d_π.

2. The overall value error and its surrogate overall target error is mixed of per-state errors according to the d_π.

3. Jointly optimizing the overall target error is possible via optimizing each state target error with a consistent optimization method.

With this, we develop the following theorem to construct this process.

Theorem 6. *Given an MDP and target policy π, let $D = diag(d_\pi(s_1), \cdots, d_\pi(s_{|\mathcal{S}|}))$ be the diagonalized on-policy distribution and $\hat{G} \equiv [G_{s_1}(\boldsymbol{\lambda}), \cdots, G_{s_{|\mathcal{S}|}}(\boldsymbol{\lambda})]^T$ be an enumeration random*

---

[1]A brief version was represented in [31].

vector of the update targets of all the states in $\mathcal{S}$, in which the targets are all parameterized by a shared parameter vector λ. Stochastic gradient descent on the overall target error $J(\hat{\boldsymbol{G}}, \mathcal{S}) \equiv \frac{1}{2} \cdot \mathbb{E}_\pi \left[\| D^{1/2} \cdot (\hat{\boldsymbol{G}} - \boldsymbol{v}) \|_2^2 \right]$ can be achieved by doing 1-step gradient descent on the state target error $J(\hat{G}_s, s) \equiv \frac{1}{2} \cdot (\hat{G}_s(\boldsymbol{\lambda}) - v_s)^2$ of update target $\hat{G}_s$ for every state s the agent is in when acting upon π. Specifically:

$$\nabla_{\boldsymbol{\lambda}} J(\hat{\boldsymbol{G}}, \mathcal{S}) \propto \sum_{s \sim \pi} \nabla_{\boldsymbol{\lambda}} J(\hat{G}_s, s)$$

Proof. According to the definition of overall target error,

$$J(\boldsymbol{\lambda}) \equiv \sum_{s \in \mathcal{S}} d_\pi(s) \cdot J_s(\boldsymbol{\lambda})) = \sum_{s \in \mathcal{S}} d_\pi(s) \cdot \mathbb{E}[G^{\boldsymbol{\lambda}}(s) - v(s)]^2$$

If we take the gradient *w.r.t.* $\boldsymbol{\lambda}^{(t+1)}$ we can see that:

$\nabla J(\hat{\boldsymbol{G}}, \mathcal{S})$

$= \sum_{s \in \mathcal{S}} d_\pi(s) \cdot \nabla J(\hat{G}_s, s)$

push the gradient inside

$= \sum_{s \in \mathcal{S}} \sum_{k=0}^{\infty} \mathbb{P}\{s_0 \to s, k, \pi, s_0 \sim d(s_0)\} \cdot \nabla J(\hat{G}_s, s)$

$\mathbb{P}\{\cdots\}$ is the prob. of $s_0 \to \cdots \to s$ in exactly k steps,

s_0 is sampled from the starting distribution $d(s_0)$.

$= \sum_{s \in \mathcal{S}} \sum_{k=0}^{\infty} \sum_{\tau} \mathbb{P}\{s_0 \xrightarrow{\tau} s, k, \pi, s_0 \sim d(s_0)\} \cdot \nabla J(\hat{G}_s, s)$

τ is a trajectory starting from s_0 and transitioning to s in exactly k steps.

equivalent to summing over the experienced states under π

$= \sum_{s \sim \pi} \nabla J(\hat{G}_s, s)$

$\square$

Corollary 1. *When the agent is acting upon another behavior policy b, then the theorem still holds if the gradients of the target error for each state s is weighted by the cumulative product ρ_{acc} of*

importance sampling ratios from the beginning of the episode until s. Specifically:

$$\nabla_{\boldsymbol{\lambda}} J(\hat{\boldsymbol{G}}, \mathcal{S}) \propto \sum_{s \sim b} \rho_{acc} \cdot \nabla_{\boldsymbol{\lambda}} J(\hat{G}_s, s)$$

Proof. According to the definition of overall target error,

$\nabla J(\hat{\boldsymbol{G}}, \mathcal{S})$

$= \sum_{s \in \mathcal{S}} \sum_{k=0}^{\infty} \sum_{\tau} \mathbb{P}\{s_0 \xrightarrow{\tau} s, k, \pi, s_0 \sim d(s_0)\} \cdot \nabla J(\hat{G}_s, s)$

see the proof of the theorem

$= \sum_{s \in \mathcal{S}} \sum_{k=0}^{\infty} \sum_{\tau} \cdots p(\tau_{k-1}, a_0, s) \pi(a_{k-1}|\tau_{k-1}) \cdot \nabla J(\hat{G}_s, s)$

τ_i is the $i+1$-th state of trajectory τ and $p(s, a, s^{'})$ is the prob. of $s \xrightarrow{a} s^{'}$ in the MDP

$= \sum_{s \in \mathcal{S}} \sum_{k=0}^{\infty} \sum_{\tau} \cdots p(\tau_{k-1}, a_0, s) \frac{\pi(a_{k-1}|\tau_{k-1})}{b(a_{k-1}|\tau_{k-1})} b(a_{k-1}|\tau_{k-1}) \cdot \nabla J(\hat{G}_s, s)$

for the convenience of injecting importance sampling ratios

$= \sum_{s \in \mathcal{S}} \sum_{k=0}^{\infty} \sum_{\tau} \cdots p(\tau_{k-1}, a_0, s) \rho_{k-1} b(a_{k-1}|\tau_{k-1}) \cdot \nabla_{\boldsymbol{\lambda}} J_s(\boldsymbol{\lambda}))$

$\rho_i \equiv \frac{\pi(a_i|\tau_i)}{b(a_i|\tau_i)}$ is the importance sampling ratio

$= \sum_{s \in \mathcal{S}} \sum_{k=0}^{\infty} \sum_{\tau} \rho_{0:k-1} \cdots \cdot p(\tau_{k-1}, a_0, s) b(a_{k-1}|\tau_{k-1}) \cdot \nabla J(\hat{G}_s, s)$

$\rho_{0:i} \equiv \prod_{v=0}^{i} \rho_v$ is the cumulative product of importance sampling ratios of τ from τ_0 to τ_i

$\approx \sum_{s \sim b} \rho_{acc} \cdot \nabla J(\hat{G}_s, s)$

equivalent to summing over the experienced states under b

$\square$

Note that we can replace the gradient operator with other consistent optimization operators that is consistent with the linearity of the decomposition of errors, *e.g.* the partial derivative operators. The idea draws similarity to emphatic methods that use importance

sampling ratios to correct the distribution of updates [24]. The theorem and the corollary apply for general parametric update targets including the λ-return, whose target error we seek to optimize in this paper. Due to the nature of function approximation, optimizing λ for each state may inevitably affect the error for other states, *i.e.*, decreasing target error for one state may increase those for others. The theorem shows if we can do gradient descent on the target error of the states according to d_π, we can achieve optimization on the overall target error, assuming the value function is changing slowly.

4.2   Approximation of State Gradients

The problem left for us is to find a way to calculate or approximate the gradients of λ for the state target errors.

Sadly, we can find that the exact computation of the per-state gradient seems infeasible in the online setting: in the state-based λ setting, the λ-return for every state is dependent on every λ of every state. These states are unknown before observation.

To remedy this, we propose a method to estimate this gradient by estimating the partial derivatives in the dimensions of the gradient vector, which are further estimated online using auxiliary learners that estimates the distributional information of the update targets. The method can be interpreted as optimizing a bias-variance tradeoff.

Proposition 4. *Let t be the current timestep and s_t be the current state (the state corresponding to the time t). The agent takes action a_t at s_t and will transition into s_{t+1} at $t+1$ while receiving reward R_{t+1}. Suppose that R_{t+1} and G_{t+1}^λ are uncorrelated, given the update target G_t^λ for state s_t, the (semi)-partial derivative of the target error $J_{s_t}(\boldsymbol{\lambda}) \equiv 1/2\mathbb{E}[(G_t^\lambda - \mathbb{E}[G_t])^2]$ of the state s_t w.r.t. $\lambda_{t+1} \equiv \lambda(s_{t+1})$ is:*

$$
\begin{aligned}
\frac{\partial J_{s_t}(\boldsymbol{\lambda})}{\partial \lambda_{t+1}} =& \gamma_{t+1}^2 [\lambda_{t+1} \left[(V(s_{t+1}) - \mathbb{E}[G_{t+1}^\lambda])^2 + Var[G_{t+1}^\lambda] \right] \\
&+ (\mathbb{E}[G_{t+1}^\lambda] - V(s_{t+1}))(\mathbb{E}[G_{t+1}] - V(s_{t+1}))]
\end{aligned}
$$

And its minimizer w.r.t. λ_{t+1} is:

$$\operatorname*{argmin}_{\lambda_{t+1}} J_{s_t}(\boldsymbol{\lambda}) = \frac{(V(s_{t+1}) - \mathbb{E}[G^{\boldsymbol{\lambda}}_{t+1}])(V(s_{t+1}) - \mathbb{E}[G_{t+1}])}{(V(s_{t+1}) - \mathbb{E}[G^{\boldsymbol{\lambda}}_{t+1}])^2 + Var[G^{\boldsymbol{\lambda}}_{t+1}]}$$

Proof.

$$2 \cdot J_{s_t}(\boldsymbol{\lambda}) \equiv \mathbb{E}[(G^{\boldsymbol{\lambda}}_t - \mathbb{E}[G_t])^2] = \mathbb{E}^2[G^{\boldsymbol{\lambda}}_t - G_t] + Var[G^{\boldsymbol{\lambda}}_t]$$

$$\mathbb{E}[G^{\boldsymbol{\lambda}}_t - G_t] = \mathbb{E}[R_{t+1} + \gamma_{t+1}((1 - \lambda_{t+1})V(s_{t+1}) + \lambda_{t+1}G^{\boldsymbol{\lambda}}_{t+1}) - (R_{t+1} + \gamma_{t+1}G_{t+1})]$$

$$= \gamma_{t+1}(1 - \lambda_{t+1})V(s_{t+1}) + \gamma_{t+1}\lambda_{t+1}\mathbb{E}[G^{\boldsymbol{\lambda}}_{t+1}] - \gamma_{t+1}\mathbb{E}[G_{t+1}]$$

$$Var[G^{\boldsymbol{\lambda}}_t] = Var[R_{t+1} + \gamma_{t+1}[(1 - \lambda_{t+1})V(s_{t+1}) + \lambda_{t+1}G^{\boldsymbol{\lambda}}_{t+1}]]$$

$$= Var[R_{t+1}] + \gamma^2_{t+1}Var[(1 - \lambda_{t+1})V(s_{t+1}) + \lambda_{t+1}G^{\boldsymbol{\lambda}}_{t+1}]]$$

(assuming R_{t+1} & $G^{\boldsymbol{\lambda}}_{t+1}$ uncorrelated)

$$= Var[R_{t+1}] + \gamma^2_{t+1}\lambda^2_{t+1}Var[G^{\boldsymbol{\lambda}}_{t+1}]$$

