Predictor Corrector Policy Optimization

Ching-An Cheng Gerogia Tech/NVIDIA
Xinyan Yan Gerogia Tech
Nathan Raliff NVIDIA
Byron Boots Georgia Tech/NVIDIA

Abstract

We present a predictor-corrector framework, called PICCOLO, that can transform a first-order model-free reinforcement or imitation learning algorithm into a new hybrid method that leverages predictive models to accelerate policy learning. The new "PICCOLOed" algorithm optimizes a policy by recursively repeating two steps: In the Prediction Step, the learner uses a model to predict the unseen future gradient and then applies the predicted estimate to update the policy; in the Correction Step, the learner runs the updated policy in the environment, receives the true gradient, and then corrects the policy using the gradient error. Unlike previous algorithms, PICCOLO corrects for the mistakes of using imperfect predicted gradients and hence does not suffer from model bias. The development of PICCOLO is made possible by a novel reduction from predictable online learning to adversarial online learning, which provides a systematic way to modify existing first-order algorithms to achieve the optimal regret with respect to predictable information. We show, in both theory and simulation, that the convergence rate of several first-order model-free algorithms can be improved by PICCOLO.

1 Introduction

Reinforcement learning (RL) has recently solved a number of challenging problems [1, 2, 3]. However, many of these successes are confined to games and simulated environments, where a large number of agent-environment interactions can be cheaply performed. Therefore, they are often unrealistic in real-word applications (like robotics) where data collection is an expensive and time-consuming process. Improving sample efficiency still remains a critical challenge for RL.

Model-based RL methods improve sample efficiency by using an accurate model that can cheaply simulate interactions to compute policy updates without real-world interactions [4]. A classical example is optimal control [5, 6, 7, 8], which has recently been extended to model abstract latent dynamics with structured neural networks [9, 10]. These methods use a model of the dynamics and cost functions to predict cost-to-go functions or policy gradients when updating policies. Another way to use model information is the hybrid DYNA framework [11, 12], which interleaves model-based and model-free updates, ideally cutting learning time in half. However, all these approaches, while potentially accelerating learning, suffer from a common drawback: when the model is inaccurate, the performance of the policy can become *biased* away from the best achievable in the policy class.

Several strategies have been proposed to remove this performance bias. Learning-to-plan attempts to train the planning process end-to-end [13, 14]. While using a policy structure similar to the model-based algorithms, these algorithms are still optimized through standard model-free RL techniques, so it is unclear as to whether they are more sample efficient. Another class of bias-free algorithms is control variate methods [15, 16, 17], which use models to reduce the variance of sampled gradients.

In this paper, we provide a novel learning framework that can leverage models to improve sample efficiency while avoiding performance bias due to modeling errors. Our approach is built on techniques from online learning [18, 19]. The use of online learning to analyze policy optimization was pioneered by Ross et al. [20], who proposed to reduce imitation learning (IL) to adversarial online learning problems. This reduction provides a framework for performance analysis, leading to algorithms such as DAGGER [20] and AGGREVATE [21]. However, it was recently shown that a naïve

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

reduction to adversarial online learning loses information [22]: in practice, IL is *predictable* [23] and can be thought of as a predictable online learning problem [24]. Based on this insight, Cheng et al. [23] recently proposed a two-step algorithm, MOBIL. The authors prove that, by leveraging predictive models to estimate future gradients in an interleaved fashion, MOBIL can speed up the convergence of IL, without incurring performance bias due to imperfect models.

Given these theoretical advances in IL, it is natural to ask if similar ideas can be extended to RL. Here we show that RL can also be formulated as a predictable online learning problem, and provide a first-order learning framework, PICCOLO (PredICtor-COrrector poLicy Optimization), for both RL and IL. PICCOLO is a *meta-algorithm*: it takes a standard online learning algorithm designed for adversarial problems (like ADAM [25]) as input and returns a new algorithm that can use model information to accelerate convergence. This new "PICCOLOed" algorithm optimizes the policy by recursively repeating two steps: In the Prediction Step, the learner uses a predictive model to estimate the gradient of the next loss function and then applies it to update the policy; in the Correction Step, the learner runs the updated policy in the environment, receives the true gradient, and then corrects the policy using the gradient *error*. We note that PICCOLO is an approach orthogonal to control variate methods; it can still improve learning even if the true gradient is noise-free (see Section 4.3.2).

Theoretically, we prove that PICCOLO can improve the convergence rate of *any* base algorithm that can be written as mirror descent [26] or Follow-the-Regularized-Leader (FTRL) [27]. This family of algorithms is rich and covers most first-order algorithms used in RL and IL [28]. And, importantly, we show that PICCOLO does not suffer from performance bias due to model error, unlike previous model-based approaches. To validate the theory, we "PICCOLO" multiple algorithms in simulation. The experimental results show that the PICCOLOed versions consistently surpass the base algorithm and are robust to model errors; they converge even the models are adversarial.

The design of PICCOLO is made possible by a novel reduction that converts a given predictable online learning problem into a new adversarial problem, so that standard online learning algorithms can be applied optimally without referring to specialized algorithms. Thus, we can treat PICCOLO as an automatic process for designing new algorithms that safely leverages imperfect predictive models (such as off-policy gradients or gradients simulated through dynamics models) to speed up learning. Particularly, MOBIL is its special case when the base algorithm is FTRL.

2 Problem Definition

We consider solving policy optimization problems of the form below: given state and action spaces $\mathbb{S}$ and $\mathbb{A}$, and a parametric policy class Π, we desire a stationary policy $\pi \in \Pi$ that solves

$$\min_{\pi\in\Pi} J(\pi), \quad J(\pi) \coloneqq \mathbb{E}_{(s,t)\sim d_\pi}\mathbb{E}_{a\sim\pi_s}\left[c_t(s,a)\right] \tag{1}$$

where $c_t(s,a)$ is the instantaneous cost at time t of state $s \in \mathbb{S}$ and $a \in \mathbb{A}$, π_s is the distribution of a at state s under policy π, and d_π is a generalized stationary distribution of states generated by running policy π in a Markov decision process (MDP); the notation $\mathbb{E}_{a\sim\pi_s}$ denotes evaluation when π is deterministic. The use of d_π in (1) abstracts different discrete-time RL/IL problems into a common setup. For example, an infinite-horizon γ-discounted problem with time-invariant cost c can be modeled by setting $c_t = c$ and $d_\pi(s,t) = (1-\gamma)\gamma^t d_{\pi,t}(s)$, where $d_{\pi,t}$ is the state distribution visited by policy π at time t; that is, we can equivalently write $J(\pi) = (1-\gamma)\mathbb{E}_{\rho_\pi}[\sum_{t=0}^{\infty}\gamma^t c(s_t,a_t)]$, where ρ_π is the trajectory distribution generated by π. An IL problem can also be written similarly [22]. For convenience, we will usually omit the random variable in expectation notation to simplify writing (e.g. we write (1) as $\mathbb{E}_{d_\pi}\mathbb{E}_\pi\left[c\right]$). For a policy π, we overload the notation π to also denote its parameter, and write $Q_{\pi,t}$ and $V_{\pi,t} \coloneqq \mathbb{E}_\pi[Q_{\pi,t}]$ as its Q-function and value function at time t, respectively.

3 IL and RL as Predictable Online Learning

Online learning is a theoretical framework for analyzing online decision making [29]. For policy optimization, let us consider the following setup: in round n, the learner plays a policy π_n in the MDP environment of (1), and then the environment reveals a loss function l_n, called the *per-round loss*. The analysis of online learning concerns *regret*, which measures how the performance of the learner's decision sequence $\{\pi_n\}$ is compared with any single constant decision in the set Π. Given a policy class Π and a weight sequence $\{w_n > 0\}$, we define the N-step weighted regret of $\{\pi_n\}$ with respect to the loss sequence $\{l_n\}$ as

$$\text{regret}_N(l) \coloneqq \textstyle\sum_{n=1}^N w_n l_n(\pi_n) - \min_{\pi\in\Pi}\sum_{n=1}^N w_n l_n(\pi) \tag{2}$$

and an expressiveness measure of the policy class Π

$$\epsilon_{\Pi,N}(l) \coloneqq \min_{\pi\in\Pi} \frac{1}{w_{1:N}} \sum_{n=1}^{N} w_n l_n(\pi) \tag{3}$$

where $w_{1:n} = \sum_{m=1}^{n} w_m$. In this section, we show that the policy optimization problems in IL and RL can be formulated into online learning problems for some per-round losses such that the policy performance can be upper-bounded in terms of regret_N and $\epsilon_{\Pi,N}$. In particular, we show that these online learning problems are *predictable*: the per-round losses are not completely adversarial but can be estimated using past information.

3.1 IL as Online Learning

We first summarize the online learning approach to IL (online IL for short) [20, 22]. The core idea of IL is to use domain knowledge about a problem through expert demonstrations. Online IL, in particular, optimizes policies by letting the learner π query the expert $\pi^\star$ for the desired action, so that a policy can be quickly trained to perform as well as the expert. Online IL at its heart is based on the following lemma, which relates the performance between π and $\pi^\star$.

Lemma 3.1 (Performance Difference Lemma [30]). *Let π and π' be two policies and $A_{\pi',t}(s,a) = Q_{\pi',t}(s,a) - V_{\pi',t}(s)$ be the (dis)advantage function with respect to running π'. Then*

$$J(\pi) = J(\pi') + \mathbb{E}_{d_\pi}\mathbb{E}_\pi[A_{\pi'}]. \tag{4}$$

Given the equality in (4), the performance difference between π and $\pi^\star$ can then be upper-bounded as

$$J(\pi) - J(\pi^\star) = \mathbb{E}_{d_\pi}\mathbb{E}_\pi[A_{\pi^\star}] \le C_{\pi^\star}\mathbb{E}_{d_\pi}[D(\pi^\star||\pi)] \tag{5}$$

for some function D_t and constant $C_{\pi^\star} > 0$. Often D_t is derived from statistical distances such as KL-divergence. When $V_{\pi^\star,t}$ is available, we can also set $D_t(\pi^\star_s||\pi_s) = \mathbb{E}_{a\sim\pi_s}[A_{\pi^\star,t}(s,a)]$, as in value aggregation (AGGREVATE) [21]. Without loss of generality, we suppose $D_t(\pi^\star_s||\pi_s) = \mathbb{E}_{a\sim\pi_s}[\bar{c}_t(s,a)]$ for some $\bar{c}_t$. By (5), minimizing the performance gap is a new RL problem $\min_{\pi\in\Pi}\mathbb{E}_{d_\pi}\mathbb{E}_\pi[\bar{c}]$. Online IL tackles this new problem indirectly through online learning, thereby avoiding difficulties in RL (such as estimating value functions). It defines a per-round loss $\tilde{l}_n$ that is an unbiased estimate of

$$l_n(\pi) \coloneqq \mathbb{E}_{d_{\pi_n}}\mathbb{E}_\pi[\bar{c}]. \tag{6}$$

The regret with respect to $\tilde{l}_n$ upper-bounds the performance gap between the learner and the expert.

Lemma 3.2 ([23]). *Let $\{w_n > 0\}$ be a weight sequence. Then*

$$\mathbb{E}\left[\sum_{n=1}^{N} \frac{w_n J(\pi_n)}{w_{1:N}}\right] \le J(\pi^\star) + C_{\pi^\star}\mathbb{E}\left[\epsilon_{\Pi,N}(\tilde{l}) + \frac{\text{regret}_N(\tilde{l})}{w_{1:N}}\right]$$

where the expectation is due to sampling $\tilde{l}_n$.

Therefore, when a no-regret algorithm is used, the performance converges to $J(\pi^\star) + C_{\pi^\star}\mathbb{E}[\epsilon_{\Pi,N}(\tilde{l})]$.

3.2 RL as Online Learning

Can we also formulate RL as online learning? Here we propose a new perspective on RL using Lemma 3.1. Given a policy π_n in round n, we define a per-round loss $\tilde{f}_n$ as an unbiased estimate of

$$f_n(\pi) = \mathbb{E}_{d_{\pi_n}}\mathbb{E}_\pi[A_{\pi_{n-1}}]. \tag{7}$$

$f_n(\pi)$ describes how well a policy π performs relative to the previous policy π_{n-1} under the state distribution of π_n. For example, by Lemma 3.1, we see that $f_n(\pi_n) = J(\pi_n) - J(\pi_{n-1})$. This choice of per-round loss in (7) has an interesting relationship both to actor-critic in RL [31] and AGGREVATE in IL [21].

Relationship to Actor-Critic Although, theoretically, the actor-critic method computes the gradient $\mathbb{E}_{d_{\pi_n}}(\nabla\mathbb{E}_\pi)[A_{\pi_n}]|_{\pi=\pi_n}$ to update the policy π_n, in practice, it computes $\mathbb{E}_{d_{\pi_n}}(\nabla\mathbb{E}_\pi)[A_{\pi_{n-1}}]|_{\pi=\pi_n}$, as the value function estimate in round n is updated only after updating the policy π_n. This gradient is *exactly* $\nabla f_n(\pi_n)$ in (7).

Relationship to Value Aggregation AGGREVATE can be viewed as taking a policy improvement step from some reference policy: e.g., with the per-round loss $\mathbb{E}_{d_{\pi_n}}\mathbb{E}_{\pi}[A_{\pi^\star}]$, it improves one step from π^*. Realizing this one step improvement in AGGREVATE, however, requires solving multiple rounds of online learning, because Π does not contain all functions and the dynamics are unknown [22]. Therefore, while ideally one can solve multiple AGGREVATE problems (one per each policy improvement step) to optimize policies, computationally this can be very challenging. Minimizing the loss in (7) can be viewed as an approximate policy improvement step in the AGGREVATE style. Rather than waiting until convergence in each AGGREVATE policy improvement step, it performs only a *single* policy update and then switch to the next AGGREVATE problem with a new reference policy (i.e. the latest policy).

We show that the regret of the online learning problem with $\tilde{f}_n$ upper-bounds the on-average performance of a policy sequence (proved in Appendix C).

Lemma 3.3. *Let$\{w_n > 0\}$ be a weight sequence satisfying $\frac{w_{n+k}}{w_n} \leq \frac{w_{m+k}}{w_m}$, $\forall n \geq m \geq 1$ and $k \geq 0$, Then, for arbitrary initial reference policy π_0,*

$$\mathbb{E}\left[\sum_{n=1}^{N} \frac{w_n J(\pi_n)}{w_{1:N}}\right] \leq J(\pi_0) \quad + \sum_{n=1}^{N} \frac{w_{N-n+1}}{w_{1:N}} \mathbb{E}\left[\text{regret}_n(\tilde{f}) + w_{1:n}\epsilon_{\Pi,n}(\tilde{f})\right] \tag{8}$$

where the expectation is due to sampling $\tilde{f}_n$.