($(1 - \lambda_{t+1})V(s_{t+1})$ not random)

$$\frac{\partial J_{s_t}(\boldsymbol{\lambda})}{\partial \lambda_{t+1}}$$

$$\equiv \frac{1}{2} \cdot \frac{\partial}{\partial \lambda_{t+1}} \left(\mathbb{E}[(G_t^{\boldsymbol{\lambda}} - \mathbb{E}[G_t])^2]) \right)$$

$$= \frac{1}{2} \cdot \frac{\partial}{\partial \lambda_{t+1}} \left((\gamma_{t+1}(1 - \lambda_{t+1})V(s_{t+1}) + \gamma_{t+1}\lambda_{t+1}\mathbb{E}[G_{t+1}^{\boldsymbol{\lambda}}] - \gamma_{t+1}\mathbb{E}[G_{t+1}])^2 + Var[R_{t+1}] + \gamma_{t+1}^2\lambda_{t+1}^2 Var[G_{t+1}^{\boldsymbol{\lambda}}]) \right)$$

$$= \frac{\gamma_{t+1}^2}{2} \cdot \frac{\partial}{\partial \lambda_{t+1}} \left((V(s_{t+1}) - \mathbb{E}[G_{t+1}] - \lambda_{t+1}(V(s_{t+1}) - \mathbb{E}[G_{t+1}^{\boldsymbol{\lambda}}]))^2 + \lambda_{t+1}^2 Var[G_{t+1}^{\boldsymbol{\lambda}}] \right)$$

$$= \frac{\gamma_{t+1}^2}{2} \cdot \frac{\partial}{\partial \lambda_{t+1}} ((V(s_{t+1}) - \mathbb{E}[G_{t+1}])^2 + \lambda_{t+1}^2(V(s_{t+1}) - \mathbb{E}[G_{t+1}^{\boldsymbol{\lambda}}])^2$$

$$- 2\lambda_{t+1}(V(s_{t+1}) - \mathbb{E}[G_{t+1}])(V(s_{t+1}) - \mathbb{E}[G_{t+1}^{\boldsymbol{\lambda}}]) + \lambda_{t+1}^2 Var[G_{t+1}^{\boldsymbol{\lambda}}])$$

$$= \frac{\gamma_{t+1}^2}{2} \cdot \frac{\partial}{\partial \lambda_{t+1}} (\lambda_{t+1}^2(V(s_{t+1}) - \mathbb{E}[G_{t+1}^{\boldsymbol{\lambda}}])^2 - 2\lambda_{t+1}(V(s_{t+1}) - \mathbb{E}[G_{t+1}])(V(s_{t+1}) - \mathbb{E}[G_{t+1}^{\boldsymbol{\lambda}}]) + \lambda_{t+1}^2 Var[G_{t+1}^{\boldsymbol{\lambda}}])$$

$$= \gamma_{t+1}^2[\lambda_{t+1} \left((V(s_{t+1}) - \mathbb{E}[G_{t+1}^{\boldsymbol{\lambda}}])^2 + Var[G_{t+1}^{\boldsymbol{\lambda}}] \right) + (\mathbb{E}[G_{t+1}^{\boldsymbol{\lambda}}] - V(s_{t+1}))(\mathbb{E}[G_{t+1}] - V(s_{t+1}))]$$

$$+ \frac{\gamma_{t+1}^2}{2} \left(\lambda_{t+1}^2 \frac{\partial Var[G_{t+1}^{\boldsymbol{\lambda}}]}{\partial \lambda_{t+1}} + 2\lambda_{t+1}(1 - \lambda_{t+1}(V(s_{t+1}) - \mathbb{E}[G_{t+1}^{\boldsymbol{\lambda}}]))\frac{\partial \mathbb{E}[G_{t+1}^{\boldsymbol{\lambda}}]}{\partial \lambda_{t+1}} \right)$$

Note that the term $\frac{\gamma_{t+1}^2}{2} \left(\lambda_{t+1}^2 \frac{\partial Var[G_{t+1}^{\boldsymbol{\lambda}}]}{\partial \lambda_{t+1}} + 2\lambda_{t+1}(1 - \lambda_{t+1}(V(s_{t+1}) - \mathbb{E}[G_{t+1}^{\boldsymbol{\lambda}}]))\frac{\partial \mathbb{E}[G_{t+1}^{\boldsymbol{\lambda}}]}{\partial \lambda_{t+1}} \right)$ is heavily discounted by high orders of $\gamma(\cdot)$ as well as $\lambda(\cdot)$. If we assume that they are negligible, *i.e.* not taking the partial derivatives of the expectation or the variance and regard them as 0, we can obtain the semi-partial derivative, which is a reasonable 1-step approximation of the derivative. If the algorithm runs offline, it is possible to compute the semi-gradient fully.

The minimizer is achieved by setting the partial derivative 0. $\qquad \square$

This proposition constructs a way to estimate the partial derivative that corresponds to the dimension of λ_{t+1} in $\nabla \boldsymbol{\lambda}$, if we know or can effectively estimate the statistics of $\mathbb{E}[G_{t+1}]$, $\mathbb{E}[G_{t+1}^{\boldsymbol{\lambda}}]$ and $Var[G_{t+1}^{\boldsymbol{\lambda}}]$. This proposition also provides the way for finding a whole series of partial derivatives and also naturally yields a multi-step method of approximating the full gradient $\nabla_{\boldsymbol{\lambda}}\mathbb{E}[(G_t^{\boldsymbol{\lambda}} - \mathbb{E}[G_t])^2]$. The partial derivative in the proposition is achieved by looking 1-step into the future. We can also look more steps ahead, and get the partial

(a) True interdependency of state-λ's and state update targets

(b) Approximated interdependency of 1-step META
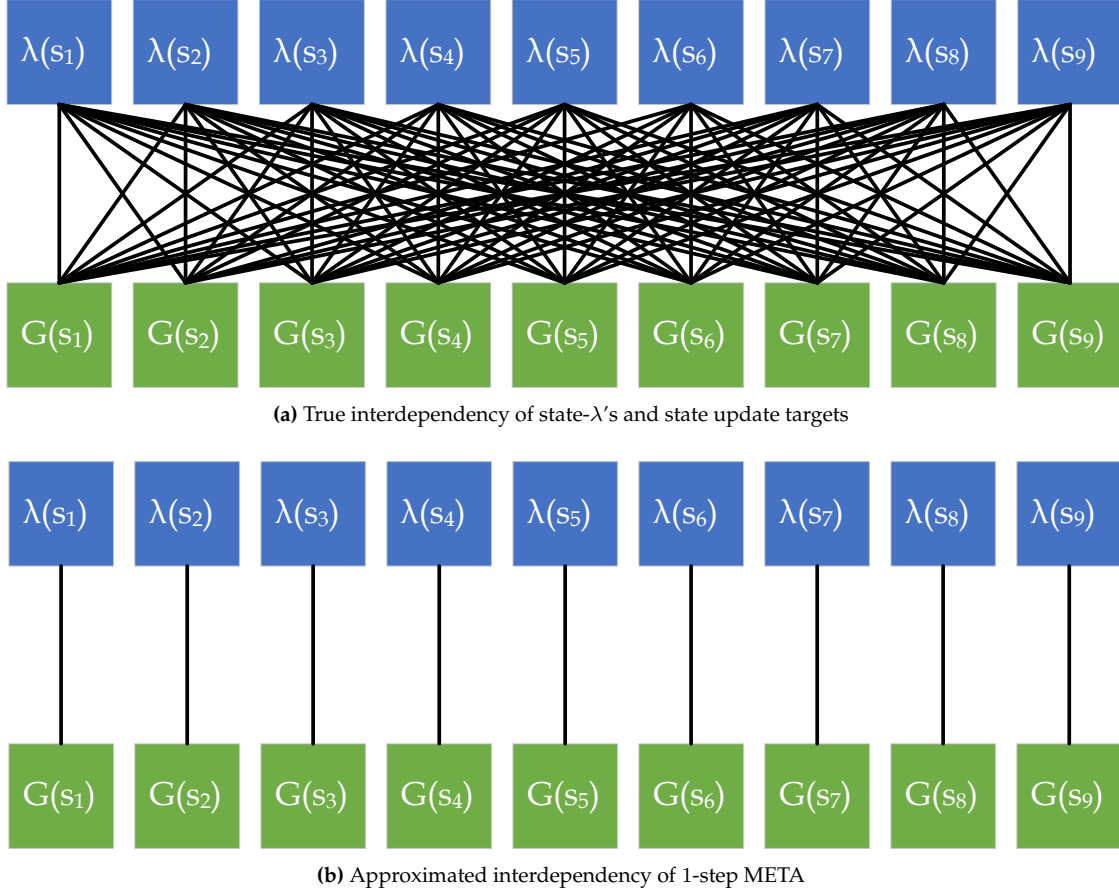
Figure 4.1: Interdependency among state-λ's and state update targets. With the increment of steps into the future, according to Proposition 4, less connections would be neglected.

derivatives *w.r.t.* $\lambda^{(t+2)}, \cdots$. These partial derivatives can be computed with the help of the auxiliary tasks as well. The more we assemble the partial derivatives, the closer we get to the full gradient. However, in our opinion, 1-step is still the most preferred not only because it can be obtained online every step without the need of buffers but also for its dominance over other dimensions of $\boldsymbol{\lambda}$: the more steps we look into the future, the more the corresponding λs of the states are more heavily discounted by the earlier γ's and λs. This result is extended naturally if we were to do the adaptations offline, in which case the partial derivatives and the gradient can be exactly computed with additional computational costs. Figure 4.1 (a) and (b) compare the interdependency of state-λ's and state update targets, under the true case and the 1-step META approximation case respectively.

It is interesting to observe that the yielded minimizer is a generalization of (3.1): the minimizer of the greedy target error can be achieved by setting $G_{t+1}^{\lambda} = G_{t+1}$. In practice, given an unknown MDP, the distributional information of the targets, *e.g.* $\mathbb{E}[G_{t+1}]$, $\mathbb{E}[G_{t+1}^{\lambda}]$ and $Var[G_{t+1}^{\lambda}]$, can only be estimated. However, such estimation has been proved viable in both offline and online settings of TD(λ) and the variants, using supervised learning and auxiliary tasks using the direct VTD method [18], respectively. This means the optimization for the "true" target error is as viable as the λ-greedy method proposed in [28], while it requires more complicated estimations than that for the "greedy" target error: we need the estimates of $\mathbb{E}[G_{t+1}]$, $\mathbb{E}[G_{t+1}^{\lambda}]$ and $Var[G_{t+1}^{\lambda}]$, while for (3.1) we only need the estimation of $\mathbb{E}[G_{t+1}]$ and $Var[G_{t+1}]$.

4.3 Trust Region Optimization

The auxiliary learners are also dependent on $\boldsymbol{\lambda}$, which brings new challenges. The optimization of the true state target error, *i.e.* the MSE between $\boldsymbol{\lambda}$-return and the true value, together with the auxiliary estimation, brings new challenges: the auxiliary estimates are learnt online and requires the stationarity of the update targets. This means if a λ for one state is changed dramatically, the auxiliary estimates of $\mathbb{E}[G_{t+1}^{\boldsymbol{\lambda}}]$ and $Var[G_{t+1}^{\boldsymbol{\lambda}}]$ will be destroyed, since they depend on each element in $\boldsymbol{\lambda}$ (whereas in λ-greedy, the pseudo targets require no $\boldsymbol{\lambda}$-controlled distributional information). If we cannot handle such challenge, either we end up with a method that have to wait for some time after each change of $\boldsymbol{\lambda}$ or we end up with λ-greedy, bearing the high bias towards the MC return and disconnection from the overall target error.

Adjusting $\boldsymbol{\lambda}$ without destroying the auxiliary estimates is a core problem. We tackle such optimization by noticing that the expectation and variance of the update targets are continuous and differentiable *w.r.t.* $\boldsymbol{\lambda}$. Thus, a small change on λ_{t+1} only yields a bounded shift of the estimates of the auxiliary tasks.
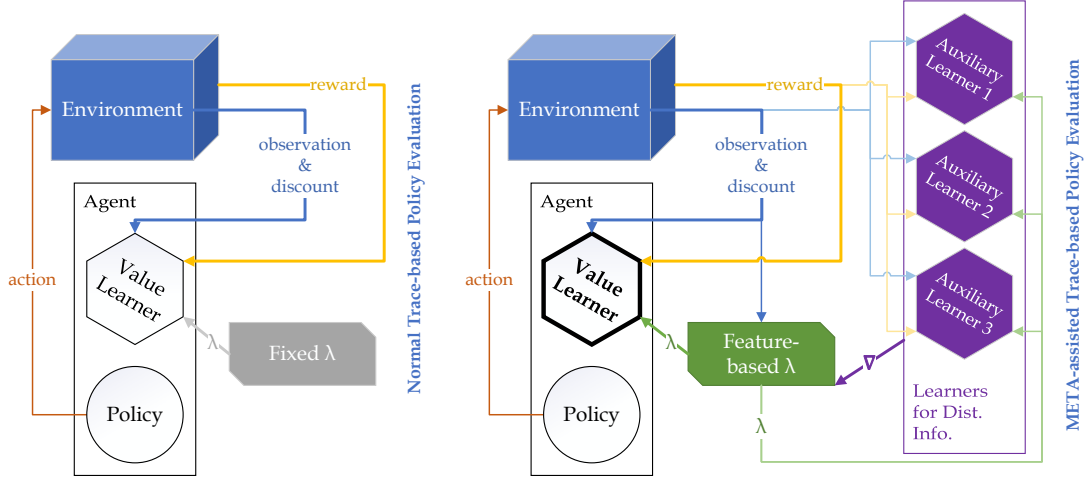
Figure 4.2: Mechanisms for META-assisted trace-based policy evaluation: the auxiliary learners learn the distributional information in parallel to the value learner and provide the approximated gradient for the adjustments of λ.