Interestingly, Lemma 3.3 applies to any initial reference policy π_0, which can be different from the learner's initial policy π_1. In particular, if we choose $\pi_0 = \pi^\star$, then the bound in (8) becomes relative to $J(\pi^\star)$. This choice tightens the link between (7), actor-critic, and AGGREVATE.

In Lemma 3.3, the amount of policy improvement is reflected in the term $\mathbb{E}[\text{regret}_n(\tilde{f}) + w_{1:n}\epsilon_{\Pi,n}(\tilde{f})]$ in (8), which trades off $\mathbb{E}[\text{regret}_n(\tilde{f})]$ and the potential performance improvement due to negative $\mathbb{E}[\epsilon_{\Pi,n}(\tilde{f})]$. Imagine $\mathbb{E}[\epsilon_{\Pi,n}(\tilde{f})] \leq -\Omega(1)$ and $\{\pi_n\}_{n=1}^N$ is updated by a no-regret algorithm. Then $\mathbb{E}[\text{regret}_n(\tilde{f}) + w_{1:n}\epsilon_{\Pi,n}(\tilde{f})] \leq -\Omega(w_{1:n})$. This means, e.g., if $w_n = 1$ and $\text{regret}_n(\tilde{f}) \leq O(\sqrt{n})$, then the average performance improves around $O(N\mathbb{E}[\epsilon_{\Pi,N}(\tilde{f})])$ away from $J(\pi_0)$ (which can be $J(\pi^\star)$). By contrast, in Lemma 3.2, the improvement is only a constant $O(\mathbb{E}[\epsilon_{\Pi,N}(\tilde{l})])$ distance from $J(\pi^\star)$. However, we note that currently we do not have a formal proof that results in an upper bound of $\mathbb{E}[\epsilon_{\Pi,n}(\tilde{f})]$. While it is intuitive to conjecture that $\mathbb{E}[\epsilon_{\Pi,n}(\tilde{f})]$ is negative in the early iterations or when the policy sequence is concentrated, because J is lower bounded, this intuition does not hold for all N. Still, Lemma 3.3 suggests an alternative perspective on AGGREVATE and actor-critic that are known to work well in practice. Further investigation into Lemma 3.3 is left to future work.

3.3 Predictability

An important property of the above online learning problems is that they are not completely adversarial for IL, as pointed out by Cheng and Boots [22]. This can be seen from the definitions of l_n and f_n in (6) and (7), respectively. For example, suppose the cost c_t in the original RL problem (1) is known; then the information unknown before playing the decision π_n in the environment is only the state distribution d_{π_n}. Therefore, the per-round loss cannot be truly adversarial, as the same dynamics and cost functions are used across different rounds. We will exploit this property to design PICCOLO.

4 A PREDICTOR-CORRECTOR LEARNING FRAMEWORK

We showed that the performance of RL and IL can be bounded by the regret of a properly constructed *predictable* online learning problem. This suggests that directly applying a standard online learning algorithm designed for adversarial settings (e.g. gradient/mirror descent) to these RL/IL problems is *suboptimal*. The predictable information must be considered in the policy update to achieve optimal convergence in learning.

One way to include predictable information is to develop specialized two-step algorithms based on mirror-prox or FTRL-prediction [32, 24, 33]. For IL, MOBIL was recently proposed [23], which updates policies by approximate Be-the-Leader [34] and provably achieves faster convergence than previous methods. However, these algorithms often have non-sequential update rules, and their adaptive versions are less accessible [35]. This can make it difficult to tune them in practice.

Here we take an alternative, reduction-based approach. We present PICCOLO, a general first-order framework for solving predictable online learning problems. PICCOLO is a meta-algorithm that

turns a base algorithm designed for adversarial problems into a new algorithm that can leverage the predictable information to achieve better performance. As a result, we can adopt sophisticated first-order adaptive algorithms to optimally learn policies, without reinventing the wheel. Specifically, given *any* first-order base algorithm belonging to the family of (adaptive) mirror descent and FTRL algorithms, we show how one can "PICCOLO it" to achieve a faster convergence rate without introducing additional performance bias due to prediction errors. Most existing first-order policy optimization algorithms belong to this family [28], so we can use PICCOLO to wrap these model-free algorithms into new hybrid algorithms that can robustly use (imperfect) predictive models, such as off-policy gradients and simulated gradients, to improve policy learning.

4.1 Building Blocks

We begin by reviewing mirror descent and FTRL, which will be used as the building blocks of PICCOLO. We assume that Π is a closed and convex subset in some normed space with norm $\|\cdot\|$, and we use $B_R(\pi||\pi') = R(\pi) - R(\pi') - \langle \nabla R(\pi'), \pi - \pi' \rangle$ to denote a Bregman divergence generated by a strictly convex function R, called the distance generator.

Mirror descent updates the decision variable based on proximal maps [36]. In round n, given first-order information g_n and weight w_n, it executes

$$\pi_{n+1} = \arg\min_{\pi \in \Pi} \langle w_n g_n, \pi \rangle + B_{R_n}(\pi||\pi_n) \tag{9}$$

where R_n is a strongly convex function. Mirror descent reduces to gradient descent with step size η_n when when $R_n(\pi) = \frac{1}{2\eta_n}\|\pi\|^2$. Alternatively, FTRL runs

$$\pi_{n+1} = \arg\min_{\pi \in \Pi} \textstyle\sum_{m=1}^{n} \langle w_m g_m, \pi \rangle + B_{r_m}(\pi||\pi_m), \tag{10}$$

where r_m is a strongly convex function. To connect FTRL with mirror descent, we can roughly think of $\langle (wg)_{1:n-1}, \cdot \rangle + \sum_{m=1}^{n} B_{r_m}$ as B_{R_n} in (9), since both act as regularization. In particular, when Π is unconstrained, they are indeed equivalent [37].

4.2 The Meta-Algorithm PICCOLO

PICCOLO decomposes a base algorithm (mirror descent or FTRL) into three basic operations

$$h \leftarrow \texttt{update}(h, H, g, w), \quad H \leftarrow \texttt{adapt}(h, H, g, w), \quad \pi \leftarrow \texttt{project}(h, H) \tag{11}$$

and recomposes them to generate the new algorithm, where g is a update direction, w is a weight, h is a representation of π, and H is a regularization function. The operators `update` and `adapt` define how h and H change when receiving first-order information g and weight w, respectively, and `project` describes how h is decoded into a policy π. Below we show how these operations define (9) and (10) and how how PICCOLO uses them to define a new algorithm (see Appendix D for details).

4.2.1 Base Algorithms

We first show the update rules in (9) and (10) can be written within a single framework as

$$H_n = \texttt{adapt}(h_n, H_{n-1}, g_n, w_n), \quad h_{n+1} = \texttt{update}(h_n, H_n, g_n, w_n) \tag{12}$$

In round n, both algorithms first decode a policy $\pi_n = \texttt{project}(h_n, H_{n-1})$ and run it in the environment to get g_n (which is equal to $\nabla \tilde{l}_n(\pi_n)$ in IL or $\nabla \tilde{f}_n(\pi_n)$ in RL). Then, based on g_n and w_n, they use `adapt` to update H_n and use `update` to compute h_{n+1} using the up-to-date H_n.

To validate this, we first define (h, H), `update`, and `project`. In mirror descent, we identify $h = \pi$ and H as the distance generator in (9), and define

$$\texttt{update}(h, H, g, w) = \arg\min_{\pi' \in \Pi} \langle wg, \pi' \rangle + B_H(\pi'||h), \quad \texttt{project}(h, H) = h \tag{13}$$

In FTRL, we identify H as the sum of Bregman divergences in (10) and h as the weighted sum of past first-order information, and define

$$\texttt{update}(h, H, g, w) = h + wg, \quad \texttt{project}(h, H) = \arg\min_{\pi' \in \Pi} \langle h, \pi' \rangle + H(\pi') \tag{14}$$

Before presenting the details of `adapt`, we can already verify that, with the above identifications, (12) indeed implements the classical update rules (9) and (10).

Algorithm 1 PICCOLO

Input: policy π_1, cost sequence $\{\psi_n\}$, regularization H_0, model Φ_1, iteration N, exponent p
Output: $\bar{\pi}_N$

Set $\hat{\pi}_1 = \pi_1$ and weights $w_n = n^p$
Sample integer K with $P(K = n) \propto w_n$
for $n = 1 \dots K - 1$* **do**
  $\hat{g}_n = \Phi_n(\hat{\pi}_n, \psi_n)$
  $\pi_n = \texttt{PredictionStep}(\hat{\pi}_n, \hat{g}_n, H_{n-1}, w_n)$
  $\mathcal{D}_n, g_n = \texttt{DataCollection}(\pi_n, \psi_n)$
  $H_n, \hat{\pi}_{n+1} = \texttt{CorrectionStep}(\pi_n, e_n, H_{n-1}, w_n)$, where $e_n = g_n - \hat{g}_n$.
  $\Phi_{n+1} = \texttt{ModelUpdate}(\Phi_n, \mathcal{D})$, where $\mathcal{D} = \mathcal{D} \bigcup \mathcal{D}_n$.
end for
Set $\bar{\pi}_N = \hat{\pi}_K$

The operator `adapt` is designed to update the regularization H (e.g. changing the step size) so that the online learner achieves small regret. Although the exact definition of `adapt` depends on the specific base algorithm, in general it updates the size of regularization H to grow *slowly* and *inversely* proportional to the norm of g. We will give a concrete example in Appendix A and please refer to Appendix D for details.

4.2.2 The PICCOLOed Algorithm

PICCOLO accelerates policy learning by using estimates of future gradients. The full algorithm is summarized in Algorithm 1. In round n, a "PICCOLOed" algorithm first queries the predictive model Φ_n for $\hat{g}_n$, which estimates g_n the gradient of the unseen per-round loss. Next, it performs the Prediction Step using $\hat{g}_n$ to generate the learner's decision (i.e. π_n) and runs this new policy in the environment to get the true gradient g_n. Using this feedback, the algorithm performs the Correction Step to amend the bias of using $\hat{g}_n$. This is accomplished by first adapting the regularization to H_n and then updating π_n to $\hat{\pi}_{n+1}$ according to prediction error $e_n = g_n - \hat{g}_n$. This intermediate policy $\hat{\pi}_{n+1}$ will be used as an estimate of the next decision π_{n+1} in querying Φ_{n+1} in the next round.

Update Rules PICCOLO recomposes the three basic operations of a base algorithm to define the Prediction and the Correction Steps in Algorithm 1[2]:

$$h_n = \texttt{update}(\hat{h}_n, H_{n-1}, \hat{g}_n, w_n) \qquad \text{[Prediction]} \tag{15}$$

$$\begin{aligned} H_n &= \texttt{adapt}(h_n, H_{n-1}, e_n, w_n) \\ \hat{h}_{n+1} &= \texttt{update}(h_n, H_n, e_n, w_n) \end{aligned} \qquad \text{[Correction]} \tag{16}$$

Like the base algorithms earlier, PICCOLO calls $\pi_n = \texttt{project}(h_n, H_{n-1})$ to make decisions. Here we write the dependencies in terms of h_n (instead of π_n) so they can be used with both mirror descent and FTRL. Note that in the Prediction Step, only h_n is updated, not the regularization. A concrete example of PICCOLOing ADAGRAD is provided in Appendix A.

Predictive Models The predictive model Φ_n is a first-order oracle, which can be updated online using past information. Given a policy π and a cost function ψ (e.g. $\bar{c}$ in IL or $A_{\pi_{n-1}}$ in RL), it predicts $\Phi_n(\pi, \psi)$ as an estimate of $\mathbb{E}_{d_\pi}(\nabla \mathbb{E}_\pi)[\psi]$. In PICCOLO, we use Φ_n and $\hat{\pi}_n = \texttt{project}(\hat{h}_n, H_{n-1})$ to construct $\hat{g}_n$. For example, we can set $\hat{g}_n = \Phi_n(\hat{\pi}_n, A_{\pi_{n-1}})$ to estimate the gradient in (7) in RL. A predictive model can be a simulator with an (online learned) dynamics model [4, 7], or a neural network trained to predict the required gradients [9, 10]. An even simpler way is to construct predictive models as *off-policy* gradients. For our previous example, $\Phi_n(\hat{\pi}_n, A_{\pi_{n-1}}) = \hat{\mathbb{E}}_{d_{\pi_{n-1}}}(\nabla \hat{\mathbb{E}}_\pi)[A_{\pi_{n-1}}]|_{\pi=\hat{\pi}_n}$ can be used to approximate the future gradient $g_n = \hat{\mathbb{E}}_{d_{\pi_n}}(\nabla \hat{\mathbb{E}}_\pi)[A_{\pi_{n-1}}]|_{\pi=\pi_n}$, where $\hat{\mathbb{E}}$ denotes finite sample approximation. A similar predictive model can be constructed using a replay buffer.

4.3 THEORETICAL ANALYSIS

We show that PICCOLO has two major benefits over previous approaches: 1) it accelerates policy learning when the models predict the required gradient well on average; and 2) it does not bias the performance of the policy, even when the prediction is incorrect. To analyze PICCOLO, we introduce an assumption to quantify the `adapt` operator of a base algorithm.

*Here we assume `project` is automatically performed inside `PredictionStep` and `CorrectionStep`.
[2]We provide a variation of PICCOLO in Appendix E.

Algorithms	Upper bounds in Big-O
PICCOLO	$\frac{1}{\sqrt{N}}\left(\lvert\Pi\rvert + \sigma_g^2 + \sigma_{\hat{g}}^2 + E_\Phi\right)$
model-free	$\frac{1}{\sqrt{N}}\left(\lvert\Pi\rvert + G_g^2 + \sigma_g^2\right)$
model-based	$\frac{1}{\sqrt{N}}\left(\lvert\Pi\rvert + G_{\hat{g}}^2 + \sigma_{\hat{g}}^2\right) + E_\Phi$
DYNA	$\frac{1}{\sqrt{2N}}\left(\lvert\Pi\rvert + \frac{1}{2}\left(G_g^2 + G_{\hat{g}}^2 + \sigma_g^2 + \sigma_{\hat{g}}\right)^2\right) + E_\Phi$

Table 1: Upper bounds of the average regret of different policy optimization algorithms.

Assumption 4.1. `adapt` *chooses a regularization sequence such that, for some* $M_N = o(w_{1:N})$, $\|H_0\|_{\mathcal{R}} + \sum_{n=1}^{N} \|H_n - H_{n-1}\|_{\mathcal{R}} \le M_N$ *for some norm* $\|\cdot\|_{\mathcal{R}}$.

This assumption, which requires the regularization to change slower than the growth of $w_{1:N}$, is satisfied by most reasonably-designed base algorithms. For example, in a uniformly weighted problem, gradient descent with a decaying step size $O(\frac{1}{\sqrt{n}})$ has $M_N = O(\sqrt{N})$. In general, for stochastic problems, an optimal base algorithm would ensure $M_N = O(\frac{w_{1:N}}{\sqrt{N}})$.

4.3.1 Convergence Properties

Now we state the main result, which quantifies the regret of PICCOLO with respect to the sequence of linear loss functions that it has access to. The proof is given in Appendix G, which is based on a general reduction from predictive online learning to adversarial online learning.

Theorem 4.1. *Suppose* H_n *defines a strongly convex function with respect to* $\|\cdot\|_n$. *Under Assumption 4.1, running* PICCOLO *ensures* $\sum_{n=1}^{N} \langle w_n g_n, \pi_n - \pi\rangle \le M_N + \sum_{n=1}^{N} \frac{w_n^2}{2}\|e_n\|_{*,n}^2 - \frac{1}{2}\|\pi_n - \hat{\pi}_n\|_{n-1}^2$, *for all* $\pi \in \Pi$.

The term $\|e_n\|_{*,n}^2$ in Theorem 4.1 says that the performance of PICCOLO depends on how well the base algorithm adapts to the error e_n, which depends on the `adapt` operation in the Correction Step. Usually `adapt` updates H_n gradually (Assumption 4.1) while minimizing $\frac{1}{2}\|e_n\|_{*,n}^2$, like we showed in ADAGRAD in Appendix A.

In general, when the base algorithm is adaptive and optimal for adversarial problems, we show in Appendix H that its PICCOLOed version guarantees that $\mathbb{E}[\sum_{n=1}^{N} \langle w_n g_n, \pi_n - \pi\rangle] \le O(1) + C_{\Pi,\Phi}\frac{w_{1:N}}{\sqrt{N}}$, where $C_{\Pi,\Phi} = O(|\Pi| + E_\Phi + \sigma_g^2 + \sigma_{\hat{g}}^2)$ is some constant related to the size of Π, the model bias E_Φ, and the sampling variance σ_g^2 and $\sigma_{\hat{g}}^2$ of g_n and $\hat{g}_n$, respectively. Through Lemma 3.2 and 3.3, this bound directly implies accelerated and bias-free policy performance.

Theorem 4.2. *Let* F_n *to be either* $\tilde{l}_n$ *or* $\tilde{f}_n$. *Suppose* F_n *is convex*[3] *and* $w_n \ge \Omega(1)$. *Then running* PICCOLO *yields* $\mathbb{E}[\text{regret}_n(F)/w_{1:N}] = O(\frac{C_{\Pi,\Phi}}{\sqrt{N}})$, *where* $C_{\Pi,\Phi} = O(|\Pi| + E_\Phi + \sigma_g^2 + \sigma_{\hat{g}}^2) = O(1)$.

4.3.2 Comparison

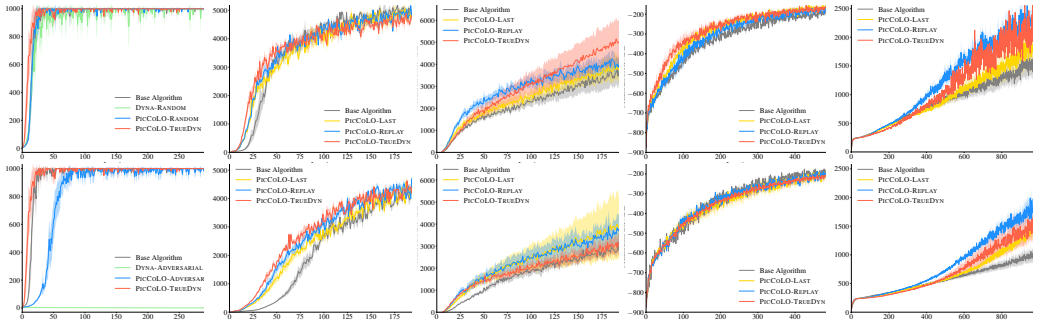
We compare several policy optimization algorithms and show that they can be viewed as incomplete versions of PICCOLO, which only either result in accelerated learning *or* are unbiased, but not both. The results are summarized in Table 1. Please see Appendix B for details.

5 EXPERIMENTS

We corroborate our theoretical findings with experiments in learning neural network policies to solve simulated robot control tasks (CartPole, Hopper, Snake, Reacher3D, and Walker3D) in the DART environment [39]. We use CartPole to verify some basic properties of PICCOLO and then use the other environments to study the empirical properties of PICCOLO in solving more complicated problems. We aim to see if PICCOLO improves the performance a base algorithm under various settings. We choose several popular first-order algorithms (ADAM [25], natural gradient descent NATGRAD [40], and trust-region optimizer TRPO [41]). We consider RL problems and compute g_n by GAE [42]. For predictive models, we consider off-policy gradients (with the samples of the last iteration LAST or a replay buffer REPLAY) and gradients computed through simulations with the true dynamics model[4] (TRUEDYN). Please refer to Appendix I for the details and complete results.

[3] The convexity assumption is standard, as used in [38, 20, 25, 22], which holds for tabular problems as well as some special cases, like continuous-time problems (cf. [22]).

[4] As we focus on studying the properties of PICCOLO, we assume that a good dynamics model is available.



(a) Cartpole (b) Hopper (c) Snake (d) Reacher3D (e) Walker3D

Figure 1: The performance of PICCOLO in various tasks. The first column shows PICCOLO and DYNA with predictive models of different accuracies (upper: random dynamics; lower: adversarial model) and ADAM as the base algorithm. From the 2nd to the 5th columns we show the performance of PICCOLO in more difficult environments (the first and the second rows use NATGRAD and TRPO as the base algorithms, respectively.) The shaded regions account for 0.5 standard deviation.

We use CartPole to study Theorem 4.2, which suggests that PICCOLO is unbiased and improves the performance when the prediction is accurate. Here we additionally consider two inaccurate models: RANDOM that predicts gradients using a randomly specified neural network dynamics, and ADVERSARIAL that predicts the gradients adversarially[5]. Figure 1(a) illustrates the performance of PICCOLO and DYNA, when ADAM is chosen as the base algorithm. We observe that PICCOLO improves the performance when the model is accurate (i.e. TRUEDYN). Moreover, PICCOLO is robust to modeling errors. It performs closely to the base algorithm (ADAM in this case) when using RANDOM, converging to the optimal performance; on the other hand, DYNA yields a performance bias. When the predictive model becomes adversarial, PICCOLO still converges, whereas DYNA fails completely. Next, in Figure 1 (b)-(e), we study the performance of PICCOLO in a range of simulated environments using TRPO and NATGRAD as base algorithms. In general, we find that PICCOLO indeed improves the performance[6] of TRPO and NATGRAD, although we also find that the exact degree of improvement depends on the environment as well as the predictive model used. For example, REPLAY gives a significant improvement in the difficult Walker3D environment, whereas it performs similarly to LAST in the other settings. Also, we find that PICCOLOed TRPO has slightly worse performance than TRPO in Reacher3D[7], whereas PICCOLO does improve the performance of NATGRAD. Finally, we note that TRUEDYN provides improvement in various settings. Although its computation time might be considered impractical to these simulated problems, it can be valuable in applications where agent-environment interaction is expensive.

6 CONCLUSION

PICCOLO is a general reduction-based framework for solving predictable online learning problems. It can be viewed as an automatic strategy for designing new algorithms that can use prediction to accelerate convergence. Furthermore, PICCOLO uses the Correction Step to recover from the mistake made in the Prediction Step, so the presence of model errors does not bias convergence, as we show in both the theory and experiments. The design of PICCOLO opens the question of designing predictive models. While PICCOLO is robust against modeling error, the accuracy of a predictive model can affect its effectiveness. PICCOLO only improves the performance when the model makes non-trivial predictions. In the experiments, we found that off-policy and simulated gradients are often useful, but they are not perfect. It is interesting to see if a predictive model that is trained to directly minimize the prediction error can further help policy learning. Also, the current design of PICCOLO takes only a gradient update in the Prediction Step. An open question is how to adaptively take multiple updates in the Prediction Step in accordance with the model accuracy, while properly correcting for the bias in the Correction Step. For example, if the predictive model is exact, an ideal Prediction Step should completely solve the problem. Finally, we note that, despite the current focus of this paper on policy optimization, PICCOLO can naturally be applied to other optimization problems.

[5]We set the adversarial model to predict $\hat{g}_{n+1} = -g_n \times \max_{m=1,\dots,n} \|g_m\|/\|g_n\|$.

[6]Different base algorithms are not directly comparable, as further fine-tuning of step sizes is required.

[7]After an investigation, we found that the prediction errors are larger than the true gradients, i.e. using the predictive models are worse than predicting zero.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[2] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[3] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[4] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

[5] David H Jacobson and David Q Mayne. Differential dynamic programming. 1970.

[6] Emanuel Todorov and Weiwei Li. A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *American Control Conference*, pages 300–306. IEEE, 2005.

[7] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *International Conference on machine learning*, pages 465–472, 2011.

[8] Yunpeng Pan and Evangelos Theodorou. Probabilistic differential dynamic programming. In *Advances in Neural Information Processing Systems*, pages 1907–1915, 2014.

[9] David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, et al. The predictron: End-to-end learning and planning. *arXiv preprint arXiv:1612.08810*, 2016.

[10] Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. In *Advances in Neural Information Processing Systems*, pages 6120–6130, 2017.

[11] Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163, 1991.

[12] Richard S Sutton, Csaba Szepesvári, Alborz Geramifard, and Michael P Bowling. Dyna-style planning with linear function approximation and prioritized sweeping. *arXiv preprint arXiv:1206.3285*, 2012.

[13] Razvan Pascanu, Yujia Li, Oriol Vinyals, Nicolas Heess, Lars Buesing, Sebastien Racanière, David Reichert, Théophane Weber, Daan Wierstra, and Peter Battaglia. Learning model-based planning from scratch. *arXiv preprint arXiv:1707.06170*, 2017.

[14] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks. *arXiv preprint arXiv:1804.00645*, 2018.

[15] Yevgen Chebotar, Karol Hausman, Marvin Zhang, Gaurav Sukhatme, Stefan Schaal, and Sergey Levine. Combining model-based and model-free updates for trajectory-centric reinforcement learning. *arXiv preprint arXiv:1703.03078*, 2017.

[16] Will Grathwohl, Dami Choi, Yuhuai Wu, Geoff Roeder, and David Duvenaud. Backpropagation through the void: Optimizing control variates for black-box gradient estimation. *arXiv preprint arXiv:1711.00123*, 2017.

[17] Matteo Papini, Damiano Binaghi, Giuseppe Canonaco, Matteo Pirotta, and Marcello Restelli. Stochastic variance-reduced policy gradient. *arXiv preprint arXiv:1806.05618*, 2018.

[18] Geoffrey J Gordon. Regret bounds for prediction problems. In *Annual Conference on Computational Learning Theory*, pages 29–40. ACM, 1999.

[19] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *International Conference on Machine Learning*, pages 928–936, 2003.

[20] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *International conference on artificial intelligence and statistics*, pages 627–635, 2011.

[21] Stephane Ross and J Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*, 2014.

[22] Ching-An Cheng and Byron Boots. Convergence of value aggregation for imitation learning. In *International Conference on Artificial Intelligence and Statistics*, volume 84, pages 1801–1809, 2018.

[23] Ching-An Cheng, Xinyan Yan, Evangelos Theodorou, and Byron Boots. Accelerating imitation learning with predictive models. *arXiv preprint arXiv:1806.04642*, 2018.

[24] Alexander Rakhlin and Karthik Sridharan. Online learning with predictable sequences. 2013.

[25] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[26] Amir Beck and Marc Teboulle. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3):167–175, 2003.

[27] H Brendan McMahan and Matthew Streeter. Adaptive bound optimization for online convex optimization. *arXiv preprint arXiv:1002.4908*, 2010.

[28] Ching-An Cheng, Xinyan Yan, Nolan Wagener, and Byron Boots. Fast policy learning through imitation and reinforcement. *arXiv preprint arXiv:1805.10413*, 2018.

[29] Elad Hazan et al. Introduction to online convex optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325, 2016.

[30] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning*, volume 2, pages 267–274, 2002.

[31] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[32] Anatoli Juditsky, Arkadi Nemirovski, and Claire Tauvel. Solving variational inequalities with stochastic mirror-prox algorithm. *Stochastic Systems*, 1(1):17–58, 2011.

[33] Nam Ho-Nguyen and Fatma Kilinc-Karzan. Exploiting problem structure in optimization under uncertainty via online convex optimization. *arXiv preprint arXiv:1709.02490*, 2017.

[34] Adam Kalai and Santosh Vempala. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 71(3):291–307, 2005.

[35] Jelena Diakonikolas and Lorenzo Orecchia. Accelerated extra-gradient descent: A novel accelerated first-order method. *arXiv preprint arXiv:1706.04680*, 2017.

[36] Neal Parikh, Stephen Boyd, et al. Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014.

[37] H Brendan McMahan. A survey of algorithms and analysis for adaptive online learning. *The Journal of Machine Learning Research*, 18(1):3117–3166, 2017.

[38] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[39] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[40] Sham M Kakade. A natural policy gradient. In *Advances in neural information processing systems*, pages 1531–1538, 2002.

[41] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[42] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[43] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, pages 1057–1063, 2000.

[44] Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190, 2008.

[45] Jan Peters, Katharina Mülling, and Yasemin Altun. Relative entropy policy search. In *AAAI*, pages 1607–1612. Atlanta, 2010.

[46] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, 2014.

[47] Wen Sun, Arun Venkatraman, Geoffrey J Gordon, Byron Boots, and J Andrew Bagnell. Deeply aggrevated: Differentiable imitation learning for sequential prediction. *arXiv preprint arXiv:1703.01030*, 2017.

[48] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2): 26–31, 2012.

[49] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[50] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *International Conference on Learning Representations*, 2018.

[51] Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2): 251–276, 1998.

[52] Elad Hazan, Amit Agarwal, and Satyen Kale. Logarithmic regret algorithms for online convex optimization. *Machine Learning*, 69(2-3):169–192, 2007.

[53] Vineet Gupta, Tomer Koren, and Yoram Singer. A unified approach to adaptive regularization in online and stochastic optimization. *arXiv preprint arXiv:1706.06569*, 2017.

[54] Sasha Rakhlin and Karthik Sridharan. Optimization, learning, and games with predictable sequences. In *Advances in Neural Information Processing Systems*, pages 3066–3074, 2013.

[55] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

[56] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[57] Jeongseok Lee, Michael X. Grey, Sehoon Ha, Tobias Kunz, Sumit Jain, Yuting Ye, Siddhartha S. Srinivasa, Mike Stilman, and C. Karen Liu. DART: Dynamic animation and robotics toolkit. *The Journal of Open Source Software*, 3(22):500, feb 2018.

A Why Does PICCOLO Work?