If we use small enough steps of the estimated gradients to change λ, we can stabilize the auxiliary estimates since they will not deviate far and will be corrected by the TD updates quickly. This method inherits the ideas of trust region methods used in optimizing the dynamic systems.

Combining the approximation of gradient and the decomposed one-step optimization method, we now have an online method to optimize λ to achieve approximate optimization of the overall target error, which we name as META. This method can be jointly used with value learning, serving as a plugin, to adapt λ real-time. The policy evaluation carried out by the value learner directs the value estimates toward the targets and META jointly optimizes the targets. We present the pseudocode of META in Alg. 10 and present the mechanisms in Fig. 4.2.

Algorithm 10: META-assisted Online Policy Evaluation

Initialize weights for the value learner and those for the auxiliary learners that learns $\hat{\mathbb{E}}[G_t]$, $\hat{\mathbb{E}}[G_t^{\lambda}]$ and $\hat{V}ar[G_t^{\lambda}]$

for *episodes* **do**

    $\rho_{acc} = 1$; //initialize cumulative product of importance sampling ratios

    Set traces for value learner and auxiliary learners to be **0**;

    $\boldsymbol{x}_0 = \text{initialize}(\mathcal{E})$; //Initialize the environment $\mathcal{E}$ and get the initial feature (observation) $\boldsymbol{x}_0$

    while $t \in \{0, 1, \dots\}$ *until terminated* **do**

        //INTERACT WITH ENVIRONMENT

        $a_t \sim b(\boldsymbol{x}_t)$; //sample a_t from behavior policy b

        $\rho_t = \pi(a_t, \boldsymbol{x}_t)/b(a_t, \boldsymbol{x}_t)$; $\rho_{acc} = \rho_{acc} \cdot \rho_t$; //get and accumulate importance sampling ratios

        $\boldsymbol{x}_{t+1}, \gamma_{t+1} = \text{step}(a_t)$; //take action a_t, get feature (observation) $\boldsymbol{x}_{t+1}$ and discount factor γ_{t+1}

        //AUXILIARY TASKS

        learn $\hat{\mathbb{E}}[G_t]$, $\hat{\mathbb{E}}[G_t^{\lambda}]$ and $\hat{V}ar[G_t^{\lambda}]$; //using direct VTD [18] with trace-based TD methods, *e.g.*, true online GTD($\boldsymbol{\lambda}$) [25]

        //APPROXIMATE SGD ON OVERALL TARGET ERROR

        $\lambda_{t+1} = \lambda_{t+1} - \kappa \gamma_{t+1}^2 \rho_{acc}$

        $\left[\lambda_{t+1} \left((V(\boldsymbol{x}_{t+1}) - \hat{\mathbb{E}}[G_{t+1}^{\lambda}])^2 + \hat{V}ar[G_{t+1}^{\lambda}] \right) + (\hat{\mathbb{E}}[G_{t+1}^{\lambda}] - V(\boldsymbol{x}_{t+1}))(\hat{\mathbb{E}}[G_{t+1}] - V(\boldsymbol{x}_{t+1})) \right]$;

        // change $\lambda_{t+2}, \cdots$ when using multi-step approximation of the gradient

        //LEARN VALUE

        learn $V(\boldsymbol{x}_t)$ using a trace-based TD method;

4.4 Discussions and Insights

4.4.1 Hyperparameter Search

The proposed method META trades the hyperparameter search for λ with κ. However, κ gives the algorithm the ability to have state-based λs: state or feature (observation) based $\boldsymbol{\lambda}$ can lead to better convergence compared to fixing λ for all states. Such potential may *never* be achieved by searching a fixed λ. Let us consider the tabular case, where the hyperparameter search for constant $\boldsymbol{\lambda}$ is equivalent to searching along the **1** direction inside a $|\mathcal{S}|$-dimensional box. By replacing the hyperparameter λ with κ, we extend the search direction into the whole $[0, 1]^{|\mathcal{S}|}$. The new degrees of freedom are crucial to the performance.

4.4.2 Reliance on Auxiliary Tasks

The META updates assume that $\hat{\mathbb{E}}[G_t]$, $\hat{\mathbb{E}}[G_t^\lambda]$ and $\hat{Var}[G_t^\lambda]$ can be well estimated by the auxiliary tasks. This is very similar to the idea of actor changing the policy upon the estimation of the values of the critic in the actor-critic methods. To implement this, we can add a buffer period for the estimates to be stable at the beginning of the learning process; Additionally, we should set the learning rates of the auxiliary learners higher than the value learner *s.t.* the auxiliary tasks are learnt faster, resembling the guidelines for setting learning rates of actor-critic. With the buffer period, we can also view META as approximately equivalent to offline hyperparameter search of λ, where with META we first reach a relatively stable accuracy and then adjust λ to slowly slide to fixed points with lower errors. Also, it is worth noting that META is in theory compatible with fancier settings of learning rate, since the meta-adaptation is independent of the values of the learning rate.

4.4.3 Generalization and Function Approximation

In the case of function approximation, the meta-learning of λ-greedy is still fully controlled by the value function and the two additional learned statistics but cannot directly make use of the features of the state itself. Whereas in META, we can use a parametric function of λ and performs gradient descent on it to make use of the state features. This feature is helpful for generalization and can be very effective when the state features contain rich information (good potential to be used with deep neural networks). This is to be demonstrated in the experiments.

4.4.4 From Prediction to Control

Within the control tasks where the quality of prediction is crucial to the policy improvement, it is viable to apply META to enhance the policy evaluation process. META is a trust region method, which requires the policy to be also changing smoothly, *s.t.* the shift of

values can be bounded. This constraint leads us naturally to the actor-critic architectures, where the value estimates can be used to improve a continuously changed parametric policy. We provide the pseudocode of META-assisted actor-critic control in Algorithm 11.

Algorithm 11: META-assisted Online Actor-Critic

Initialize weights for the value learner and those for the auxiliary learners that
 learns $\hat{\mathbb{E}}[G_t]$, $\hat{\mathbb{E}}[G_t^{\lambda}]$ and $\hat{V}ar[G_t^{\lambda}]$
Initialize parameterized policies $\pi(\cdot|\theta_\pi)$ and $b(\cdot|\theta_b)$;
for *episodes* **do**
 Set traces for value learner and auxiliary learners to be $\mathbf{0}$;
 $\boldsymbol{x}_0 = \text{initialize}(\mathcal{E})$;
 while $t \in \{0, 1, \dots\}$ *until terminated* **do**
  //INTERACT WITH ENVIRONMENT
  $a_t \sim b(\boldsymbol{x}_t)$; $\rho_t = \pi(a_t, \boldsymbol{x}_t)/b(a_t, \boldsymbol{x}_t)$; $\rho_{acc} = \rho_{acc} \cdot \rho_t$;
  $\boldsymbol{x}_{t+1}, \gamma_{t+1} = \text{step}(a_t)$;
  //AUXILIARY TASKS and SGD ON OVERALL TARGET ERROR
  learn $\hat{\mathbb{E}}[G_t]$, $\hat{\mathbb{E}}[G_t^{\lambda}]$ and $\hat{V}ar[G_t^{\lambda}]$;
  $\lambda_{t+1} = \lambda_{t+1} - \kappa\gamma_{t+1}^2\rho_{acc}$
  $\left[\lambda_{t+1}\left((V(\boldsymbol{x}_{t+1}) - \mathbb{E}[G_{t+1}^{\lambda}])^2 + Var[G_{t+1}^{\lambda}]\right) + (\mathbb{E}[G_{t+1}^{\lambda}] - V(\boldsymbol{x}_{t+1}))(\mathbb{E}[G_{t+1}] - V(\boldsymbol{x}_{t+1}))\right]$;
  //LEARN VALUE
  learn $V(\boldsymbol{x}_t)$ using a trace-based TD method;
  //LEARN POLICY
  One (small) step of policy gradient (actor-critic) on θ_π;

Chapter 5

Experimental Studies

In this chapter, we examine the empirical behavior of the proposed method, META, by comparing it to the baselines true online TD(λ) [26] and true online GTD(λ) [25] as well as the λ-greedy method [28][1]. For all the three sets of tests, we start adapting λs from 1, which is the same as λ-greedy [28]. This setting is enabled by using $\lambda(\boldsymbol{x}) = 1 - \boldsymbol{w}_\lambda^T \boldsymbol{x}$ as the function approximator of the parametric λ, with all the weights initialized as $\boldsymbol{0}$.

5.1   RingWorld: Tabular Case, Low Variance

This set of experiments focuses on a low-variance environment, the 11-state "ringworld" [28], in which the agent moves either left or right in a ring of states. The state transitions are deterministic and rewards only appear in the terminal states. In this set of experiments, we stick to the tabular setting and use true online TD($\boldsymbol{\lambda}$) [25] as the learner[2], for the value estimate as well as all the auxiliary estimates. As discussed in 4.4.2, for the accuracy of the auxiliary learners, we double their learning rate so that they can adapt to the changes of the estimates faster. We select 3 pairs of behavior-target policies: 1) the behavior policy goes left with 0.4 probability while the target policy does