PICCOLO uses the predicted gradient to take an extra step to accelerate learning. In the meanwhile, to prevent the error accumulation, it adaptively adjusts the step size (i.e. the regularization) based on the prediction error and corrects for the bias on the policy right away. To gain some intuition, let us consider ADAGRAD [38] as a base algorithm[8]:

$$G_n = G_{n-1} + \mathrm{diag}(w_n g_n \odot w_n g_n)$$
$$\pi_{n+1} = \arg\min_{\pi\in\Pi} \langle w_n g_n, \pi\rangle + \frac{1}{2\eta}(\pi - \pi_n)^\top G_n^{1/2}(\pi - \pi_n)$$

where $G_0 = \epsilon I$ and $\eta, \epsilon > 0$, and $\odot$ denotes element-wise multiplication. With its `update` and `project` defined standardly in (13), its `adapt` is defined as $\texttt{adapt}(h, H, g, w) = G + \mathrm{diag}(wg \odot wg)$ which updates the Bregman divergence based on the gradient size.

PICCOLO transforms ADAGRAD into a new algorithm. In the Prediction Step, it performs

$$\pi_n = \arg\min_{\pi\in\Pi} \langle w_n \hat{g}_n, \pi\rangle + \frac{1}{2\eta}(\pi - \pi_{n-1})^\top G_{n-1}^{1/2}(\pi - \pi_{n-1})$$

In the Correction Step, it performs

$$G_n = G_{n-1} + \mathrm{diag}(w_n e_n \odot w_n e_n)$$
$$\hat{\pi}_{n+1} = \arg\min_{\pi\in\Pi} \langle w_n e_n, \pi\rangle + \frac{1}{2\eta}(\pi - \hat{\pi}_n)^\top G_n^{1/2}(\pi - \hat{\pi}_n)$$

We see that the PICCOLO-ADAGRAD updates G_n proportional to the prediction error e_n instead of g_n. It takes larger steps when models are accurate, and decreases the step size once the prediction deviates. As a result, PICCOLO is robust to model quality: it accelerates learning when the model is informative, and prevents inaccurate (potentially adversarial) models from hurting the policy. We will further demonstrate this in theories and experiments.

B Theoretical Comparison

Here we provide a discussion of Table 1.

We first consider the common model-free approach [43, 40, 44, 45, 46, 47, 28], i.e. applying the base algorithm with g_n. To make the comparison concrete, suppose $\|\mathbb{E}[g_n]\|_*^2 \le G_g^2$ for some constant G_g, where we recall g_n is the sampled true gradient. As the model-free approach is equivalent to setting $\hat{g}_n = 0$ in PICCOLO, by Theorem 4.1 (with $e_n = g_n$), the constant C_Π in Theorem 4.2 would become $O(|\Pi| + G_g^2 + \sigma_g^2)$. In other words, PICCOLOing the base algorithm improves the constant factor from G_g^2 to $\sigma_{\hat{g}}^2 + E_\Phi$. Therefore, while the model-free approach is bias-free, its convergence can be further improved by PICCOLO, as long as the models $\{\Phi_n\}$ are reasonably accurate on average.[9]

Next we consider the pure model-based approach with a model that is potentially learned online [5, 6, 7, 8]. As such approach is equivalent to only performing the Prediction Step, its performance suffers from any modeling error. Specifically, suppose $\|\mathbb{E}[\hat{g}_n]\|_*^2 \le G_{\hat{g}}^2$ for some constant $G_{\hat{g}}$. One can show that the bound in Theorem 4.2 would become $O((|\Pi| + G_{\hat{g}}^2 + \sigma_{\hat{g}}^2)/\sqrt{N} + E_\Phi)$, introducing a constant bias in $O(E_\Phi)$. Note that the above discussion simplifies the model-based planning into a gradient step along the direction provided by Φ_n (which is not exactly [5, 6, 7, 8]). Nevertheless, it captures the important effect of performance bias, which also appears in these multi-step algorithms.

A hybrid heuristic to combine the model-based and model-free updates is DYNA [11, 12], which interleaves the two steps during policy optimization. This is equivalent to applying g_n, instead of the error e_n, in the Correction Step of PICCOLO. Following a similar analysis as above, one can show that the convergence rate in Theorem 4.2 would become $O((|\Pi| + G^2 + \sigma^2)/\sqrt{2N} + E_\Phi)$, where $G^2 = \frac{1}{2}(G_g^2 + G_{\hat{g}}^2)$ and $\sigma^2 = \frac{1}{2}(\sigma_g^2 + \sigma_{\hat{g}}^2)$. Therefore, DYNA is effectively twice as fast as the

[8] We provide another example of natural gradient descent in Appendix F.

[9] It can be shown that if the model is learned online with a no-regret algorithm, it would perform similarly to the best model in the hindsight (cf. Appendix H.4)

pure model-free approach. However, it would eventually suffer from the performance bias due model error, as reflected in the term E_Φ. We will demonstrate this property experimentally in Figure **??**.

Finally, we note that the idea of using Φ_n as control variate [15, 16, 17] is orthogonal to the setups considered above, and it can be naturally combined with PICCOLO. Specifically, we can use Φ_n to compute a better sampled gradient g_n with smaller variance (line 6 of Algorithm 1). This would improve σ_g^2 in the bounds to a smaller $\tilde{\sigma}_g^2$, the size of reduced variance.

C Proof of Lemma 3.3

Without loss of generality we suppose $w_1 = 1$ and $J(\pi) \geq 0$ for all π. And we assume the weighting sequence $\{w_n\}$ satisfies, for all $n \geq m \geq 1$ and $k \geq 0$, $\frac{w_{n+k}}{w_n} \leq \frac{w_{m+k}}{w_m}$. This means $\{w_n\}$ is an non-decreasing sequence and it does not grow faster than exponential (for which $\frac{w_{n+k}}{w_n} = \frac{w_{m+k}}{w_m}$). For example, if $w_n = n^p$ with $p \geq 0$, it easy to see that

$$\frac{(n+k)^p}{n^p} \leq \frac{(m+k)^p}{m^p} \iff \frac{n+k}{n} \leq \frac{m+k}{m} \iff \frac{k}{n} \leq \frac{k}{m}$$

For simplicity, let us first consider the case where f_n is deterministic. Given this assumption, we bound the performance in terms of the weighted regret below.

$$\begin{aligned}
&\sum_{n=1}^{N} w_n J(\pi_n) \\
&= \sum_{n=1}^{N} w_n J(\pi_{n-1}) + w_n \mathbb{E}_{d_{\pi_n}} \mathbb{E}_{\pi_n}[A_{\pi_{n-1}}] \\
&= \sum_{n=1}^{N} w_n J(\pi_{n-1}) + w_n f_n(\pi_n) \\
&= w_1 J(\pi_0) + \sum_{n=1}^{N-1} w_{n+1} J(\pi_n) + \sum_{n=1}^{N} w_n f_n(\pi_n) \\
&= w_1 J(\pi_0) + \sum_{n=1}^{N-1} w_{n+1} J(\pi_{n-1}) + \sum_{n=1}^{N-1} w_{n+1} f_n(\pi_n) + \sum_{n=1}^{N} w_n f_n(\pi_n) \\
&= (w_1 + w_2) J(\pi_0) + \sum_{n=1}^{N-2} w_{n+2} J(\pi_n) + \sum_{n=1}^{N-1} w_{n+1} f_n(\pi_n) + \sum_{n=1}^{N} w_n f_n(\pi_n) \\
&= w_{1:N} J(\pi_0) + \left(w_N f_1(\pi_1) + \sum_{n=1}^{2} w_{n+N-2} f_n(\pi_n) + \cdots + \sum_{n=1}^{N-1} w_{n+1} f_n(\pi_n) + \sum_{n=1}^{N} w_n f_n(\pi_n) \right) \\
&= w_{1:N} J(\pi_0) + \left(w_N f_1(\pi_1) + \sum_{n=1}^{2} \frac{w_{n+N-2}}{w_n} w_n f_n(\pi_n) + \cdots + \sum_{n=1}^{N-1} \frac{w_{n+1}}{w_n} w_n f_n(\pi_n) + \sum_{n=1}^{N} w_n f_n(\pi_n) \right) \\
&\leq w_{1:N} J(\pi_0) + \left(w_N f_1(\pi_1) + \frac{w_{N-1}}{w_1} \sum_{n=1}^{2} w_n f_n(\pi_n) + \cdots + \frac{w_2}{w_1} \sum_{n=1}^{N-1} w_n f_n(\pi_n) + \sum_{n=1}^{N} w_n f_n(\pi_n) \right) \\
&= w_{1:N} J(\pi_0) + \left(w_N f_1(\pi_1) + w_{N-1} \sum_{n=1}^{2} w_n f_n(\pi_n) + \cdots + w_2 \sum_{n=1}^{N-1} w_n f_n(\pi_n) + \sum_{n=1}^{N} w_n f_n(\pi_n) \right) \\
&= w_{1:N} J(\pi_0) + \sum_{n=1}^{N} w_{N-n+1} \left(\text{regret}_n(f) + w_{1:N} \epsilon_n(f) \right)
\end{aligned}$$

where the inequality is due to the assumption on the weighting sequence, and the last equality is due to the definition of regret_N and ϵ_N. For stochastic problems, because π_n does not depends on f_n, the

above bound applies to the performance in expectation. This proves the statement:

$$\mathbb{E}\left[\frac{1}{w_{1:N}}\sum_{n=1}^{N} w_n J(\pi_n)\right] \leq J(\pi_0) + \frac{1}{w_{1:N}}\sum_{n=1}^{N} w_{N-n+1}\mathbb{E}\left[\text{regret}_n(f) + w_{1:n}\epsilon_n(f)\right]$$

D The Basic Operations of Base Algorithms

In Section 4, we show that the update rule of any base mirror-descent or FTRL algorithm can be represented in terms of the three basic operations

$$h \leftarrow \texttt{update}(h, H, g, w), \qquad H \leftarrow \texttt{adapt}(h, H, g, w), \qquad \pi \leftarrow \texttt{project}(h, H) \tag{11}$$

and provide explicitly the identifications of `update` and `project`: for mirror descent,

$$\texttt{update}(h, H, g, w) = \arg\min_{\pi' \in \Pi} \langle wg, \pi' \rangle + B_H(\pi||h), \qquad \texttt{project}(h, H) = h \tag{13}$$

and for FTRL,

$$\texttt{update}(h, H, g, w) = h + wg, \qquad \texttt{project}(h, H) = \arg\min_{\pi' \in \Pi} \langle h, \pi' \rangle + H(\pi') \tag{14}$$

While `update` and `project` are defined standardly, the exact definition of `adapt` depends on the specific base algorithm. Particularly, `adapt` may depend also on whether the problem is weighted, as different base algorithms may handle weighted problems differently. Based on the way weighted problems are handled, we roughly categorize the algorithms (in both mirror descent and FTRL families) into two classes: the *stationary* regularization class and the *non-stationary* regularization class. Here we provide more details into the algorithm-dependent `adapt` operation, through some commonly used base algorithms as examples.

D.1 Stationary Regularization Class

The `adapt` operation of these base algorithms features two major functions: 1) a moving-average adaptation and 2) a step-size adaption. The moving-average adaptation is designed to estimate some statistics G such that $\|g\|_* = O(G)$ (which is an important factor in regret bounds), whereas the step-size adaptation updates a scalar multiplier η according to the weight w to ensure convergence.

This family of algorithms includes basic mirror descent [26] and FTRL [27, 37] with a scheduled step size, and adaptive algorithms based on moving average e.g. RMSPROP [48] ADADELTA [49], ADAM [25], AMSGRAD [50], and the adaptive NATGRAD we used in the experiments. Below we showcase how `adapt` is defined using some examples.

D.1.1 Basic mirror descent [26]

We define G to be some constant such that $G \geq \sup \|g_n\|_*$ and define

$$\eta_n = \frac{\eta}{1 + cw_{1:n}/\sqrt{n}}, \tag{17}$$

as a function of the iteration counter n, where $\eta > 0$ is a step size multiplier and $c > 0$ determines the decaying rate of the step size. The choice of hyperparameters η, c pertains to how far the optimal solution is from the initial condition, which is related to the size of Π. In implementation, `adapt` updates the iteration counter n and updates the multiplier η_n using w_n in (17).

Together (n, G, η_n) defines $H_n = R_n$ in the mirror descent update rule (9) through setting $R_n = \frac{G}{\eta_n} R$, where R is a strongly convex function. That is, we can write (9) equivalently as

$$\begin{aligned}
\pi_{n+1} &= \arg\min_{\pi \in \Pi} \langle w_n g_n, \pi \rangle + \frac{G}{\eta_n} B_R(\pi||\pi_n) \\
&= \arg\min_{\pi \in \Pi} \langle w_n g_n, \pi \rangle + B_{H_n}(\pi) \\
&= \texttt{update}(h_n, H_n, g_n, w_n)
\end{aligned}$$

When the weight is constant (i.e. $w_n = 1$), we can easily see this update rule is equivalent to the classical mirror descent with a step size $\frac{\eta/G}{1+c\sqrt{n}}$, which is the optimal step size [37]. For general

$w_n = \Theta(n^p)$ with some $p > -1$, it can viewed as having an effective step size $\frac{w_n \eta_n}{G} = O(\frac{1}{G\sqrt{n}})$, which is optimal in the weighted setting. The inclusion of the constant G makes the algorithm invariant to the scaling of loss functions. But as the same G is used across all the iterations, the basic mirror descent is conservative.

D.1.2 Basic FTRL [37]

Before discussing `adapt` in the basic FTRL, we first provide some more details into general FTRL by explicitly relating its `project` operation and the update rule in (10). We recall, in the nth iteration, the definition of h_n, H_n, and `project` of FTRL in (14) are

$$h_n = \sum_{m=1}^{n} w_m g_m, \qquad H_n(\pi) = \sum_{m=1}^{n} r_m(\pi||\pi_n), \qquad \texttt{project}(h, H) = \arg\min_{\pi' \in \Pi} \langle h, \pi' \rangle + H(\pi')$$

Therefore, we can see that $\pi_{n+1} = \texttt{project}(h_n, H_n)$ indeed gives the update (10):

$$\begin{aligned}
\pi_{n+1} &= \texttt{project}(h_n, H_n) \\
&= \texttt{project}(\sum_{m=1}^{n} w_m g_m, \sum_{m=1}^{n} r_m(\pi||\pi_n)) \\
&= \arg\min_{\pi \in \Pi} \sum_{m=1}^{n} \langle w_m g_m, \pi \rangle + r_m(\pi||\pi_m)
\end{aligned}$$

For the basic FTRL, the `adapt` operator is similar to the basic mirror descent, which uses a constant G and updates the memory (n, η_n) using (17). The main differences are how (G, η_n) is mapped to H_n and that the basic FTRL updates H_n also using h_n (i.e. π_n). Specifically, it performs $H_n \leftarrow \texttt{adapt}(h_n, H_{n-1}, g_n, w_n)$ through the following:

$$H_n(\cdot) = H_{n-1}(\cdot) + r_n(\cdot||\pi_n)$$

where following [37] we set

$$r_n(\pi||\pi_n) = G(\frac{1}{\eta_n} - \frac{1}{\eta_{n-1}}) B_R(\pi||\pi_n)$$

and η_n is updated using some scheduled rule.