---

[1]Implementation is open-source at `https://github.com/PwnerHarry/META`
[2]We prefer true online algorithms since they achieve the exact equivalence of the bi-directional view of λ-returns.
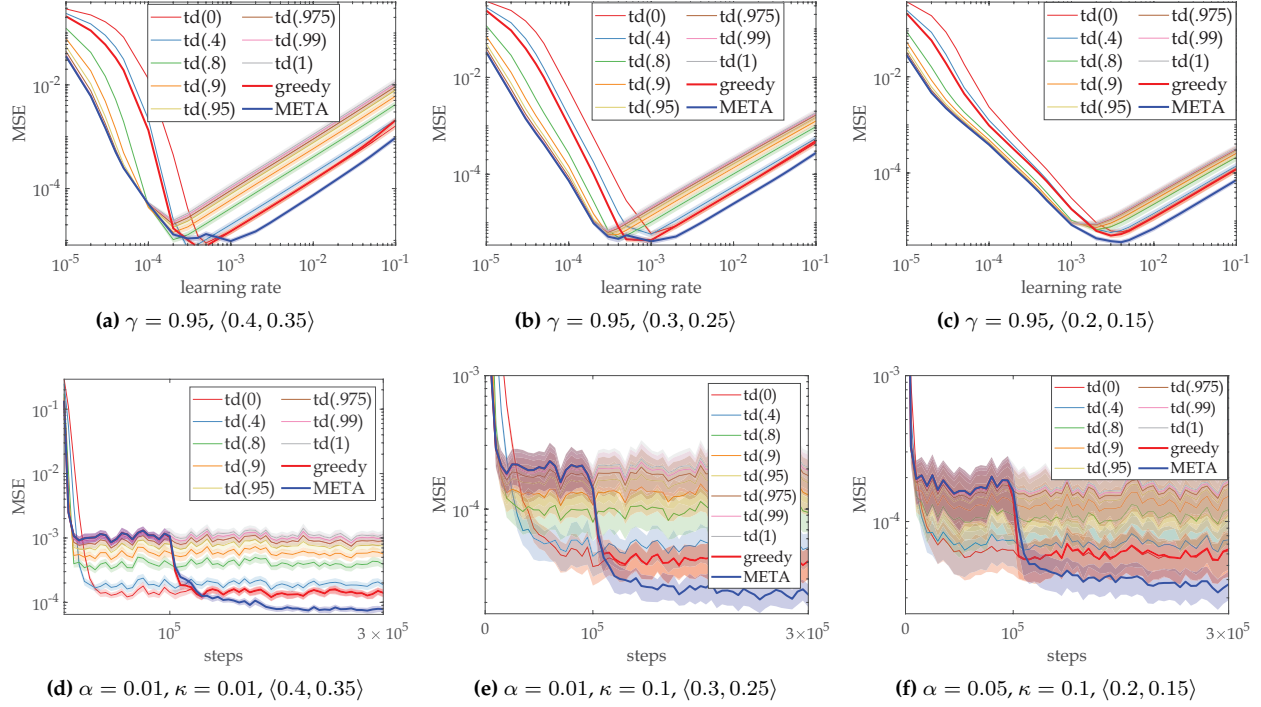
Figure 5.1: U-shaped curves and learning curves for META, λ-greedy and the baselines on RingWorld, under three pairs of behavior-target policies. For (a), (b) and (c), the x-axes represent the values of the learning rate α for prediction (or the critic), while the y-axes represent the overall value errors. Each point in the graphs contains the mean (solid) and standard deviation (shaded) collected from 240 independent runs, with 10^6 steps for prediction; For (d), (e) and (f), the x-axis represents the steps of learning and y-axis is the same as (a), (b) and (c). We choose one representative case for each U-shaped curve corresponding to different policy pairs and plot the corresponding learning curves. In these learning curves, the best known values for the hyperparameters are used. The buffer period lengths are 10^5 steps (10%). The buffer period and the adaptation period have been ticked on the x-axes. The rest of the steps for (d), (e) and (f) have been cut off since there is no significant change afterwards.

w.p. 0.35; 2) behavior 0.3 and target 0.25; 3) behavior 0.2 and target 0.15. The three pairs of policies are increasingly greedy. The baseline true online TD has 2 hyperparameters (α & λ) and so does META (α & κ), excluding those for the auxiliary learners. We test the two methods on grids of hyperparameter pairs. More specifically, for the baseline true online TD, we test on $\langle \alpha, \lambda \rangle \in \{10^{-5}, \dots, 5 \times 10^{-5}, 10^{-4}, \dots, 5 \times 10^{-4}, \dots, 5 \times 10^{-2}, 10^{-1}\} \times \{0, 0.4, 0.8, 0.9, 0.95, 0.975, 0.99, 1\}$ while for META, $\langle \alpha, \kappa \rangle \in \{10^{-5}, \dots, 5 \times 10^{-5}, 10^{-4}, \dots, 5 \times 10^{-4}, \dots, 5 \times 10^{-2}, 10^{-1}\} \times \{10^{-7}, \dots, 10^{-1}\}$. The results are presented as the U-shaped curves in Fig. 5.1. We plot the curves of the baseline under different λs and the best performance that META could get under each learning rate. The detailed results for the three pairs of policies are presented in Table 5.1, 5.2 and 5.3, respectively.

Table 5.1: Detailed Results on RingWorld (Target: 0.35, Behavior: 0.4)

baseline	True Online TD																		META	
$\alpha \backslash \lambda$	0		0.4		0.8		0.9		0.95		0.975		0.99		1		greedy			
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std	*mean*	*std*
1.0e-5	2.91e-1	2.38e-4	2.39e-1	3.72e-4	1.24e-1	5.1e-4	7.86e-2	4.68e-4	5.53e-2	4.06e-4	4.43e-2	3.61e-4	3.83e-2	3.28e-4	**3.46e-2**	3.04e-4	2.1e-1	6.06e-4	*3.56e-2*	*3.07e-4*
2.0e-5	2.23e-1	4.26e-4	1.5e-1	5.23e-4	4.23e-2	4.01e-4	1.86e-2	2.7e-4	1.05e-2	1.88e-4	7.71e-3	1.46e-4	6.49e-3	1.21e-4	**5.85e-3**	1.04e-4	1.11e-1	6.53e-4	*6.06e-3*	*1.08e-4*
3.0e-5	1.64e-1	5.34e-4	9.04e-2	5.27e-4	1.47e-2	2.52e-4	4.83e-3	1.38e-4	2.44e-3	8.4e-5	1.8e-3	5.94e-5	1.55e-3	4.6e-5	**1.44e-3**	3.79e-5	5.98e-2	5.36e-4	*1.48e-3*	*3.98e-5*
4.0e-5	1.18e-1	5.64e-4	5.35e-2	4.58e-4	5.25e-3	1.52e-4	1.41e-3	7.25e-5	7.22e-4	3.98e-5	5.71e-4	2.65e-5	5.21e-4	2.0e-5	**4.99e-4**	1.66e-5	3.33e-2	4.12e-4	*5.06e-4*	*1.73e-5*
5.0e-5	8.29e-2	5.39e-4	3.12e-2	3.69e-4	1.95e-3	9.28e-5	4.91e-4	3.93e-5	2.93e-4	2.0e-5	2.59e-4	1.36e-5	2.5e-4	1.12e-5	**2.47e-4**	1.03e-5	1.92e-2	3.12e-4	*2.48e-4*	*1.05e-5*
1.0e-4	1.3e-2	2.42e-4	2.16e-3	9.66e-6	4.68e-5	9.39e-6	**4.3e-5**	6.39e-6	4.78e-5	7.85e-6	5.07e-5	8.83e-6	5.26e-5	9.58e-6	5.4e-5	1.02e-5	1.33e-3	7.25e-5	*5.01e-5*	*7.71e-6*
2.0e-4	4.48e-4	4.44e-5	**2.56e-5**	1.18e-5	1.04e-5	6.65e-6	1.43e-5	9.43e-6	1.78e-5	1.21e-5	2.05e-5	1.42e-5	2.26e-5	1.59e-5	2.44e-5	1.73e-5	1.71e-5	5.96e-6	*1.31e-5*	*7.5e-6*
3.0e-4	4.11e-5	1.48e-5	**7.07e-6**	5.34e-6	1.28e-5	9.63e-6	1.82e-5	1.39e-5	2.34e-5	1.8e-5	2.74e-5	2.12e-5	3.07e-5	2.38e-5	3.33e-5	2.6e-5	1.02e-5	6.2e-6	*1.09e-5*	*7.48e-6*
4.0e-4	1.2e-5	7.86e-6	8.44e-6	6.18e-6	1.68e-5	1.28e-5	2.41e-5	1.84e-5	3.11e-5	2.38e-5	3.66e-5	2.82e-5	4.09e-5	3.18e-5	4.45e-5	3.47e-5	**7.4e-6**	4.9e-6	*1.12e-5*	*7.43e-6*
5.0e-4	8.95e-6	6.34e-6	1.04e-5	7.55e-6	2.1e-5	1.59e-5	3.01e-5	2.28e-5	3.89e-5	2.98e-5	4.59e-5	3.53e-5	5.14e-5	3.99e-5	5.59e-5	4.37e-5	**7.77e-6**	4.99e-6	*1.33e-5*	*8.69e-6*
1.0e-3	1.5e-5	1.01e-5	2.02e-5	1.46e-5	4.15e-5	3.11e-5	6.02e-5	4.55e-5	7.84e-5	6.03e-5	9.25e-5	7.2e-5	1.04e-4	8.14e-5	1.13e-4	8.91e-5	**1.5e-5**	1.01e-5	*9.7e-6*	*5.39e-6*
2.0e-3	2.94e-5	1.97e-5	3.98e-5	2.86e-5	8.28e-5	6.22e-5	1.2e-4	9.22e-5	1.56e-4	1.22e-4	1.84e-4	1.44e-4	2.07e-4	1.63e-4	2.25e-4	1.78e-4	**2.94e-5**	1.97e-5	*1.49e-5*	*7.36e-6*
3.0e-3	4.36e-5	2.91e-5	5.94e-5	4.24e-5	1.24e-4	9.35e-5	1.79e-4	1.38e-4	2.33e-4	1.81e-4	2.75e-4	2.15e-4	3.08e-4	2.43e-4	3.35e-4	2.65e-4	**4.38e-5**	2.93e-5	*2.19e-5*	*1.06e-5*
4.0e-3	5.78e-5	3.84e-5	7.9e-5	5.61e-5	1.65e-4	1.24e-4	2.38e-4	1.83e-4	3.09e-4	2.4e-4	3.64e-4	2.85e-4	4.09e-4	3.22e-4	4.45e-4	3.52e-4	**5.81e-5**	3.87e-5	*2.93e-5*	*1.42e-5*
5.0e-3	7.22e-5	4.77e-5	9.87e-5	6.99e-5	2.05e-4	1.55e-4	2.96e-4	2.27e-4	3.85e-4	2.98e-4	4.54e-4	3.54e-4	5.09e-4	4.0e-4	5.55e-4	4.37e-4	**7.26e-5**	4.82e-5	*3.67e-5*	*1.78e-5*
1.0e-2	1.45e-4	9.42e-5	1.97e-4	1.38e-4	4.05e-4	3.03e-4	5.86e-4	4.43e-4	7.64e-4	5.84e-4	9.03e-4	6.97e-4	1.02e-3	7.88e-4	1.11e-3	8.63e-4	**1.47e-4**	9.57e-5	*7.46e-5*	*3.62e-5*
2.0e-2	2.91e-4	1.86e-4	3.94e-4	2.73e-4	8.09e-4	5.93e-4	1.17e-3	8.79e-4	1.53e-3	1.17e-3	1.81e-3	1.41e-3	2.04e-3	1.6e-3	2.22e-3	1.75e-3	**3.01e-4**	1.95e-4	*1.55e-4*	*7.63e-5*
3.0e-2	4.41e-4	2.81e-4	5.95e-4	4.08e-4	1.22e-3	8.87e-4	1.76e-3	1.33e-3	2.3e-3	1.79e-3	2.73e-3	2.16e-3	3.07e-3	2.45e-3	3.34e-3	2.7e-3	**4.68e-4**	3.07e-4	*2.4e-4*	*1.2e-4*
4.0e-2	5.95e-4	3.8e-4	8.0e-4	5.46e-4	1.63e-3	1.19e-3	2.36e-3	1.81e-3	3.09e-3	2.44e-3	3.66e-3	2.94e-3	4.11e-3	3.35e-3	4.48e-3	3.68e-3	**6.44e-4**	4.23e-4	*3.28e-4*	*1.69e-4*
5.0e-2	7.54e-4	4.82e-4	1.01e-3	6.85e-4	2.04e-3	1.51e-3	2.97e-3	2.31e-3	3.89e-3	3.11e-3	4.6e-3	3.75e-3	5.17e-3	4.26e-3	5.63e-3	4.68e-3	**8.37e-4**	5.55e-4	*4.21e-4*	*2.22e-4*
1.0e-1	1.62e-3	1.04e-3	2.1e-3	1.45e-3	4.21e-3	3.24e-3	6.09e-3	4.89e-3	7.94e-3	6.52e-3	9.38e-3	7.79e-3	1.05e-2	8.81e-3	1.14e-2	9.65e-3	**2.08e-3**	1.45e-3	*9.58e-4*	*5.44e-4*

Color indicators are added to locate the extreme values: the bluer the better accuracy, the redder the worse. Also, the best result for each α (each row) is marked in **bold** font.