One can also show that the choice of η_n scheduling in (17) leads to an optimal regret. When the problem is uniformly weighted (i.e. $w_n = 1$), this gives exactly the update rule in [37]. For general $w_n = \Theta(n^p)$ with $p > -1$, a proof of optimality can be found, for example, in the appendix of [23].

D.1.3 ADAM [25] and AMSGRAD [50]

As a representing mirror descent algorithm that uses moving-average estimates, ADAM keeps in the memory of the statistics of the first-order information that is provided in `update` and `adapt`. Here we first review the standard description of ADAM and then show how it is summarized in

$$H_n = \texttt{adapt}(h_n, H_{n-1}, g_n, w_n), \qquad h_{n+1} = \texttt{update}(h_n, H_n, g_n, w_n) \tag{12}$$

using properly constructed `update`, `adapt`, and `project` operations.

The update rule of ADAM proposed by Kingma and Ba [25] is originally written as, for $n \geq 1$,[10]

$$\begin{aligned}
m_n &= \beta_1 m_{n-1} + (1 - \beta_1) g_n \\
v_n &= \beta_2 v_{n-1} + (1 - \beta_2) g_n \odot g_n \\
\hat{m}_n &= m_n / (1 - \beta_1^n) \\
\hat{v}_n &= v_n / (1 - \beta_2^n) \\
\pi_{n+1} &= \pi_n - \eta_n \hat{m}_n \oslash (\sqrt{\hat{v}_n} + \epsilon)
\end{aligned} \tag{18}$$

[10] We shift the iteration index so it conforms with our notation in online learning, in which π_1 is the initial policy before any update.

where $\eta_n > 0$ is the step size, $\beta_1, \beta_2 \in [0, 1)$ (default $\beta_1 = 0.9$ and $\beta_2 = 0.999$) are the mixing rate, and $0 < \epsilon \ll 1$ is some constant for stability (default $\epsilon = 10^{-8}$), and $m_0 = v_0 = 0$. The symbols $\odot$ and $\oslash$ denote element-wise multiplication and division, respectively. The third and the forth steps are designed to remove the 0-bias due to running moving averages starting from 0.

The above update rule can be written in terms of the three basic operations. First, we define the memories $h_n = (m_n, \pi_n)$ for policy and (v_n, η_n, n) for regularization that is defined as

$$H_n(\pi) = \frac{1}{2\eta_n} \pi^\top (\mathrm{diag}(\sqrt{\hat{v}_n}) + \epsilon I)\pi \tag{19}$$

where $\hat{v}_n$ is defined in the original ADAM equation in (18).

The `adapt` operation updates the memory to (v_n, η_n, n) in the step

$$H_n \leftarrow \texttt{adapt}(h_n, H_{n-1}, g_n, w_n)$$

It updates the iteration counter n and η_n in the same way in the basic mirror descent using (17), and update v_n (which along with n defines $\hat{v}_n$ used in (19)) using the original ADAM equation in (18).

For `update`, we slightly modify the definition of `update` in (13) (replacing g_n with $\hat{m}_n$) to incorporate the moving average and write

$$\texttt{update}(h_n, H_n, g_n, w_n) = \arg\min_{\pi' \in \Pi} \langle w_n \hat{m}_n, \pi' \rangle + B_{H_n}(\pi' || \pi) \tag{20}$$

where m_n and $\hat{m}_n$ are defined the same as in the original ADAM equations in (18). One can verify that, with these definitions, the update rule in (12) is equivalent to the update rule (18), when the weight is uniform (i.e. $w_n = 1$).

Here the $\sqrt{\hat{v}_n}$ plays the role of G as in the basic mirror descent, which can be viewed as an estimate of the upper bound of $\|g_n\|_*$. ADAM achieves a better performance because a coordinate-wise online estimate is used. With this equivalence in mind, we can easily deduct that using the same scheduling of η_n as in the basic mirror descent would achieve an optimal regret (cf. [25, 50]). We note that ADAM may fail to converge in some particular problems due to the moving average [50]. AMSGRAD [50] modifies the moving average and uses strictly increasing estimates. However in practice AMSGRAD behaves more conservatively.

For weighted problems, we note one important nuance in our definition above: it separates the weight w_n from the moving average and considers w_n as part of the η_n update, because the growth of w_n in general can be much faster than the rate the moving average converges. In other words, the moving average can only be used to estimate a stationary property, not a time-varying one like w_n. Hence, we call this class of algorithms, the *stationary* regularization class.

D.1.4 Adaptive NATGRAD

Given first-order information g_n and weight w_n, we consider an update rule based on Fisher information matrix:

$$\pi_{n+1} = \arg\min_{\pi \in \Pi} \langle w_n g_n, \pi \rangle + \frac{\sqrt{\hat{G}_n}}{2\eta_n} (\pi - \pi_n)^\top F_n (\pi - \pi_n) \tag{21}$$

where F_n is the Fisher information matrix of policy π_n [51] and $\hat{G}_n$ is an adaptive multiplier for the step size which we will describe. When $\hat{G}_n = 1$, the update in (21) gives the standard natural gradient descent update with step size η_n [40] .

The role of $\hat{G}_n$ is to adaptively and *slowly* changes the step size to minimize $\sum_{n=1}^N \frac{\eta_n}{\sqrt{G_n}} \|g_n\|^2_{F_n,*}$, which plays an important part in the regret bound (see Section 4.3, Appendix G, and e.g. [37] for details). Following the idea in ADAM, we update $\hat{G}_n$ by setting (with $G_0 = 0$)

$$\begin{aligned} G_n &= \beta_2 G_n + (1 - \beta_2) \frac{1}{2} g_n^\top F_n^{-1} g_n \\ \hat{G}_n &= G_n / (1 - \beta_2^n) \end{aligned} \tag{22}$$

similar to the concept of updating v_n and $\hat{v}_n$ in ADAM in (18), and update η_n in the same way as in the basic mirror descent using (17). Consequently, this would also lead to a regret like ADAM but in terms of a different local norm.

The `update` operation of adaptive NATGRAD is defined standardly in (9) (as used in the experiments). The `adapt` operation updates n and η_n like in ADAM and updates G_n through (22).

D.2 Non-Stationary Regularization Class

The algorithms in the non-stationary regularization class maintains a regularization that is increasing over the number of iterations. Notable examples of this class include ADAGRAD [38] and ONLINE NEWTON STEP [52], and its regularization function is updated by applying BTL in a secondary online learning problem whose loss is an upper bound of the original regret (see [53] for details). Therefore, compared with the previous stationary regularization class, the adaption property of η_n and G_n exchanges: η_n here becomes constant and G_n becomes time-varying. This will be shown more clearly in the ADAGRAD example below. We note while these algorithms are designed to be optimal in the convex, they are often too conservative (e.g. decaying the step size too fast) for non-convex problems.

D.2.1 ADAGRAD

The update rule of the diagonal version of ADAGRAD in [38] is given as

$$\begin{aligned} G_n &= G_{n-1} + \operatorname{diag}(g_n \odot g_n) \\ \pi_{n+1} &= \arg\min_{\pi \in \Pi} \langle g_n, \pi \rangle + \frac{1}{2\eta}(\pi - \pi_n)^\top (\epsilon I + G_n)^{1/2}(\pi - \pi_n) \end{aligned} \tag{23}$$

where $G_0 = 0$ and $\eta > 0$ is a constant. ADAGRAD is designed to be optimal for online linear optimization problems. Above we provide the update equations of its mirror descent formulation in (23); a similar FTRL is also available (again the difference only happens when Π is constrained).

In terms of our notation, its `update` and `project` are defined standardly as in (13), i.e.

$$\texttt{update}(h_n, H_n, g_n, w_n) = \arg\min_{\pi' \in \Pi} \langle w_n g_n, \pi' \rangle + B_{H_n}(\pi' || \pi_n) \tag{24}$$

and its `adapt` essentially only updates G_n:

$$\texttt{adapt}(h_n, H_{n-1}, g_n, w_n) : G_n = G_{n-1} + \operatorname{diag}(w_n g_n \odot w_n g_n)$$

where the regularization is defined the updated G_n and the constant η as

$$H_n(\pi) = \frac{1}{2\eta} \pi^\top (\epsilon I + G_n)^{1/2} \pi.$$

One can simply verify the above definitions of `update` and `adapt` agrees with (23).

E A Practical Variation of PICCOLO

In Section 4.2, we show that, given a base algorithm in mirror descent/FTRL, PICCOLO generates a new first-order update rule by recomposing the three basic operations into

$$h_n = \texttt{update}(\hat{h}_n, H_{n-1}, \hat{g}_n, w_n) \qquad \text{[Prediction]} \tag{15}$$

$$\begin{aligned} H_n &= \texttt{adapt}(h_n, H_{n-1}, e_n, w_n) \\ \hat{h}_{n+1} &= \texttt{update}(h_n, H_n, e_n, w_n) \end{aligned} \qquad \text{[Correction]} \tag{16}$$

where $e_n = g_n - \hat{g}_n$ and $\hat{g}_n$ is an estimate of g_n given by a predictive model Φ_n.

Here we propose a slight variation which introduces another operation `shift` inside the Prediction Step. This leads to the new set of update rules:

$$\begin{aligned} \hat{H}_n &= \texttt{shift}(\hat{h}_n, H_{n-1}) \\ h_n &= \texttt{update}(\hat{h}_n, \hat{H}_n, \hat{g}_n, w_n) \end{aligned} \qquad \text{[Prediction]} \tag{25}$$

$$\begin{aligned} H_n &= \texttt{adapt}(h_n, \hat{H}_n, e_n, w_n) \\ \hat{h}_{n+1} &= \texttt{update}(h_n, H_n, e_n, w_n) \end{aligned} \qquad \text{[Correction]} \tag{26}$$

The new `shift` operator additionally changes the regularization based on $\hat{h}_n$ the current representation of the policy in the Prediction Step, *independent* of the predicted gradient $\hat{g}_n$ and weight w_n.

The main purpose of including this additional step is to deal with numerical difficulties, such as singularity of H_n. For example, in natural gradient descent, the Fisher information of some policy can be close to being singular along the direction of the gradients that are evaluated at different policies. As a result, in the original Prediction Step of PICCOLO, H_{n-1} which is evaluated at π_{n-1} might be singular in the direction of $\hat{g}_n$ which is evaluated $\hat{\pi}_n$.

The new operator `shift` brings in an extra degree of freedom to account for such issue. Although from a theoretical point of view (cf. Appendix G) the use of `shift` would only increase regrets and should be avoided if possible, in practice, its merits in handling numerical difficulties can out weight the drawback. Because `shift` does not depend on the size of $\hat{g}_n$ and w_n, the extra regrets would only be proportional to $O(\sum_{n=1}^{N} \|\pi_n - \hat{\pi}_n\|_n)$, which can be smaller than other terms in the regret bound (see Appendix G).

In the experiments of NATGRAD and TRPO, $\hat{H}_n$ is defined by the Fisher information matrix evaluated at $\hat{\pi}_n$ and H_n is the average of the Fisher information matrices at $\hat{\pi}_n$ and π_n. But we note that the scalar multiplier that defines *magnitude* of the regularization is only updated in the Correction Step using g_n and w_n.

F Example: PICCOLOing Natural Gradient Descent

We give an alternative example to illustrate how one can use the above procedure to "PICCOLO" a base algorithm into a new algorithm. Here we consider the adaptive natural gradient descent rule in Appendix D as the base algorithm, which (given first-order information g_n and weight w_n) updates the policy through

$$\pi_{n+1} = \arg\min_{\pi\in\Pi} \langle w_n g_n, \pi\rangle + \frac{\sqrt{\hat{G}_n}}{2\eta_n}(\pi - \pi_n)^\top F_n(\pi - \pi_n) \tag{27}$$

where F_n is the Fisher information matrix of policy π_n [51], η_n a scheduled learning rate, and $\hat{G}_n$ is an adaptive multiplier for the step size which we will shortly describe. When $\hat{G}_n = 1$, the update in (27) gives the standard natural gradient descent update with step size η_n [40] .

The role of $\hat{G}_n$ is to adaptively and *slowly* changes the step size to minimize $\sum_{n=1}^{N} \frac{\eta_n}{\sqrt{G_n}} \|g_n\|^2_{F_n,*}$, which plays an important part in the regret bound (see Section 4.3, Appendix G, and e.g. [37] for details). To this end, we update $\hat{G}_n$ by setting (with $G_0 = 0$)

$$G_n = \beta_2 G_{n-1} + (1-\beta_2)\tfrac{1}{2} g_n^\top F_n^{-1} g_n, \qquad \hat{G}_n = G_n/(1-\beta_2^n) \tag{28}$$

similar to the moving average update rule in ADAM, and update η_n in the same way as in the basic mirror descent algorithm (e.g. $\eta_n = O(1/\sqrt{n})$). As a result, this leads to a similar regret like ADAM with $\beta_1 = 0$, but in terms of a local norm specified by the Fisher information matrix.

Now, let's see how to PICCOLO the adaptive natural gradient descent rule above. (For simplicity, we ignore potential singularities and consider the basic version in (15) and (16)). First, it is easy to see that the adaptive natural gradient descent rule is an instance of mirror descent (with $h_n = \pi_n$ and $H_n(g) = \frac{\sqrt{\hat{G}_n}}{2\eta_n} g^\top F_n g$), so the `update` and `project` operations are defined in the standard way, as in Section 4.2. The `adapt` operation updates the iteration counter n, the learning rate η_n, and updates $\hat{G}_n$ through (28).

To be more specific, let us explicitly write out the Prediction Step and the Correction Step of the PICCOLOed adaptive natural gradient descent rule in closed form as below: e.g. if $\eta_n = \frac{1}{\sqrt{n}}$, then we can write them as

[Prediction]
$$\pi_n = \arg\min_{\pi\in\Pi} \langle w_n \hat{g}_n, \pi\rangle + \frac{\sqrt{\hat{G}_{n-1}}}{2\eta_{n-1}}(\pi - \hat{\pi}_n)^\top F_{n-1}(\pi - \hat{\pi}_n)$$

[Correction]
$$\begin{aligned} \eta_n &= 1/\sqrt{n} \\ G_n &= \beta_2 G_{n-1} + (1-\beta_2)\tfrac{1}{2} g_n^\top F_n^{-1} g_n \\ \hat{G}_n &= G_n/(1-\beta_2^n) \\ \hat{\pi}_{n+1} &= \arg\min_{\pi\in\Pi} \langle w_n e_n, \pi\rangle + \frac{\sqrt{\hat{G}_n}}{2\eta_n}(\pi - \pi_n)^\top F_n(\pi - \pi_n) \end{aligned}$$

G Regret Analysis of PICCOLO

The main idea of PICCOLO is to achieve optimal performance in predictable online learning problems by *reusing* existing adaptive, optimal first-order algorithms that are designed for adversarial online learning problems. This is realized by the reduction techniques presented in this section.