The best performance of a fine-tuned baseline can be extracted from the figure by combining the lowest points of the set of curves obtained by the baseline under different λs. But still, the fine-tuned META provides better performance, especially when the learning rate is relatively high. We can say that once META is fine-tuned, it provides significantly better performance that the baseline algorithm can possibly achieve, since it meta-learns state-based $\boldsymbol{\lambda}$s, which allow it to go beyond the scope of the optimization of the baseline. The results can also be interpreted as META having less sensitivity to the learning rate

Table 5.2: Detailed Results on RingWorld (Target: 0.25, Behavior: 0.3)

baseline	True Online TD																greedy		*META*	
	0		0.4		0.8		0.9		0.95		0.975		0.99		1					
$\alpha\backslash\lambda$	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std	*mean*	*std*
1.0e-5	3.72e-1	3.14e-4	2.78e-1	4.24e-4	1.17e-1	3.99e-4	7.06e-2	3.08e-4	4.95e-2	2.44e-4	4.02e-2	2.09e-4	3.51e-2	1.87e-4	**3.2e-2**	1.72e-4	2.39e-1	5.17e-4	*3.31e-2*	*1.75e-4*
2.0e-5	2.4e-1	4.8e-4	1.34e-1	4.54e-4	2.85e-2	1.99e-4	1.26e-2	1.15e-4	7.57e-3	7.88e-5	5.76e-3	6.35e-5	4.9e-3	5.51e-5	**4.41e-3**	5.0e-5	9.32e-2	4.14e-4	*4.55e-3*	*5.14e-5*
3.0e-5	1.46e-1	4.95e-4	6.31e-2	3.39e-4	8.73e-3	9.03e-5	3.65e-3	4.99e-5	2.25e-3	3.62e-5	1.76e-3	3.09e-5	1.52e-3	2.82e-5	**1.39e-3**	2.66e-5	3.95e-2	2.63e-4	*1.4e-3*	*2.69e-5*
4.0e-5	8.57e-2	4.19e-4	3.02e-2	2.19e-4	3.56e-3	4.74e-5	1.62e-3	2.92e-5	1.06e-3	2.34e-5	8.6e-4	2.12e-5	7.56e-4	2.0e-5	**6.93e-4**	1.93e-5	1.89e-2	1.66e-4	*6.96e-4*	*1.94e-5*
5.0e-5	5.03e-2	3.18e-4	1.53e-2	1.36e-4	1.84e-3	3.0e-5	9.09e-4	2.1e-5	6.22e-4	1.79e-5	5.11e-4	1.66e-5	4.53e-4	1.59e-5	**4.17e-4**	1.54e-5	9.98e-3	1.09e-4	*4.18e-4*	*1.55e-5*
1.0e-4	5.57e-3	6.25e-5	1.59e-3	2.46e-5	2.69e-4	1.16e-5	1.47e-4	9.44e-6	1.05e-4	8.38e-6	8.81e-5	7.88e-6	7.92e-5	7.6e-6	**7.37e-5**	7.42e-6	1.02e-3	2.38e-5	*7.38e-5*	*7.43e-6*
2.0e-4	5.71e-4	1.47e-5	1.78e-4	9.52e-6	2.75e-5	5.14e-6	1.52e-5	4.32e-6	1.18e-5	4.2e-6	1.06e-5	4.29e-6	1.01e-5	4.42e-6	**9.88e-6**	4.53e-6	1.0e-4	7.62e-6	*9.74e-6*	*4.24e-6*
3.0e-4	1.51e-4	9.27e-6	4.26e-5	6.08e-6	6.92e-6	3.59e-6	5.6e-6	3.94e-6	5.89e-6	4.59e-6	6.31e-6	5.1e-6	6.69e-6	5.48e-6	6.99e-6	5.78e-6	2.27e-5	4.33e-6	*5.15e-6*	*3.58e-6*
4.0e-4	5.8e-5	6.95e-6	1.57e-5	4.47e-6	4.97e-6	3.77e-6	5.76e-6	4.87e-6	6.84e-6	5.94e-6	7.66e-6	6.68e-6	8.27e-6	7.23e-6	8.75e-6	7.64e-6	8.13e-6	2.98e-6	*4.64e-6*	*3.65e-6*
5.0e-4	2.84e-5	5.62e-6	8.45e-6	3.74e-6	5.41e-6	4.48e-6	6.97e-6	6.05e-6	8.47e-6	7.42e-6	9.54e-6	8.37e-6	1.03e-5	9.06e-6	1.09e-5	9.58e-6	**4.52e-6**	2.42e-6	*5.5e-6*	*4.38e-6*
1.0e-3	5.93e-6	3.68e-6	5.78e-6	4.43e-6	1.03e-5	8.91e-6	1.39e-5	1.21e-5	1.71e-5	1.48e-5	1.92e-5	1.67e-5	2.08e-5	1.8e-5	2.21e-5	1.9e-5	4.35e-6	3.0e-6	*4.11e-6*	*2.65e-6*
2.0e-3	8.61e-6	6.08e-6	1.11e-5	8.87e-6	2.07e-5	1.76e-5	2.8e-5	2.37e-5	3.41e-5	2.88e-5	3.84e-5	3.24e-5	4.15e-5	3.51e-5	4.39e-5	3.71e-5	8.6e-6	6.08e-6	*5.18e-6*	*2.81e-6*
3.0e-3	1.29e-5	9.25e-6	1.68e-5	1.33e-5	3.1e-5	2.57e-5	4.17e-5	3.46e-5	5.08e-5	4.22e-5	5.72e-5	4.75e-5	6.18e-5	5.14e-5	6.54e-5	5.44e-5	1.29e-5	9.29e-6	*7.49e-6*	*3.93e-6*
4.0e-3	1.72e-5	1.24e-5	2.24e-5	1.75e-5	4.12e-5	3.36e-5	5.53e-5	4.52e-5	6.74e-5	5.53e-5	7.57e-5	6.23e-5	8.19e-5	6.75e-5	8.65e-5	7.14e-5	1.72e-5	1.24e-5	*9.84e-6*	*5.07e-6*
5.0e-3	2.15e-5	1.54e-5	2.79e-5	2.17e-5	5.13e-5	4.12e-5	6.88e-5	5.57e-5	8.38e-5	6.82e-5	9.42e-5	7.69e-5	1.02e-4	8.33e-5	1.08e-4	8.81e-5	2.16e-5	1.55e-5	*1.23e-5*	*6.28e-6*
1.0e-2	4.31e-5	2.98e-5	5.56e-5	4.08e-5	1.01e-4	7.9e-5	1.36e-4	1.08e-4	1.65e-4	1.33e-4	1.85e-4	1.5e-4	1.99e-4	1.62e-4	2.1e-4	1.72e-4	4.34e-5	3.01e-5	*2.47e-5*	*1.26e-5*
2.0e-2	8.63e-5	5.68e-5	1.11e-4	7.88e-5	2.01e-4	1.59e-4	2.67e-4	2.18e-4	3.23e-4	2.66e-4	3.61e-4	2.99e-4	3.89e-4	3.23e-4	4.1e-4	3.41e-4	8.82e-5	5.87e-5	*4.96e-5*	*2.59e-5*
3.0e-2	1.3e-4	8.5e-5	1.66e-4	1.2e-4	2.99e-4	2.43e-4	3.96e-4	3.28e-4	4.77e-4	3.98e-4	5.33e-4	4.45e-4	5.73e-4	4.8e-4	6.03e-4	5.07e-4	1.35e-4	9.14e-5	*7.47e-5*	*3.92e-5*
4.0e-2	1.74e-4	1.15e-4	2.21e-4	1.64e-4	3.95e-4	3.25e-4	5.22e-4	4.35e-4	6.28e-4	5.25e-4	7.01e-4	5.88e-4	7.53e-4	6.34e-4	7.92e-4	6.69e-4	1.84e-4	1.27e-4	*1.01e-4*	*5.27e-5*
5.0e-2	2.19e-4	1.47e-4	2.76e-4	2.09e-4	4.91e-4	4.06e-4	6.47e-4	5.4e-4	7.77e-4	6.52e-4	8.66e-4	7.3e-4	9.3e-4	7.87e-4	9.78e-4	8.32e-4	2.32e-4	1.64e-4	*1.28e-4*	*6.74e-5*
1.0e-1	4.51e-4	3.18e-4	5.55e-4	4.31e-4	9.62e-4	7.98e-4	1.26e-3	1.06e-3	1.51e-3	1.29e-3	1.67e-3	1.47e-3	1.8e-3	1.6e-3	1.89e-3	1.7e-3	4.89e-4	3.65e-4	*2.77e-4*	*1.49e-4*

Color indicators are added to locate the extreme values: the bluer the better accuracy, the redder the worse. Also, the best result for each α (each row) is marked in **bold** font.

Table 5.3: Detailed Results on RingWorld (Target: 0.15, Behavior: 0.2)

baseline	True Online TD																greedy		*META*	
	0		0.4		0.8		0.9		0.95		0.975		0.99		1					
$\alpha\backslash\lambda$	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std	*mean*	*std*
1.0e-5	3.65e-1	2.65e-4	2.43e-1	3.15e-4	8.72e-2	2.12e-4	5.3e-2	1.48e-4	3.89e-2	1.14e-4	3.28e-2	9.74e-5	2.94e-2	8.77e-5	2.74e-2	8.15e-5	2.09e-1	3.22e-4	*2.83e-2*	*8.32e-5*
2.0e-5	1.87e-1	3.37e-4	8.42e-2	2.41e-4	1.6e-2	6.53e-5	8.52e-3	3.93e-5	6.17e-3	3.27e-5	5.29e-3	3.07e-5	4.85e-3	2.99e-5	4.58e-3	2.94e-5	5.72e-2	1.81e-4	*4.65e-3*	*2.95e-5*
3.0e-5	8.51e-2	2.67e-4	2.92e-2	1.21e-4	5.3e-3	2.76e-5	3.35e-3	2.33e-5	2.71e-3	2.22e-5	2.45e-3	2.17e-5	2.3e-3	2.14e-5	2.22e-3	2.12e-5	1.92e-2	8.72e-5	*2.22e-3*	*2.12e-5*
4.0e-5	3.8e-2	1.63e-4	1.18e-2	5.48e-5	2.8e-3	1.92e-5	1.99e-3	1.75e-5	1.68e-3	1.67e-5	1.55e-3	1.64e-5	1.47e-3	1.62e-5	1.43e-3	1.61e-5	8.33e-3	4.64e-5	*1.43e-3*	*1.61e-5*
5.0e-5	1.82e-2	9.0e-5	5.94e-3	2.87e-5	1.84e-3	1.51e-5	1.38e-3	1.41e-5	1.19e-3	1.36e-5	1.11e-3	1.35e-5	1.06e-3	1.34e-5	1.03e-3	1.33e-5	4.43e-3	2.82e-5	*1.03e-3*	*1.33e-5*
1.0e-4	2.35e-3	1.33e-5	1.25e-3	1.03e-5	6.05e-4	9.11e-6	4.88e-4	8.72e-6	4.36e-4	8.54e-6	4.11e-4	8.46e-6	3.96e-4	8.42e-6	3.86e-4	8.39e-6	9.71e-4	1.05e-5	*3.87e-4*	*8.39e-6*
2.0e-4	6.14e-4	7.11e-6	3.7e-4	5.75e-6	1.91e-4	4.66e-6	1.56e-4	4.48e-6	1.4e-4	4.42e-6	1.32e-4	4.4e-6	1.28e-4	4.39e-6	1.25e-4	4.38e-6	3.09e-4	5.63e-6	*1.25e-4*	*4.38e-6*
3.0e-4	2.95e-4	4.43e-6	1.82e-4	3.8e-6	9.74e-5	3.45e-6	8.07e-5	3.4e-6	7.31e-5	3.39e-6	6.95e-5	3.39e-6	6.74e-5	3.4e-6	6.6e-5	3.4e-6	1.61e-4	3.97e-6	*6.6e-5*	*3.4e-6*
4.0e-4	1.78e-4	3.43e-6	1.11e-4	3.19e-6	5.95e-5	2.95e-6	4.95e-5	2.93e-6	4.49e-5	2.94e-6	4.28e-5	2.96e-6	4.16e-5	2.98e-6	4.07e-5	2.99e-6	9.96e-5	3.24e-6	*4.07e-5*	*2.99e-6*
5.0e-4	1.19e-4	3.02e-6	7.37e-5	2.82e-6	3.92e-5	2.6e-6	3.25e-5	2.62e-6	2.96e-5	2.68e-6	2.82e-5	2.73e-6	2.74e-5	2.77e-6	2.69e-5	2.8e-6	6.71e-5	2.77e-6	*2.69e-5*	*2.77e-6*
1.0e-3	2.99e-5	1.93e-6	1.8e-5	1.87e-6	1.03e-5	2.44e-6	9.37e-6	2.91e-6	9.11e-6	3.26e-6	9.06e-6	3.48e-6	9.06e-6	3.64e-6	9.07e-6	3.75e-6	1.77e-5	1.69e-6	*8.32e-6*	*2.95e-6*
2.0e-3	9.21e-6	2.13e-6	6.72e-6	2.66e-6	6.59e-6	4.43e-6	7.35e-6	5.46e-6	8.01e-6	6.19e-6	8.45e-6	6.64e-6	8.76e-6	6.95e-6	8.99e-6	7.18e-6	6.24e-6	2.04e-6	*4.37e-6*	*2.35e-6*
3.0e-3	6.24e-6	2.94e-6	5.77e-6	3.87e-6	7.6e-6	6.54e-6	9.06e-6	8.05e-6	1.02e-5	9.14e-6	1.09e-5	9.82e-6	1.14e-5	1.03e-5	1.18e-5	1.06e-5	4.98e-6	2.89e-6	*3.82e-6*	*1.86e-6*
4.0e-3	5.92e-6	3.84e-6	6.37e-6	5.11e-6	9.41e-6	8.61e-6	1.14e-5	1.06e-5	1.3e-5	1.21e-5	1.4e-5	1.3e-5	1.46e-5	1.36e-5	1.51e-5	1.4e-5	5.35e-6	3.82e-6	*3.63e-6*	*2.02e-6*
5.0e-3	6.51e-6	4.77e-6	7.48e-6	6.34e-6	1.15e-5	1.06e-5	1.41e-5	1.31e-5	1.6e-5	1.49e-5	1.72e-5	1.61e-5	1.8e-5	1.69e-5	1.87e-5	1.74e-5	6.24e-6	4.77e-6	*4.01e-6*	*2.32e-6*
1.0e-2	1.2e-5	9.39e-6	1.43e-5	1.23e-5	2.24e-5	2.06e-5	2.75e-5	2.57e-5	3.13e-5	2.94e-5	3.37e-5	3.17e-5	3.54e-5	3.33e-5	3.66e-5	3.45e-5	1.2e-5	9.45e-6	*6.99e-6*	*4.09e-6*
2.0e-2	2.37e-5	1.83e-5	2.83e-5	2.4e-5	4.42e-5	4.08e-5	5.42e-5	5.07e-5	6.17e-5	5.79e-5	6.63e-5	6.24e-5	6.95e-5	6.54e-5	7.18e-5	6.76e-5	2.39e-5	1.85e-5	*1.38e-5*	*8.21e-6*
3.0e-2	3.54e-5	2.71e-5	4.25e-5	3.61e-5	6.6e-5	6.07e-5	8.07e-5	7.48e-5	9.15e-5	8.51e-5	9.83e-5	9.14e-5	1.03e-4	9.58e-5	1.06e-4	9.9e-5	3.58e-5	2.77e-5	*2.08e-5*	*1.23e-5*
4.0e-2	4.73e-5	3.62e-5	5.67e-5	4.81e-5	8.77e-5	7.98e-5	1.07e-4	9.81e-5	1.21e-4	1.12e-4	1.3e-4	1.2e-4	1.36e-4	1.26e-4	1.4e-4	1.3e-4	4.8e-5	3.73e-5	*2.79e-5*	*1.64e-5*
5.0e-2	5.93e-5	4.54e-5	7.08e-5	5.97e-5	1.09e-4	9.83e-5	1.33e-4	1.21e-4	1.5e-4	1.38e-4	1.61e-4	1.48e-4	1.68e-4	1.56e-4	1.73e-4	1.61e-4	6.02e-5	4.69e-5	*3.5e-5*	*2.05e-5*
1.0e-1	1.19e-4	8.82e-5	1.4e-4	1.13e-4	2.13e-4	1.89e-4	2.57e-4	2.34e-4	2.89e-4	2.66e-4	3.08e-4	2.85e-4	3.21e-4	2.98e-4	3.31e-4	3.08e-4	1.2e-4	9.19e-5	*7.12e-5*	*4.02e-5*

Color indicators are added to locate the extreme values: the bluer the better accuracy, the redder the worse. Also, the best result for each α (each row) is marked in **bold** font.

(a) FrozenLake, $\gamma = 0.95$

(b) FrozenLake, $\alpha = \beta = 0.0001$, $\kappa = 10^{-6}$, $\kappa_{\mathrm{np}} = 10^{-4}$

Figure 5.2: U-shaped curves and learning curves for META, λ-greedy and the baselines on FrozenLake. For (a), the x-axis represents the values of the learning rate α for prediction, while the y-axis represents the overall value error. Each point in the graph contains the mean (solid) and standard deviation (shaded) collected from 240 independent runs, with 10^6 steps for prediction; For (d), the x-axis represents the steps of learning and y-axis is the same as (a). In the learning curves, the best known values for the hyperparameters are used. The buffer period lengths are 10^5 steps (10%). The buffer period and the adaptation period have been ticked on the x-axes.

hyperparameter than the baseline true online TD. It is also interesting to notice that the greedier the policies, the larger the performance boost that META can provide.

5.2 FrozenLake: Linear Function Approximation with High Variance

This set of experiments is carried out on a higher variance environment, the "4×4" FrozenLake, in which the transitions are stochastic and the agent seeks to fetch a frisbee back on a frozen lake surface with holes from the northwest to the southeast. We craft an exploratory policy that takes the actions with equal probability, and a heuristic policy that has 0.3 probability for going south or east, 0.2 for going north or west. This

Table 5.4: Detailed Results on FrozenLake

| | baseline | | True Online GTD | | | | | | | | | | | | | | greedy | | META(np) | | META | |
| | 0 | | 0.4 | | 0.8 | | 0.9 | | 0.95 | | 0.975 | | 0.99 | | 1 | | | | | | | |
$\alpha\backslash\lambda$	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
1.0e-5	2.14e-3	1.11e-5	1.85e-3	1.63e-5	1.38e-3	1.91e-5	1.26e-3	1.72e-5	1.21e-3	1.55e-5	1.18e-3	1.45e-5	1.17e-3	1.39e-5	1.16e-3	1.35e-5	1.92e-3	2.09e-5	1.16e-3	1.35e-5	**1.16e-3**	1.35e-5
2.0e-5	1.93e-3	1.57e-5	1.55e-3	2.15e-5	1.1e-3	2.18e-5	1.01e-3	1.93e-5	9.67e-4	1.78e-5	9.48e-4	1.72e-5	9.38e-4	1.7e-5	9.32e-4	1.7e-5	1.62e-3	2.84e-5	9.32e-4	1.7e-5	9.33e-4	1.7e-5
3.0e-5	1.76e-3	1.88e-5	1.35e-3	2.45e-5	9.28e-4	2.34e-5	8.42e-4	2.1e-5	8.05e-4	1.98e-5	7.88e-4	1.95e-5	7.79e-4	1.97e-5	7.73e-4	2.01e-5	1.41e-3	3.18e-5	**7.73e-4**	2.01e-5	7.74e-4	2.0e-5
4.0e-5	1.63e-3	2.12e-5	1.2e-3	2.64e-5	7.97e-4	2.43e-5	7.17e-4	2.19e-5	6.82e-4	2.1e-5	6.67e-4	2.12e-5	6.59e-4	2.19e-5	6.53e-4	2.26e-5	1.25e-3	3.35e-5	6.54e-4	2.26e-5	6.54e-4	2.24e-5
5.0e-5	1.51e-3	2.3e-5	1.07e-3	2.78e-5	6.93e-4	2.48e-5	6.18e-4	2.24e-5	5.86e-4	2.18e-5	5.72e-4	2.26e-5	5.65e-4	2.37e-5	5.6e-4	2.49e-5	1.12e-3	3.42e-5	**5.6e-4**	2.49e-5	5.61e-4	2.45e-5
1.0e-4	1.13e-3	2.84e-5	6.98e-4	3.07e-5	3.91e-4	2.4e-5	3.39e-4	2.24e-5	3.2e-4	2.55e-5	3.13e-4	2.99e-5	3.11e-4	3.41e-5	3.1e-4	3.79e-5	7.76e-4	3.45e-5	3.1e-4	3.43e-5	**3.09e-4**	2.94e-5
2.0e-4	7.85e-4	3.4e-5	3.99e-4	3.12e-5	2.0e-4	2.11e-5	1.83e-4	2.84e-5	1.84e-4	4.2e-5	1.89e-4	5.45e-5	1.94e-4	6.51e-5	1.99e-4	7.42e-5	5.25e-4	3.39e-5	1.81e-4	3.32e-5	**1.81e-4**	2.04e-5
3.0e-4	6.18e-4	3.71e-5	2.78e-4	2.93e-5	1.58e-4	2.35e-5	1.6e-4	4.15e-5	1.73e-4	6.51e-5	1.85e-4	8.57e-5	1.95e-4	1.03e-4	2.04e-4	1.18e-4	4.01e-4	3.29e-5	1.64e-4	5.08e-5	**1.5e-4**	2.42e-5
4.0e-4	5.06e-4	3.85e-5	2.2e-4	2.69e-5	1.5e-4	2.91e-5	1.64e-4	5.68e-5	1.84e-4	9.08e-5	2.02e-4	1.2e-4	2.17e-4	1.46e-4	2.29e-4	1.68e-4	3.23e-4	3.12e-5	1.58e-4	1.41e-5	**1.47e-4**	3.06e-5
5.0e-4	4.23e-4	3.87e-5	1.89e-4	2.48e-5	1.51e-4	3.61e-5	1.73e-4	7.31e-5	2.0e-4	1.18e-4	2.23e-4	1.58e-4	2.42e-4	1.92e-4	2.57e-4	2.21e-4	2.73e-4	2.94e-5	**1.49e-4**	1.59e-5	1.51e-4	3.78e-5
1.0e-3	2.3e-4	3.29e-5	1.54e-4	2.2e-5	1.78e-4	7.43e-5	2.29e-4	1.59e-4	2.86e-4	2.66e-4	3.33e-4	3.63e-4	3.72e-4	4.48e-4	4.03e-4	5.21e-4	1.95e-4	2.56e-5	**1.54e-4**	2.9e-5	1.57e-4	1.97e-5
2.0e-3	1.74e-4	2.69e-5	1.63e-4	3.3e-5	2.38e-4	1.53e-4	3.4e-4	3.43e-4	4.55e-4	5.89e-4	5.49e-4	8.16e-4	6.26e-4	1.02e-3	6.89e-4	1.19e-3	1.92e-4	3.39e-5	**1.61e-4**	1.82e-5	1.64e-4	2.81e-5
3.0e-3	1.72e-4	2.9e-5	1.77e-4	4.63e-5	2.96e-4	2.37e-4	4.49e-4	5.43e-4	6.18e-4	9.38e-4	7.57e-4	1.3e-3	8.7e-4	1.61e-3	9.63e-4	1.88e-3	2.04e-4	4.56e-5	**1.67e-4**	2.41e-5	1.76e-4	2.81e-5
4.0e-3	1.76e-4	3.31e-5	1.91e-4	5.98e-5	3.54e-4	3.26e-4	5.56e-4	7.53e-4	7.78e-4	1.3e-3	9.58e-4	1.79e-3	1.1e-3	2.21e-3	1.22e-3	2.57e-3	2.17e-4	5.7e-5	**1.73e-4**	2.97e-5	1.76e-4	3.28e-5
5.0e-3	1.81e-4	3.75e-5	2.06e-4	7.33e-5	4.11e-4	4.17e-4	6.6e-4	9.68e-4	9.32e-4	1.66e-3	1.15e-3	2.27e-3	1.33e-3	2.79e-3	1.47e-3	3.24e-3	2.3e-4	6.83e-5	**1.8e-4**	3.5e-5	1.81e-4	3.73e-5
1.0e-2	2.08e-4	5.91e-5	2.81e-4	1.41e-4	6.89e-4	8.84e-4	1.15e-3	2.05e-3	1.62e-3	3.41e-3	1.99e-3	4.54e-3	2.27e-3	5.44e-3	2.5e-3	6.17e-3	3.06e-4	1.3e-4	2.09e-4	5.97e-5	**2.08e-4**	5.91e-5
2.0e-2	2.67e-4	1.07e-4	4.33e-4	2.82e-4	1.19e-3	1.84e-3	1.97e-3	4.14e-3	2.73e-3	6.67e-3	3.28e-3	8.61e-3	3.7e-3	1.01e-2	4.02e-3	1.12e-2	5.17e-4	3.6e-4	2.68e-4	1.07e-4	**2.67e-4**	1.07e-4
3.0e-2	3.31e-4	1.71e-4	5.85e-4	4.37e-4	1.63e-3	2.81e-3	2.66e-3	6.15e-3	3.63e-3	9.67e-3	4.34e-3	1.23e-2	4.86e-3	1.42e-2	5.26e-3	1.57e-2	8.09e-4	9.22e-4	3.32e-4	1.73e-4	**3.31e-4**	1.71e-4
4.0e-2	3.99e-4	2.54e-4	7.32e-4	6.05e-4	2.03e-3	3.8e-3	3.27e-3	8.09e-3	4.44e-3	1.25e-2	5.28e-3	1.57e-2	5.9e-3	1.81e-2	6.38e-3	1.99e-2	1.17e-3	1.94e-3	4.01e-4	2.59e-4	**3.99e-4**	2.54e-4
5.0e-2	4.71e-4	3.54e-4	8.76e-4	7.84e-4	2.4e-3	4.78e-3	3.83e-3	9.98e-3	5.18e-3	1.52e-2	6.15e-3	1.9e-2	6.87e-3	2.18e-2	7.42e-3	2.4e-2	1.3e-3	1.8e-3	4.71e-4	3.54e-4	**4.71e-4**	3.54e-4
1.0e-1	8.97e-4	1.06e-3	1.52e-3	1.82e-3	3.85e-3	9.39e-3	6.11e-3	1.84e-2	8.26e-3	2.74e-2	9.8e-3	3.38e-2	1.09e-2	3.87e-2	1.18e-2	4.24e-2	2.59e-3	4.99e-3	8.97e-4	1.06e-3	**8.97e-4**	1.06e-3