Here we prove the performance of PICCOLO in general predictable online learning problems, independent of the context of policy optimization. In Appendix G.1, we first show an elegant reduction from predictable problems to adversarial problems. Then we prove Theorem 4.1 in Appendix G.2, showing how the optimal regret bound for predictable linear problems can be achieved by PICCOLOing mirror descent and FTRL algorithms. Note that we will abuse the notation l_n to denote the per-round costs in this general setting.

G.1 Reduction from Predictable Online Learning to Adversarial Online Learning

Consider a predictable online learning problem with per-round loss l_n. Suppose in round n, before playing π_n and revealing $_n$, we have access to some prediction of l_n, called $\hat{l}_n$. Running an (adaptive) online learning algorithm designed for the general adversarial setting is not optimal here, as its regret would be in $O(\sum_{n=1}^N \|\nabla l_n\|_{n,*}^2)$, where $\|\cdot\|_n$ is some local norm chosen by the algorithm and $\|\cdot\|_{n,*}$ is its dual norm. Ideally, we would only want to pay for the information that is unpredictable. Specifically, we wish to achieve an optimal regret in $O(\sum_{n=1}^N \|\nabla l_n - \nabla \hat{l}_n\|_{n,*}^2)$ instead [24].

To achieve the optimal regret bound yet without referring to specialized, nested two-step algorithms (e.g. mirror-prox [32], optimistic mirror descent [54], FTRL-prediction [24]), we consider decomposing a *predictable* problem with N rounds into an *adversarial* problem with $2N$ rounds:

$$\sum_{n=1}^N l_n(\pi_n) = \sum_{n=1}^N \hat{l}_n(\pi_n) + e_n(\pi_n) \tag{29}$$

Therefore, we can treat the predictable problem as a new adversarial online learning problem with a loss sequence $\hat{l}_1, e_1, \hat{l}_2, e_2, \ldots, \hat{l}_N, e_N$ and consider solving this new problem with some standard online learning algorithm designed for the adversarial setting.

Before analysis, we first introduce a new decision variable $\hat{\pi}_n$ and denote the decision sequence in this new problem as $\hat{\pi}_1, \pi_1, \hat{\pi}_2, \pi_2, \ldots, \hat{\pi}_N, \pi_N$, so the definition of the variables are consistent with that in the problem before. Because this new problem is unpredictable, the optimal regret of this new decision sequence is

$$\sum_{n=1}^N \hat{l}_n(\hat{\pi}_n) + e_n(\pi_n) - \min_{\pi\in\Pi} \sum_{n=1}^N \hat{l}_n(\pi) + e_n(\pi) = O(\sum_{n=1}^N \|\nabla \hat{l}_n\|_{n,*}^2 + \|\nabla e_n\|_{n+1/2,*}^2) \tag{30}$$

where the subscript $n+1/2$ denotes the extra round due to the reduction.

At first glance, our reduction does not meet the expectation of achieving regret in $O(\sum_{n=1}^N \|\nabla l_n - \nabla \hat{l}_n\|_{n,*}^2) = O(\sum_{n=1}^N \|\nabla e_n\|_{n,*}^2)$. However, we note that the regret for the new problem is too loose for the regret of the original problem, which is

$$\sum_{n=1}^N \hat{l}_n(\pi_n) + e_n(\pi_n) - \min_{\pi\in\Pi} \sum_{n=1}^N \hat{l}_n(\pi) + e_n(\pi)$$

where the main difference is that originally we care about $\hat{l}_n(\pi_n)$ rather than $\hat{l}_n(\hat{\pi}_n)$. Specifically, we can write

$$\begin{aligned}\sum_{n=1}^N l_n(\pi_n) &= \sum_{n=1}^N \hat{l}_n(\pi_n) + e_n(\pi_n) \\ &= \left(\sum_{n=1}^N \hat{l}_n(\hat{\pi}_n) + e_n(\pi_n)\right) + \left(\sum_{n=1}^N \hat{l}_n(\pi_n) - \hat{l}_n(\hat{\pi}_n)\right)\end{aligned}$$

Therefore, if the update rule for generating the decision sequence $\hat{\pi}_1, \pi_1, \hat{\pi}_2, \pi_2, \ldots, \hat{\pi}_N, \pi_N$ contributes sufficient negativity in the term $\hat{l}_n(\pi_n) - \hat{l}_n(\hat{\pi}_n)$ compared with the regret of the new adversarial problem, then the regret of the original problem can be smaller than (30). This is potentially possible, as π_n is made after $\hat{l}_n$ is revealed. In particular, in the next section, we show that when the base algorithm, which is adopted to solve the new adversarial problem given by the reduction, is in the family of mirror descent and FTRL. Then the regret bound of PICCOLO with respect to the original predictable problem is optimal.

G.2 Optimal Regret Bounds for Predictable Problems

We show that if the base algorithm of PICCOLO belongs to the family of optimal mirror descent and FTRL designed for adversarial problems, then PICCOLO can achieve the optimal regret of predictable problems. In this subsection, we assume the loss sequence is linear, i.e. $l_n(\pi) = \langle \nabla l_n, \pi \rangle$, and the results are summarized as Theorem 4.1 in the main paper (in a slightly different notation).

G.2.1 Mirror Descent

First, we consider the case where the base algorithm is mirror descent. In this case, when can write the PICCOLO update rule as

$$\pi_n = \arg\min_{\pi \in \Pi} \left\langle \nabla \hat{l}_n, x \right\rangle + B_{H_{n-1}}(\pi || \hat{\pi}_n) \qquad \text{[Prediction]}$$

$$\hat{\pi}_{n+1} = \arg\min_{\pi \in \Pi} \langle \nabla e_n, \pi \rangle + B_{H_n}(\pi || \pi_n) \qquad \text{[Correction]}$$

where H_n can be updated based on $\nabla e_n = \nabla l_n(\pi_n) - \nabla \hat{l}_n(\hat{\pi}_n)$. Notice that in the Prediction Step, PICCOLO uses the regularization from the previous Correction Step.

To analyze the performance, we use a lemma of the mirror descent's properties. The proof is a straightforward application of the optimality condition of the proximal map [55]. We provide a proof here for completeness.

Lemma G.1. *Let $\mathcal{K}$ be a convex set. Suppose R is 1-strongly convex with respect to norm $\|\cdot\|$. Let g be a vector in some Euclidean space and let*

$$y = \arg\min_{z \in \mathcal{K}} \langle g, z \rangle + \frac{1}{\eta} B_R(z||x)$$

Then for all $z \in \mathcal{K}$

$$\eta \langle g, y - z \rangle \leq B_R(z||x) - B_R(z||y) - B_R(y||x) \tag{31}$$

which implies

$$\eta \langle g, x - z \rangle \leq B_R(z||x) - B_R(z||y) + \frac{\eta^2}{2} \|g\|_*^2 \tag{32}$$

Proof. Recall the definition $B_R(z||x) = R(z) - R(x) - \langle \nabla R(x), z - x \rangle$. The optimality of the proximal map can be written as

$$\langle \eta g + \nabla R(y) - \nabla R(x), y - z \rangle \leq 0, \qquad \forall z \in \mathcal{K}$$

By rearranging the terms, we can rewrite the above inequality in terms Bregman divergences as follows and derive the first inequality (31):

$$\begin{aligned}
\langle \eta g, y - z \rangle &\leq \langle \nabla R(x) - \nabla R(y), y - z \rangle \\
&= B_R(z||x) - B_R(z||y) + \langle \nabla R(x) - \nabla R(y), y \rangle - \langle \nabla R(x), x \rangle + \langle \nabla R(y), y \rangle + R(x) - R(y) \\
&= B_R(z||x) - B_R(z||y) + \langle \nabla R(x), y - x \rangle + R(x) - R(y) \\
&= B_R(z||x) - B_R(z||y) - B_R(y||x)
\end{aligned}$$

The second inequality is the consequence of (31). First, we rewrite (31) as

$$\langle \eta g, x - z \rangle = B_R(z||x) - B_R(z||y) - B_R(y||x) + \langle \eta g, x - y \rangle$$

Then we use the fact that B_R is 1-strongly convex with respect to $\|\cdot\|$, which implies

$$-B_R(y||x) + \langle \eta g, x - y \rangle \leq -\frac{1}{2}\|x - y\|^2 + \langle \eta g, x - y \rangle \leq \frac{\eta^2}{2} \|g\|_*^2$$

Combining the two inequalities yields (32). ■

Lemma G.1 is usually stated with (32), which concerns the decision made before seeing the per-round loss (as in the standard adversarial online learning setting). Here, we additionally concern $\hat{l}_n(\pi_n)$, which is the decision made after seeing $\hat{l}_n$, so we need a tighter bound (31).

Now we show that the regret bound of PICCOLO in the predictable linear problems when the base algorithm is mirror descent.

Proposition G.1. *Assume the base algorithm of* PICCOLO *is mirror descent satisfying the Assumption 4.1. Then it holds that, for any* $\pi \in \Pi$*,*

$$\sum_{n=1}^{N} w_n \left\langle \nabla l_n, \pi_n - \pi \right\rangle \leq M_N + \sum_{n=1}^{N} \frac{w_n^2}{2} \|\nabla e_n\|_{*,n}^2 - \frac{1}{2}\|\pi_n - \hat{\pi}_n\|_{n-1}^2$$

Proof. Suppose R_n, which is defined by H_n, is 1-strongly convex with respect to $\|\cdot\|_n$. Then by Lemma G.1, we can write, for all $\pi \in \Pi$,

$$\begin{aligned} w_n \left\langle \nabla l_n, \pi_n - \pi \right\rangle &= w_n \left\langle \nabla \hat{l}_n, \pi_n - \pi \right\rangle + w_n \left\langle \nabla e_n, \pi_n - \pi \right\rangle \\ &\leq B_{R_{n-1}}(\pi||\hat{\pi}_n) - B_{R_{n-1}}(\pi||\pi_n) - B_{R_{n-1}}(\pi_n||\hat{\pi}_n) \\ &\quad + B_{R_n}(\pi||\pi_n) - B_{R_n}(\pi||\hat{\pi}_{n+1}) + \frac{w_n^2}{2}\|\nabla e_n\|_{*,n}^2 \end{aligned} \tag{33}$$

where we use (31) for the loss $\hat{l}_n$ and (32) for the loss e_n.

To show the regret bound of the original (predictable) problem, we first notice that

$$\begin{aligned} &\sum_{n=1}^{N} B_{R_{n-1}}(\pi||\hat{\pi}_n) - B_{R_{n-1}}(\pi||\pi_n) + B_{R_n}(\pi||\pi_n) - B_{R_n}(\pi||\hat{\pi}_{n+1}) \\ &= B_{R_0}(\pi||\hat{\pi}_1) - B_{R_N}(\pi||\hat{\pi}_{N+1}) + \sum_{n=1}^{N} B_{R_{n-1}}(\pi||\hat{\pi}_n) - B_{R_{n-1}}(\pi||\pi_n) + B_{R_n}(\pi||\pi_n) - B_{R_{n-1}}(\pi||\hat{\pi}_n) \\ &= B_{R_0}(\pi||\hat{\pi}_1) - B_{R_N}(\pi||\hat{\pi}_{N+1}) + \sum_{n=1}^{N} B_{R_n}(\pi||\pi_n) - B_{R_{n-1}}(\pi||\pi_n) \leq M_N \end{aligned}$$

where the last inequality follows from the assumption on the base algorithm. Therefore, by telescoping the inequality in (33) and using the strong convexity of R_n, we get

$$\begin{aligned} \sum_{n=1}^{N} w_n \left\langle \nabla l_n, \pi_n - \pi \right\rangle &\leq M_N + \sum_{n=1}^{N} \frac{w_n^2}{2}\|\nabla e_n\|_{*,n}^2 - B_{R_{n-1}}(\pi_n||\hat{\pi}_n) \\ &\leq M_N + \sum_{n=1}^{N} \frac{w_n^2}{2}\|\nabla e_n\|_{*,n}^2 - \frac{1}{2}\|\pi_n - \hat{\pi}_n\|_{n-1}^2 \end{aligned}$$

■

G.2.2 Follow-the-Regularized-Leader

We consider another type of base algorithm, FTRL, which is mainly different from mirror descent in the way that constrained decision sets are handled [37]. In this case, the exact update rule of PICCOLO can be written as

$$\pi_n = \operatorname*{arg\,min}_{\pi \in \Pi} \left\langle w_n \nabla \hat{l}_n, \pi \right\rangle + \sum_{m=1}^{n-1} \left\langle w_m \nabla l_m, \pi \right\rangle + B_{r_m}(\pi||\pi_m) \qquad \text{[Prediction]}$$

$$\hat{\pi}_{n+1} = \operatorname*{arg\,min}_{\pi \in \Pi} \sum_{m=1}^{n} \left\langle w_m \nabla l_m, \pi \right\rangle + B_{r_m}(\pi||\pi_m) \qquad \text{[Correction]}$$

From the above equations, we verify that MOBIL [23] is indeed a special case of PICCOLO, when the base algorithm is FTRL.

We show PICCOLO with FTRL has the following guarantee.

Proposition G.2. *Assume the base algorithm of* PICCOLO *is FTRL satisfying the Assumption 4.1. Then it holds that, for any* $\pi \in \Pi$*,*

$$\sum_{n=1}^{N} w_n \left\langle \nabla l_n, \pi_n - \pi \right\rangle \leq M_N + \sum_{n=1}^{N} \frac{w_n^2}{2} \|\nabla e_n\|_{*,n}^2 - \frac{1}{2}\|\pi_n - \hat{\pi}_n\|_{n-1}^2$$

We show the above results of PICCOLO using a different technique from [23]. Instead of developing a specialized proof like they do, we simply use the properties of FTRL on the $2N$-step new adversarial problem!

To do so, we recall some facts of the base algorithm FTRL. First, FTRL in (10) is equivalent to Follow-the-Leader (FTL) on a surrogate problem with the per-round loss is $\langle g_n, \pi \rangle + B_{r_n}(\pi||\pi_n)$. Therefore, the regret of FTRL can be bounded by the regret of FTL in the surrogate problem plus the size of the additional regularization $B_{r_n}(\pi||\pi_n)$. Second, we recall a standard techniques in proving FTL, called Strong FTL Lemma (see e.g. [37]), which is proposed for *adversarial* online learning.