Color indicators are added to locate the extreme values: the bluer the better accuracy, the redder the worse. Also, the best result for each α (each row) is marked in **bold** font.

time we use the linear function approximation and true online GTD(λ) with discrete tile coding (4 tiles, 4 randomly offset tilings). We set the second learning rate β introduced in true online GTD(λ) to be the same as α (for the value learners as well as the auxiliary learners in all the compared algorithms). Additionally, we remove the parametric setting of λ to get a method that we call "META(np)", and which demonstrates the potential of a parametric feature (observation) based λ. The U-shaped curves, obtained using the exact same settings as in the ringworld tests, are provided in Fig. 5.2. The detailed results are provided in Table 5.4.

We observe similar patterns with the first set of experiments. Comparing META with "META(np)", the generalization provided by the parametric λ is beneficial: in panel (b) we observe generally better performance and in panel (e) we see that a parametric λ has better sample efficiency. This would suggest that using parametric λ in environments with relatively smooth dynamics would be generally beneficial for sample efficiency.

(a) MountainCar, $\gamma = 1, \eta = 1$ **(b)** MountainCar, $\alpha = \beta = 10^{-5}, \eta = 1, \kappa = 10^{-5}$
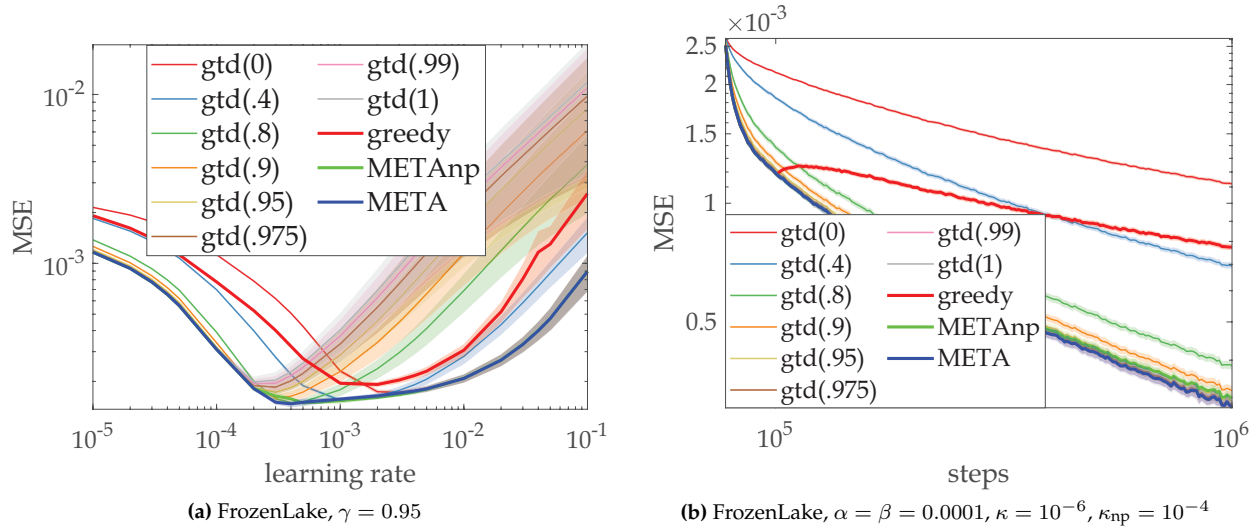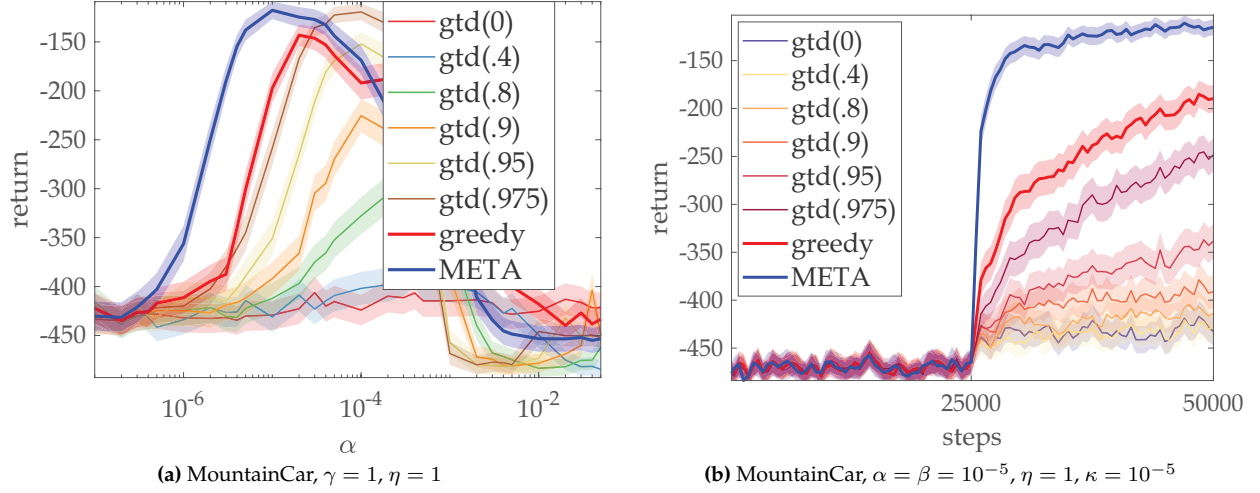
Figure 5.3: U-shaped curves and learning curves for META, λ-greedy and baselines on MountainCar. For (a), the x-axis represents the values of the learning rate α for prediction (the critic), while the y-axis represents the discounted return for MountainCar. Each point in the graph contains the mean (solid) and standard deviation (shaded) collected from 240 independent runs, 50000 steps for control; For (d), the x-axis represents the steps of learning and y-axis is the same as (a). In the learning curves, the best known values for the hyperparameters are used. The buffer period length is 25000 steps (50%). The buffer period and the adaptation period have been ticked on the x-axes. Note that in the buffer period of control, we also freeze the policy.

5.3   MountainCar: On-Policy Actor-Critic Control with Linear Function Approximation

In this set of experiments we investigate the case in which META assists on-policy actor-critic control on a modified version of the environment MountainCar with noise added to the state transitions. The state features used in these experiments resemble the classical setting in [20], with the tile encoding of 8 tilings and 8 offsets. We use a softmax policy parameterized by a $|A| \times D$ matrix, where D is the dimension of the state features, with true online GTD(λ) as the learners (critics). This time, the U-shaped curves presented in Fig. 5.3 show performance better than the baselines, and significantly better than λ-greedy assisted actor-critic.

In this set of experiments we intentionally set the step size of the gradient ascent of the policy to be high ($\eta = 1$), in order to emphasise the quality of policy evaluation. However, typically in actor-critic we keep η small. If η is small, META is expected to help a lot less, as the accuracy of the value function would be less important. Enhancing the policy evaluation quality may not be sufficient for increasing the sample efficiency of control problems.

From the curves presented in Figure 5.3, the most significant improvements are obtained when the learning rate of the critic is small. Typically in actor-critic, we set the learning rate of the critic to be higher than the actor, in order to improve the quality of the update of the actor. META alleviates the requirement for such a setting by boosting the sample efficiency of the critic.

Chapter 6

Conclusion and Future Work

6.1   Summary of Contributions

In this thesis, we provided the following original contributions:

1. The surrogate approach of optimizing sample efficiency by the optimization of update targets.

2. A principled approach, META, which achieves approximate meta-optimization of the sample efficiency of policy evaluation while remaining compatible with online updating, off-policy learning, function approximation and control, with minimal additional computational cost.

3. Identification of the distribution mismatch between the "on-policy" distribution and the assumed "stationary" distribution.

We demonstrated the merits of the proposed approach in several experiments.

Beyond this, several smaller contributions are also presented:

1. A re-framing of the basics of RL using state-based discounting setting.

2. Identification of the distribution mismatch between the "on-policy" distribution with the assumed "stationary" distribution in the episodic setting.

3. A new proof of the policy gradient theorem under state-based discount setting.

6.2   Future Work

The contribution of this thesis points out several promising directions worthy of research in the future:

1. Investigate the possibilities of using traces to accumulate one-step meta-gradient updates, in order to achieve better approximation of the true gradients: the approximation of the per-state meta-gradients can be ameliorated by looking more steps into the future, which draws great similarity to the online accumulation of the eligibility traces. Perhaps better meta-gradients can be achieved by utilizing traces.

2. Investigate the convergence properties of the update targets under meta-optimization: under certain assumptions, we proved that the meta-gradient updates can optimize the overall targets. But do these targets converge? Or, what dynamic patterns do these targets follow?

3. Investigate formally the relationship between update targets and sample efficiency: optimization of update targets, as a surrogate of sample efficiency, greatly simplifies the analyses and exhibits significant empirical performance boost. Yet, we have not formally established how the quality of the update targets influences the sample efficiency.