Lemma G.2 (Strong FTL Lemma [37])**.** *For any sequence* $\{\pi_n \in \Pi\}$ *and* $\{l_n\}$*,*

$$\text{regret}_N(l) \coloneqq \sum_{n=1}^{N} l_n(\pi_n) - \min_{\pi \in \Pi} \sum_{n=1}^{N} l_n(\pi) \leq \sum_{n=1}^{N} l_{1:n}(\pi_n) - l_{1:n}(\pi_n^\star)$$

where $\pi_n^\star \in \arg\min_{\pi \in \Pi} l_{1:n}(\pi)$.

Using the decomposition idea above, we show the performance of PICCOLO following sketch below: first, we show a bound on the regret in the surrogate predictable problem with per-round loss $\langle \nabla l_n, \pi \rangle + B_{r_n}(\pi||\pi_n)$; second, we derive the bound for the original predictable problem with per-round loss $\langle \nabla l_n, \pi \rangle$ by considering the effects of $B_{r_n}(\pi||\pi_n)$. We will prove the first step by applying FTL on the transformed $2N$-step adversarial problem of the original N-step predictable surrogate problem and then showing that PICCOLO achieves the optimal regret in the original N-step predictable surrogate problem. Interestingly, we recover the bound in the stronger FTL Lemma (Lemma G.3) by Cheng et al. [23], which they suggest is necessary for proving the improved regret bound of their FTRL-prediction algorithm (MOBIL).

Lemma G.3 (Stronger FTL Lemma [23])**.** *For any sequence* $\{\pi_n\}$ *and* $\{l_n\}$*,*

$$\text{regret}_N(l) = \sum_{n=1}^{N} l_{1:n}(\pi_n) - l_{1:n}(\pi_n^\star) - \Delta_n$$

where $\Delta_{n+1} \coloneqq l_{1:n}(\pi_{n+1}) - l_{1:n}(\pi_n^\star) \geq 0$ *and* $\pi_n^\star \in \arg\min_{\pi \in \Pi} l_{1:n}(\pi)$.

Our new reduction-based regret bound is presented below.

Proposition G.3. *Let* $\{l_n\}$ *be a predictable loss sequence with predictable information* $\{\hat{l}_n\}$*. Suppose the decision sequence* $\hat{\pi}_1, \pi_1, \hat{\pi}_2, \ldots, \hat{\pi}_N, \pi_N$ *is generated by running FTL on the transformed adversarial loss sequence* $\hat{l}_1, e_1, \hat{l}_2, \ldots, \hat{l}_N, e_N$*, then the bound in the Stonger FTL Lemma holds. That is,* $\text{regret}_N(l) \leq \sum_{n=1}^{N} l_{1:n}(\pi_n) - l_{1:n}(\pi_n^\star) - \Delta_n$*, where* $\Delta_{n+1} \coloneqq l_{1:n}(\pi_{n+1}) - l_{1:n}(\pi_n^\star) \geq 0$ *and* $\pi_n^\star \in \arg\min_{\pi \in \Pi} l_{1:n}(\pi)$.

Proof. First, we transform the loss sequence and write

$$\sum_{n=1}^{N} l_n(\pi_n) = \sum_{n=1}^{N} \hat{l}_n(\pi_n) + e_n(\pi_n) = \left(\sum_{n=1}^{N} \hat{l}_n(\hat{\pi}_n) + e_n(\pi_n) \right) + \left(\sum_{n=1}^{N} \hat{l}_n(\pi_n) - \hat{l}_n(\hat{\pi}_n) \right)$$

Then we apply standard Strong FTL Lemma on the new adversarial problem in the left term.

$$\begin{aligned}
&\sum_{n=1}^{N} \hat{l}_n(\hat{\pi}_n) + e_n(\pi_n) \\
&\leq \sum_{n=1}^{N} (\hat{l} + e)_{1:n}(\pi_n) - \min_{\pi \in \Pi} (\hat{l} + e)_{1:n}(\pi) + \sum_{n=1}^{N} ((\hat{l} + e)_{1:n-1} + \hat{l}_n)(\hat{\pi}_n) - \min_{\pi \in \Pi} ((\hat{l} + e)_{1:n-1} + \hat{l}_n)(\pi) \\
&= \sum_{n=1}^{N} l_{1:n}(\pi_n) - \min_{\pi \in \Pi} l_{1:n}(\pi) + \sum_{n=1}^{N} (l_{1:n-1} + \hat{l}_n)(\hat{\pi}_n) - (l_{1:n-1} + \hat{l}_n)(\pi_n)
\end{aligned}$$

where the first inequality is due to Strong FTL Lemma and the second equality is because FTL update assumption.

Now we observe that if we add the second term above and $\sum_{n=1}^N \hat{l}_n(\pi_n) - \hat{l}_n(\hat{\pi}_n)$ together, we have

$$\begin{aligned}
&\sum_{n=1}^N (l_{1:n-1} + \hat{l}_n)(\hat{\pi}_n) - (l_{1:n-1} + \hat{l}_n)(\pi_n) + (\hat{l}_n(\pi_n) - \hat{l}_n(\hat{\pi}_n)) \\
&= \sum_{n=1}^N (l_{1:n-1})(\hat{\pi}_n) - l_{1:n-1}(\pi_n) = \Delta_n
\end{aligned}$$

Thus, combing previous two inequalities, we have the bound in the Stronger FTL Lemma:

$$\sum_{n=1}^N l_n(\pi_n) \le \sum_{n=1}^N l_{1:n}(\pi_n) - \min_{\pi \in \Pi} l_{1:n}(\pi) - \Delta_n \qquad \blacksquare$$

Using Proposition G.3, we can now bound the regret of PICCOLO in Proposition G.2 easily.

Proof of Proposition G.2. Suppose $\sum_{m=1}^n B_{r_m}(\cdot||\pi_m)$ is 1-strongly convex with respect to some norm $\|\cdot\|_n$. Let $f_n = \langle w_n \nabla l_n, \pi_n \rangle + B_{r_n}(\pi||\pi_m)$[11]. Then by a simple convexity analysis (see e.g. see [37]) and Proposition G.3, we can derive

$$\begin{aligned}
\text{regret}_N(f) &\le \sum_{n=1}^N (f_{1:n}(\pi_n) - \min_{\pi \in \Pi} f_{1:n}(\pi)) - (f_{1:n-1}(\pi_n) - f_{1:n-1}(\hat{\pi}_n)) \\
&\le \sum_{n=1}^N \frac{w_n^2}{2} \|\nabla e_n\|_{n,*}^2 - \frac{1}{2}\|\pi_n - \hat{\pi}_n\|_{n-1}^2
\end{aligned}$$

Finally, because r_n is proximal (i.e. $B_{r_n}(\pi_n||\pi_n) = 0$), we can bound the original regret: for any $\pi \in \Pi$, it satisfies that

$$\begin{aligned}
\sum_{n=1}^N w_n \langle \nabla l_n, \pi_n - \pi \rangle &\le \sum_{n=1}^N f_n(\pi_n) - f_n(\pi) + B_{r_n}(\pi||\pi_n) \\
&\le M_N + \sum_{n=1}^N \frac{w_n^2}{2} \|\nabla e_n\|_{*,n}^2 - \frac{1}{2}\|\pi_n - \hat{\pi}_n\|_{n-1}^2
\end{aligned}$$

where we use Assumption 4.1 and the bound of $\text{regret}_N(f)$ in the second inequality. $\blacksquare$

H Policy Optimization Analysis of PICCOLO

In this section, we discuss how to interpret the bound given in Theorem 4.1

$$\sum_{n=1}^N w_n \langle g_n, \pi_n - \pi \rangle \le M_N + \sum_{n=1}^N \frac{w_n^2}{2} \|e_n\|_{*,n}^2 - \frac{1}{2}\|\pi_n - \hat{\pi}_n\|_{n-1}^2$$

in the context of policy optimization and show exactly how the optimal bound

$$\mathbb{E}\left[\sum_{n=1}^N \langle w_n g_n, \pi_n - \pi \rangle\right] \le O(1) + C_{\Pi,\Phi} \frac{w_{1:N}}{\sqrt{N}} \tag{34}$$

is derived. We will discuss how model learning can further help minimize the regret bound later in Appendix H.4.

[11] Again we overload the notation; f_n does not refer to the per-round loss in (7).

H.1 Assumptions

We introduce some assumptions to characterize the sampled gradient g_n. Recall $g_n = \nabla \tilde{h}_n(\pi_n)$ for IL and $g_n = \nabla \tilde{f}_n(\pi_n)$ for RL.

Assumption H.1. *$\|\mathbb{E}[g_n]\|_*^2 \le G_g^2$ and $\|g_n - \mathbb{E}[g_n]\|_*^2 \le \sigma_g^2$ for some finite constants G_g and σ_g.*

Similarly, we consider properties of the predictive model Φ_n that is used to estimate the gradient of the next per-round loss. Let $\mathcal{P}$ denote the class of these models (i.e. $\Phi_n \in \mathcal{P}$), which can potentially be *stochastic*. We make assumptions on the size of $\hat{g}_n$ and its variance.

Assumption H.2. *$\|\mathbb{E}[\hat{g}_n]\|_*^2 \le G_{\hat{g}}^2$ and $\mathbb{E}[\|\hat{g}_n - \mathbb{E}[\hat{g}_n]\|_*^2] \le \sigma_{\hat{g}}^2$ for some finite constants $G_{\hat{g}}$ and $\sigma_{\hat{g}}$.*

Additionally, we assume these models are Lipschitz continuous.

Assumption H.3. *There is a constant $L \in [0, \infty)$ such that, for any instantaneous cost ψ and any $\Phi \in \mathcal{P}$, it satisfies $\|\mathbb{E}[\Phi(\pi, \psi)] - \mathbb{E}[\Phi(\pi', \psi)]\|_* \le L\|\pi - \pi'\|$.*

Lastly, as PICCOLO is agnostic to the base algorithm, we assume the local norm $\|\cdot\|_n$ chosen by the base algorithm at round n satisfies $\|\cdot\|_n^2 \ge \alpha_n \|\cdot\|^2$ for some $\alpha_n > 0$. This condition implies that $\|\cdot\|_{n,*}^2 \le \frac{1}{\alpha_n}\|\cdot\|_*^2$. In addition, we assume α_n is non-decreasing so that $M_N = O(\alpha_N)$ in Assumption 4.1, where the leading constant in the bound $O(\alpha_N)$ is proportional to $|\Pi|$, as commonly chosen in online convex optimization.

H.2 A Useful Lemma

We study the bound in Theorem 4.1 under the assumptions made in the previous section. We first derive a basic inequality, following the idea in [23, Lemma 4.3].

Lemma H.1. *Under Assumptions H.1, H.2, and H.3, it holds*

$$\mathbb{E}[\|e_n\|_{*,n}^2] = \mathbb{E}[\|g_n - \hat{g}_n\|_{*,n}^2] \le \frac{4}{\alpha_n}\left(\sigma_g^2 + \sigma_{\hat{g}}^2 + L_n^2\|\pi_n - \hat{\pi}_n\|_n^2 + E_n(\Phi_n)\right)$$

where $E_n(\Phi_n) = \|\mathbb{E}[g_n] - \mathbb{E}[\Phi_n(\pi_n, \psi_n)]\|_*^2$ is the prediction error of model Φ_n.

Proof. Recall $\hat{g}_n = \Phi_n(\hat{\pi}_n, \psi_n)$. Using the triangular inequality, we can simply derive

$$\begin{aligned}
&\mathbb{E}[\|g_n - \hat{g}_n\|_{*,n}^2] \\
&\le 4\left(\mathbb{E}[\|g_n - \mathbb{E}[g_n]\|_{*,n}^2] + \|\mathbb{E}[g_n] - \mathbb{E}[\Phi_n(\pi_n, \psi_n)]\|_{*,n}^2 + \|\mathbb{E}[\Phi_n(\pi_n, \psi_n)] - \mathbb{E}[\hat{g}_n]\|_{*,n}^2 + \mathbb{E}[\|\mathbb{E}[\hat{g}_n] - \hat{g}_n\|_{*,n}^2]\right) \\
&= 4\left(\mathbb{E}[\|g_n - \mathbb{E}[g_n]\|_{*,n}^2] + \|\mathbb{E}[g_n] - \mathbb{E}[\Phi_n(\pi_n, \psi_n)]\|_{*,n}^2 + \|\mathbb{E}[\Phi_n(\pi_n, \psi_n)] - \mathbb{E}[\Phi_n(\hat{\pi}_n, \psi_n)]\|_{*,n}^2 + \mathbb{E}[\|\mathbb{E}[\hat{g}_n] - \hat{g}_n\|_{*,n}^2]\right) \\
&\le 4\left(\frac{1}{\alpha_n}\sigma_g^2 + \frac{1}{\alpha_n}E_n(\Phi_n) + \|\mathbb{E}[\Phi_n(\pi_n, \psi_n)] - \mathbb{E}[\Phi_n(\hat{\pi}_n, \psi_n)]\|_{*,n}^2 + \frac{1}{\alpha_n}\sigma_{\hat{g}}^2\right) \\
&\le \frac{4}{\alpha_n}\left(\sigma_g^2 + \sigma_{\hat{g}}^2 + L^2\|\pi_n - \hat{\pi}_n\|_n^2 + E_n(\Phi_n)\right)
\end{aligned}$$

where the last inequality is due to Assumption H.3. ∎

H.3 Optimal Regret Bounds

We now analyze the regret bound in Theorem 4.1

$$\sum_{n=1}^{N} w_n \langle g_n, \pi_n - \pi\rangle \le M_N + \sum_{n=1}^{N} \frac{w_n^2}{2}\|e_n\|_{*,n}^2 - \frac{1}{2}\|\pi_n - \hat{\pi}_n\|_{n-1}^2 \tag{35}$$

We first gain some intuition about the size of

$$M_N + \mathbb{E}\left[\sum_{n=1}^{N} \frac{w_n^2}{2}\|e_n\|_{*,n}^2\right]. \tag{36}$$

Because when $\texttt{adapt}(h_n, H_{n-1}, e_n, w_n)$ is called in the Correction Step in (16) with the error gradient e_n as input, an optimal base algorithm (e.g. all the base algorithms listed in Appendix D)

would choose a local norm sequence $\|\cdot\|_n$ such that (36) is optimal. For example, suppose $\|e_n\|_*^2 = O(1)$ and $w_n = n^p$ for some $p > -1$. If the base algorithm is basic mirror descent (cf. Appendix D), then $\alpha_n = O(\frac{w_{1:n}}{\sqrt{n}})$. By our assumption that $M_N = O(\alpha_N)$, it implies (36) can be upper bounded by

$$\begin{aligned} M_N + \mathbb{E}\left[\sum_{n=1}^N \frac{w_n^2}{2}\|e_n\|_{*,n}^2\right] &\le O\left(\frac{w_{1:N}}{\sqrt{N}}\right) + \left[\sum_{n=1}^N \frac{w_n^2\sqrt{n}}{2w_{1:n}}\|e_n\|_*^2\right] \\ &\le O\left(\frac{w_{1:N}}{\sqrt{N}} + \sum_{n=1}^N \frac{w_n^2\sqrt{n}}{w_{1:n}}\right) = O\left(N^{p+1/2}\right) \end{aligned}$$

which will lead to an optimal weighted average regret in $O(\frac{1}{\sqrt{N}})$.

PICCOLO actually has a better regret than the simplified case discussed above, because of the negative term $-\frac{1}{2}\|\pi_n - \hat{\pi}_n\|_{n-1}^2$ in (35). To see its effects, we combine Lemma H.1 with (35) to reveal some insights:

$$\begin{aligned} &\mathbb{E}\left[\sum_{n=1}^N w_n \langle g_n, \pi_n - \pi\rangle\right] \\ &\le O(\alpha_N) + \mathbb{E}\left[\sum_{n=1}^N \frac{w_n^2}{2}\|e_n\|_{*,n}^2 - \frac{1}{2}\|\pi_n - \hat{\pi}_n\|_{n-1}^2\right] \qquad (37) \\ &\le O(\alpha_N) + \mathbb{E}\left[\sum_{n=1}^N \frac{2w_n^2}{\alpha_n}\left(\sigma_g^2 + \sigma_{\hat{g}}^2 + L^2\|\pi_n - \hat{\pi}_n\|_n^2 + E_n(\Phi_n)\right) - \frac{\alpha_{n-1}}{2}\|\pi_n - \hat{\pi}_n\|^2\right] \\ &= \left(O(\alpha_N) + \mathbb{E}\left[\sum_{n=1}^N \frac{2w_n^2}{\alpha_n}\left(\sigma_g^2 + \sigma_{\hat{g}}^2 + E_n(\Phi_n)\right)\right]\right) + \left(\mathbb{E}\left[\sum_{n=1}^N (\frac{2w_n^2}{\alpha_n}L^2 - \frac{\alpha_{n-1}}{2})\|\pi_n - \hat{\pi}_n\|^2\right]\right) \qquad (38) \end{aligned}$$

The first term in (38) plays the same role as (36); when the base algorithm has an optimal `adapt` operation and $w_n = n^p$ for some $p > -1$, it would be in $O\left(N^{p+1/2}\right)$. Here we see that the constant factor in this bound is proportional to $\sigma_g^2 + \sigma_{\hat{g}}^2 + E_n(\Phi_n)$. Therefore, if the variances σ_g^2, $\sigma_{\hat{g}}^2$ of the gradients are small, the regret would mainly depend on the prediction error $E_n(\Phi_n)$ of Φ_n. In the next section (Appendix H.4), we will show that when Φ_n is learned online (as the authors in [23] suggest), on average the regret is close to the regret of using the best model in the hindsight. The second term in (38) contributes to $O(1)$ in the regret, when the base algorithm adapts properly to w_n. For example, when $\alpha_n = \Theta(\frac{w_{1:n}}{\sqrt{n}})$ and $w_n = n^p$ for some $p > -1$, then

$$\sum_{n=1}^N \frac{2w_n^2}{\alpha_n}L^2 - \frac{\alpha_{n-1}}{2} = \sum_{n=1}^N O(n^{p-1/2} - n^{p+1/2}) = O(1)$$

In addition, because $\|\pi_n - \hat{\pi}_n\|$ would converge to zero, the effects of the second term in (38) becomes even minor.

In summary, for a reasonable base algorithm and $w_n = n^p$ with $p > -1$, running PICCOLO has the regret bound

$$\mathbb{E}\left[\sum_{n=1}^N w_n \langle g_n, \pi_n - \pi\rangle\right] = O(\alpha_N) + O\left(\frac{w_{1:N}}{\sqrt{N}}(\sigma_g^2 + \sigma_{\hat{g}}^2)\right) + O(1) + \mathbb{E}\left[\sum_{n=1}^N \frac{2w_n^2}{\alpha_n}E_n(\Phi_n)\right] \qquad (39)$$

Suppose $\alpha_n = \Theta(|\Pi|\frac{w_{1:n}}{\sqrt{n}})$ and $w_n = n^p$ for some $p > -1$, This implies the inequality

$$\mathbb{E}\left[\sum_{n=1}^N \langle w_n g_n, \pi_n - \pi\rangle\right] \le O(1) + C_{\Pi,\Phi}\frac{w_{1:N}}{\sqrt{N}} \qquad (34)$$

where $C_{\Pi,\Phi} = O(|\Pi| + \sigma_g^2 + \sigma_{\hat{g}}^2 + \sup_n E_n(\Phi_n))$. The use of non-uniform weights can lead to a faster on average decay of the standing $O(1)$ term in the final weighted average regret bound, i.e.

$$\frac{1}{w_{1:N}}\mathbb{E}\left[\sum_{n=1}^{N}\langle w_n g_n, \pi_n - \pi\rangle\right] \leq O\left(\frac{1}{w_{1:N}}\right) + \frac{C_{\Pi,\Phi}}{\sqrt{N}}$$

In general, the authors in [28, 23] recommend using $p \ll N$ (e.g. in the range of $[0, 5]$) to remove the undesirable constant factor, yet without introducing large multiplicative constant factor.

H.4 Model Learning

The regret bound in (39) reveals an important factor that is due to the prediction error $\mathbb{E}\left[\sum_{n=1}^{N}\frac{2w_n^2}{\alpha_n}E_n(\Phi_n)\right]$, where we recall $E_n(\Phi_n) = \|\mathbb{E}[g_n] - \mathbb{E}[\Phi_n(\pi_n, \psi_n)]\|_*^2$. Cheng et al. [23] show that, to minimize this error sum through model learning, a secondary online learning problem with per-round loss $E_n(\cdot)$ can be considered. Note that this is a standard weighted adversarial online learning problem (weighted by $\frac{2w_n^2}{\alpha_n}$), because $E_n(\cdot)$ is revealed after one commits to using model Φ_n.

While in implementation the exact function $E_n(\cdot)$ is unavailable (as it requires infinite data), we can adopt an unbiased upper bound. For example, Cheng et al. [23] show that $E_n(\cdot)$ can be upper bounded by the single- or multi-step prediction error of a transition dynamics model. More generally, we can learn a neural network to minimize the gradient prediction error directly. As long as this secondary online learning problem is solved by a no-regret algorithm, the error due to online model learning would contribute a term in $O(w_{1:N}\epsilon_{\mathcal{P},N}/\sqrt{N}) + o(w_{1:N}/\sqrt{N})$ in (39), where $\epsilon_{\mathcal{P},N}$ is the minimal error achieved by the best model in the model class $\mathcal{P}$ (see [23] for details).

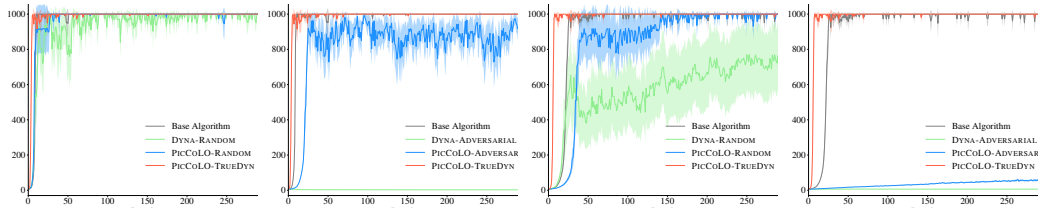
I Experimental Details

I.1 Algorithms

Base Algorithms In the experiments, we consider three commonly used first-order online learning algorithms: ADAM, NATGRAD, and TRPO, all of which adapt the regularization online to alleviate the burden of learning rate tuning. We provide the decomposition of ADAM into the basic three operations in Appendix D, and that of NATGRAD in Appendix F. In particular, the adaptivity of NATGRAD is achieved by adjusting the step size based on a moving average of the dual norm of the gradient. TRPO adjusts the step size to minimize a given cost function (here it is a linear function defined by the first-order oracle) within a pre-specified KL divergence centered at the current decision. While greedily changing the step size in every iteration makes TRPO an inappropriate candidate for adversarial online learning. Nonetheless, it can still be written in the form of mirror descent and allows a decomposition using the three basic operators; its `adapt` operator can be defined as the process of finding the maximal scalar step along the natural gradient direction such that the updated decision stays within the trust region. For all the algorithms, a decaying step size multiplier in the form $\eta/(1 + \alpha\sqrt{n})$ is also used; for TRPO, it is used to specify the size of trust regions. The values chosen for the hyperparameters η and α can be found in Table 2. To the best of our knowledge, the conversion of these approaches into unbiased model-based algorithms is novel.

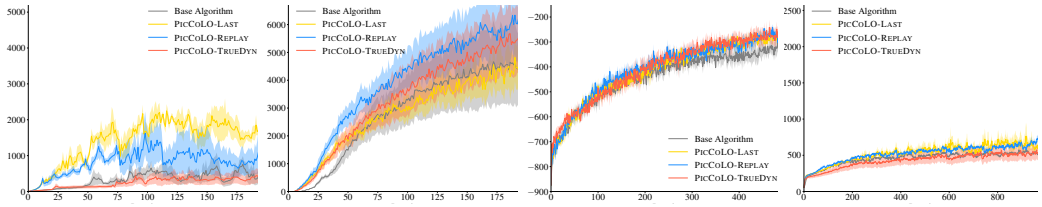
Reinforcement Learning Per-round Loss In iteration n, in order to compute the online gradient (7), GAE [42] is used to estimate the advantage function $A_{\pi_{n-1}}$. More concretely, this advantage estimate utilizes an estimate of value function $V_{\pi_{n-1}}$ (which we denote $\hat{V}_{\pi_{n-1}}$) and on-policy samples. We chosen $\lambda = 0.98$ in GAE to reduce influence of the error in $V_{\pi_{n-1}}$, which can be catastrophic. Importance sampling can be used to estimate $A_{\pi_{n-1}}$ in order to leverage data that are collected on-policy by running π_n. However, since we select a large λ, importance sampling can lead to vanishing importance weights, making the gradient extremely noisy. Therefore, in the experiments, importance sampling is not applied.

Gradient Computation and Control Variate The gradients are computed using likelihood-ratio trick and the associated advantage function estimates described above. A scalar control variate is



(a) Random dynamics (b) Adversarial model (c) Random dynamics (d) Adversarial model

Figure 2: Performance of PICCOLO and DYNA with predictive models of different accuracies. NATGRAD (left two plots) and TRPO (right two plots) are used as base algorithms. The shaded regions are 0.5 standard deviation.



(a) Hopper (b) Snake (c) Reacher3D (d) Walker3D

Figure 3: The performance of PICCOLO in various tasks. ADAM is used as the base algorithms. The shaded regions account for 0.5 standard deviation.

further used to reduce the variance of the sampled gradient, which is set to the mean of the advantage estimates evaluated on newly collected data.

Policies and Value Networks Simple feed-forward neural networks are used to construct all of the function approximators (policy and value function) in the tasks. They have 1 hidden layer with 32 $\tanh$ units for all policy networks, and have 2 hidden layers with 64 $\tanh$ units for value function networks. Gaussian stochastic policies are considered, i.e., for any state $s \in \mathbb{S}$, π_s is Gaussian, and the mean of π_s is modeled by the policy network, whereas the diagonal covariance matrix is state independent (which is also learned). After the policy update, a new value function estimate $\hat{V}_{\pi_n}$ is computed by minimizing the mean of squared difference between $\hat{V}_{\pi_n}$ and $\hat{V}_{\pi_{n-1}} + \hat{A}_{\pi_n}$, where $\hat{A}_{\pi_n}$ is the GAE estimate using $\hat{V}_{\pi_{n-1}}$ and $\lambda = 0.98$, through ADAM with batch size 128, number of batches 2048, and learning rate 0.001.

I.2 Tasks

The robotic control tasks that are considered in the experiments are CartPole, Hopper, Snake, Reacher3D, and Walker3D from OpenAI Gym [56] with the DART physics engine [57][12]. CartPole is a classic control problem, and its goal is to keep a pole balanced in a upright posture, by only applying force to the cart. Hopper, Snake, and Walker3D are locomotion tasks, of which the goal is to control an agent to move forward as quickly as possible without falling down (for Hopper and Walker3D) or deviating too much from moving forward (for Snake). Hopper is monopedal and Walker3D is bipedal, and both of them are subjected to significant contact discontinuities that are hard or even impossible to predict. The Reacher3D task is about controlling a 5 degrees-of-freedom manipulator to reach a random target position in a 3D space.

I.3 Complete Experimental Results

Here we provide additional experimental results where the performance is measured by the accumulated rewards. We note that, when reading these figures, we should only compare the PICCOLOed algorithm with the base algorithm. The purpose of these experiments is to validate whether PICCOLO, as a meta-algorithm, can improve the original approach.

[12]The environments are defined in DartEnv, hosted at https://github.com/DartEnv.

In general, we see PICCOLO improves the base algorithm, but the exact amount depends on the specific setting. This trend is similar to trend of the experiments presented in the main paper. However, we note the followings. First, we find that ADAM does not work well in Hopper and Walker3D, even the base algorithm. While PICCOLO does improve its performance in these two tasks, ADAM is still inferior to NATGRAD and TRPO. This might be due to that ADAM only has a diagonal regularization, which may be insufficient to capture the nonlinearity in these tasks. Second, we find that PICCOLO with TRPO has not yet converged under the adversarial setting. This might be because 1) the convergence is slow and would take more iterations 2) TRPO as a *base algorithm* is not designed for the adversarial setting (as mentioned above). We suspect the latter may be the cause, as TRPO changes the regularization size greedily for every iteration, which could incur a linear regret in adversarial online learning. A further investigation into this issue and robustifying TRPO are left as future work.

	CartPole	Hopper	Snake	Reacher3D	Walker3D
Observation space dimension	4	11	17	21	41
Action space dimension	1	3	6	5	15
State space dimension	4	12	18	10	42
Number of samples from env. per iteration	4k	16k	16k	16k	16k
Number of samples from model dyn. per iteration	4k	48k	16k	48k	48k
Length of horizon	1,000	1,000	1,000	500	1,000
Number of iterations	100	200	200	500	1,000
Number of seeds	12	4	6	4	4
α [13]	0.1	0.01	0.01	0.01	0.01
η in ADAM	0.01	0.03	0.005	0.01	0.01
η in NATGRAD	0.1	0.1	0.06	0.1	0.2
η in TRPO	0.003	0.004	0.002	0.002	0.008
β_1 in ADAM	0	0.9	0.9	0.9	0.9

Table 2: Tasks specifics and hyperparameters. $\beta_2 = 0.999$ is set for NATGRAD and ADAM.

[13] α, and η appear in the decaying step size multiplier for all the algorithms in the form $\eta/(1 + \alpha\sqrt{n})$.