4. Investigate better ways of combining the proposed approach with actor-critic methods: the trust regions required by the auxiliary learners already limit the learning rate of the critic to be small. Together with the fact that we need to limit the learning rate of the actor to be smaller than the critic, this further limits the learning speed of actor-critic systems. We would like to find a more efficient algorithmic approach than the one we used in this thesis.

Bibliography

[1] BARNARD, E. Temporal-difference methods and markov models. *IEEE Transactions on Systems, Man, and Cybernetics 23*, 2 (March 1993), 357–365.

[2] BERTSEKAS, D. P., BERTSEKAS, D. P., BERTSEKAS, D. P., AND BERTSEKAS, D. P. *Dynamic programming and optimal control*, vol. 1. Athena scientific Belmont, MA, 1995.

[3] BERTSEKAS, D. P., AND TSITSIKLIS, J. N. *Neuro-dynamic programming*, vol. 5. Athena Scientific Belmont, MA, 1996.

[4] BROCKMAN, G., CHEUNG, V., PETTERSSON, L., SCHNEIDER, J., SCHULMAN, J., TANG, J., AND ZAREMBA, W. Openai gym, 2016.

[5] DABNEY, W., AND BARTO, A. Adaptive step-size for online temporal difference learning, 2012.

[6] DAYAN, P., AND SEJNOWSKI, T. J. Td(λ) converges with probability 1. *Machine Learning 14*, 3 (1994), 295–301.

[7] DOWNEY, C., AND SANNER, S. Temporal difference bayesian model averaging: A bayesian perspective on adapting lambda. In *ICML* (2010), pp. 311–318.

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

[9] KEARNS, M. J., AND SINGH, S. P. Bias-variance error bounds for temporal difference updates. In *Conference on Computational Learning Theory* (2000), COLT '00, pp. 142–147.

[10] KONIDARIS, G., NIEKUM, S., AND THOMAS, P. S. Td$_\gamma$: Re-evaluating complex backups in temporal difference learning. In *Proceedings of the 24th International Conference on Neural Information Processing Systems* (Red Hook, NY, USA, 2011), NIPS'11, Curran Associates Inc., p. 2402–2410.

[11] MAEI, H. R., AND SUTTON, R. S. Gq(lambda): A general gradient algorithm for temporal-difference prediction learning with eligibility traces. In *3d Conference on Artificial General Intelligence (AGI-2010)* (2010/06), Atlantis Press.

[12] MANN, T. A., PENEDONES, H., MANNOR, S., AND HESTER, T. Adaptive lambda least-squares temporal difference learning. *CoRR abs/1612.09465* (2016).

[13] MNIH, V., KAVUKCUOGLU, K., SILVER, D., RUSU, A. A., VENESS, J., BELLEMARE, M. G., GRAVES, A., RIEDMILLER, M., FIDJELAND, A. K., OSTROVSKI, G., ET AL. Human-level control through deep reinforcement learning. *Nature 518*, 7540 (2015), 529.

[14] PUTERMAN, M. L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[15] RAMSTEDT, S., AND PAL, C. Real-time reinforcement learning. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 3067–3076.

[16] RIQUELME, C., PENEDONES, H., VINCENT, D., MAENNEL, H., GELLY, S., MANN, T. A., BARRETO, A., AND NEU, G. Adaptive temporal-difference learning for policy evaluation with per-state uncertainty estimates. In *Advances in Neural Information*

Processing Systems 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 11872–11882.

[17] SCHAPIRE, R. E., AND WARMUTH, M. K. On the worst-case analysis of temporal-difference learning algorithms. *Machine Learning 22*, 1 (1996), 95–121.

[18] SHERSTAN, C., BENNETT, B., YOUNG, K., ASHLEY, D., WHITE, A., WHITE, M., AND SUTTON, R. Directly estimating the variance of the λ-return using temporal-difference methods. *arXiv abs/1801.08287* (2018).

[19] SINGH, S., AND DAYAN, P. Analytical mean squared error curves for temporal difference learning. *Machine Learning 32*, 1 (1998), 5–40.

[20] SUTTON, R., AND BARTO, A. *Reinforcement learning - An Introduction*. MIT Press, 2018.

[21] SUTTON, R., MAHMOOD, A. R., PRECUP, D., AND HASSELT, H. A new Q(λ) with interim forward view and monte carlo equivalence. In *International Conference on Machine Learning* (2014), vol. 32 of *Proceedings of Machine Learning Research*, pp. 568–576.

[22] SUTTON, R., AND SINGH, S. P. On step-size and bias in temporal-difference learning. In *Center for Systems Science, Yale University* (1994), pp. 91–96.

[23] SUTTON, R. S. Td models: Modeling the world at a mixture of time scales. In *Machine Learning Proceedings 1995*, A. Prieditis and S. Russell, Eds. Morgan Kaufmann, San Francisco (CA), 1995, pp. 531 – 539.

[24] SUTTON, R. S., MAHMOOD, A. R., AND WHITE, M. An emphatic approach to the problem of off-policy temporal-difference learning. *CoRR abs/1503.04269* (2015).

[25] VAN HASSELT, H., MAHMOOD, A. R., AND SUTTON, R. Off-policy TD(λ) with a true online equivalence. In *Conference on Uncertainty in Artificial Intelligence* (2014), pp. 330–339.

[26] VAN SEIJEN, H., MAHMOOD, A. R., PILARSKI, P. M., MACHADO, M. C., AND SUTTON, R. S. True online temporal-difference learning. *Journal of Machine Learning Research 17*, 1 (2016), 5057–5096.

[27] WHITE, M. Unifying task specification in reinforcement learning. *CoRR abs/1609.01995* (2016).

[28] WHITE, M., AND WHITE, A. A greedy approach to adapting the trace parameter for temporal difference learning. In *International Conference on Autonomous Agents and Multiagent Systems* (2016), AAMAS '16, International Foundation for Autonomous Agents and Multiagent Systems, pp. 557–565.

[29] XU, Z., VAN HASSELT, H., AND SILVER, D. Meta-gradient reinforcement learning. *CoRR abs/1805.09801* (2018).

[30] YU, H. Least squares temporal difference methods: An analysis under general conditions. *SIAM Journal on Control and Optimization 50*, 6 (2012), 3310–3343.

[31] ZHAO, M., LUAN, S., PORADA, I., CHANG, X.-W., AND PRECUP, D. META-learning state-based λ for more sample-efficient policy evaluation. *ArXiv abs/1904.11439* (2020).

79

[26] VAN SEIJEN, H., MAHMOOD, A. R., PILARSKI, P. M., MACHADO, M. C., AND SUTTON, R. S. True online temporal-difference learning. *Journal of Machine Learning Research 17*, 1 (2016), 5057–5096.

[27] WHITE, M. Unifying task specification in reinforcement learning. *CoRR abs/1609.01995* (2016).

[28] WHITE, M., AND WHITE, A. A greedy approach to adapting the trace parameter for temporal difference learning. In *International Conference on Autonomous Agents and Multiagent Systems* (2016), AAMAS '16, International Foundation for Autonomous Agents and Multiagent Systems, pp. 557–565.

[29] XU, Z., VAN HASSELT, H., AND SILVER, D. Meta-gradient reinforcement learning. *CoRR abs/1805.09801* (2018).

[30] YU, H. Least squares temporal difference methods: An analysis under general conditions. *SIAM Journal on Control and Optimization 50*, 6 (2012), 3310–3343.

[31] ZHAO, M., LUAN, S., PORADA, I., CHANG, X.-W., AND PRECUP, D. META-learning state-based λ for more sample-efficient policy evaluation. *ArXiv abs/1904.11439* (2020).

APPENDICES

In this chapter, the assistive details of the thesis will be provided.

6.3 Technical Auxiliaries for Experiments

6.3.1 Environments

RingWorld

The RingWorld domain was introduced as a suitable domain for investigating λ in [9]. It describes an environment a ring of states. The starting state is always in the top-middle of the ring and the agent can take two actions, either moving to the state to the left or the state to the right. There are two adjoining terminal states at the bottom middle of the ring. The left end gives -1 reward and the right end gives $+1$. Reaching the bottom two terminal states result in the teleportation back to the starting state. The sparsity of the rewards made the selection of an appropriate λ value worthy of investigating.

In the experiments, the RingWorld environment is reproduced with the help of the description in [28] and [9]. Despite being loyal to the original setting as much as possible, due to limitations of our understanding, we cannot see the difference between the RingWorld and a random walk environment with the rewards on the two tails. The RingWorld environment is described as a symmetric ring of the states with the starting state at the top-middle, for which we think the number of states should be odd. However, the authors claimed that they conducted experiments on the 10-state and 50-state instances.

We instead did the experiments on the 11-state instance. Thus in the experiments, we adopted a random walk version of RingWorld.

FrozenLake

The FrozenLake environment is a very noisy and high-variance domain in the OpenAI gym environment bundle [4]. The environment features a scenario in which an agent tries to fetch back a lost frisbee on the surface of a frozen lake. In this task, the agent controls the movement of a character in a grid world. Some tiles of the grid are walkable, and others lead to the agent falling into the water, which terminates the episode and teleports the agent back to the starting point. Furthermore, the movement direction of the agent is uncertain and only partially depends on the chosen direction. The agent is rewarded for finding a walkable path to a goal tile.

To make sure the Markovian properties of the environment, we removed the episode length limit of the FrozenLake environment and thus the environment can be solvable by dynamic programming (the true values of a policy as well as the state distribution).

It is modified based on the Gym environment with the same name. We have used the instance of "4x4", *i.e.* with 16 states.

The episode length limit of MountainCar is also removed. We also added noise to the state transitions: actions will be randomized at 20% probability. The noise is to prevent the cases in which $\lambda = 1$ yields the best performance (to prevent META from using extremely small κ's to get good performance). Additionally, due to the poor exploration of the softmax policy, we extended the starting location to be uniformly anywhere from the left to right on the slopes.

State Features

For RingWorld, we used onehot encoding to get equivalence to tabular case; For Frozen-Lake, we used a discrete variant of tile coding, for which there are 4 tilings, with each tile covering one grid as well as symmetric and even offset; For MountainCar, we adopted the

roughly the same setting as Chapter 10.1 Page 245 in [20], except that we used ordinary symmetric and even offset instead of the asymmetric offset.

About λ-greedy

We have replaced VTD [28] with direct VTD [18]. This modification is expected only to improve the stability, without touching the core mechanisms of λ-greedy [28].

The target used in [28] is biased toward $\lambda = 1$, as the λ's into the future are assumed to be 1. Thus we do not think it is helpful to conduct tests on environments with very low variance. For example, RingWorld is low-variance, as the state transitions are deterministic. Also, the policies adopted in [28] is very greedy. Such setting further reduces the variance. This is the reason why we have tested different policies (less greedy) in our experiments.

Buffer Period and Learning Rate

In the prediction tasks, we used the first 10% of the episodes as the buffer period. The learning rate hyperparameters of the auxiliary learners are set to be the twice of that of the value learner. These settings were not considered in [28], in which there were no buffer period and identical learning rates were used for all learners.

For the control task of MountainCar, λ-greedy and META will both crash without these two additional settings.

Details for Non-Parametric $\lambda(\cdot)$

To disable the generalization of the parametric $\boldsymbol{\lambda}$ for "META(np)", we replaced the feature vectors for each state with onehot-encoded features.

More Policies for Prediction

For RingWorld, we have done 6 different behavior-target policy pairs (3 on-policy & 3 off-policy). The off-policy pair that we have shown in the manuscript shares the same

patterns as the rest of the pairs. The accuracy improvement brought by META is significant across these pairs of policies; For FrozenLake, we have done two pairs of policies (on- and off-policy). We observe the same pattern as in the RingWorld tests.

Implementation

The source code could be found at `https://github.com/PwnerHarry/META`. The implementation is based on numpy and python, with massive parallelization for hyper-processing workers. These are to ensure fast experimental results on large scale CPUs.

Due to the estimation instability, sometimes the META updates could bring state λ values outside $[0, 1]$. In the implementation, whenever we detect such kind of update, we simply cancel that operation.