Abstract

We present a predictor-corrector framework, called P\textsuperscript{IC}C\textsuperscript{O}LO, 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 “P\textsuperscript{IC}C\textsuperscript{O}LOed” 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, P\textsuperscript{IC}C\textsuperscript{O}LO corrects for the mistakes of using imperfect predicted gradients and hence does not suffer from model bias. The development of P\textsuperscript{IC}C\textsuperscript{O}LO 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 P\textsuperscript{IC}C\textsuperscript{O}LO.

1. Introduction

Reinforcement learning (RL) has recently solved a number of challenging problems (Mnih et al., 2013; Duan et al., 2016; Silver et al., 2018). 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-world 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 leveraging an accurate model that can cheaply simulate interactions to compute policy updates in lieu of real-world interactions (Tan et al., 2018). A classical example of pure model-based methods is optimal control (Jacobson & Mayne, 1970; Todorov & Li, 2005; Deisenroth & Rasmussen, 2011; Pan & Theodorou, 2014), which has recently been extended to model abstract latent dynamics with neural networks (Silver et al., 2017; Oh et al., 2017). These methods use a (local) model of the dynamics and cost functions to predict cost-to-go functions, policy gradients, or promising improvement direction when updating policies (Levine & Koltun, 2013; Sun et al., 2018; Anthony et al., 2017). Another way to use model information is the hybrid DYNA framework (Sutton, 1991; Sutton et al., 2012), which interleaves model-based and model-free updates, ideally cutting learning time in half. However, all of these approaches, while potentially accelerating policy 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 (Pascanu et al., 2017; Srinivas et al., 2018; Amos et al., 2018), so the performance of a given planning structure is directly optimized. However, these algorithms are still optimized through standard model-free RL techniques; it is unclear as to whether they are more sample efficient. In parallel, another class of bias-free algorithms is control variate methods (Chebotar et al., 2017; Grathwohl et al., 2018; Papini et al., 2018), which use models to reduce the variance of sampled gradients to improve convergence.

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 (Gordon, 1999; Zinkevich, 2003). The use of online learning to analyze policy optimization was pioneered by Ross et al. (2011), 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 (Ross et al., 2011) and AGREVE\textsuperscript{ATE} (Ross & Bagnell, 2014). However, it was recently shown that the naive reduction to adversarial online learning...
losses information (Cheng & Boots, 2018): in practice, IL is **predictable** (Cheng et al., 2019) and can be thought of as a predictable online learning problem (Rakhlin & Sridharan, 2013a). Based on this insight, Cheng et al. (2019) recently proposed a two-step algorithm, MOBIL. The authors prove that, by leveraging predictive models to estimate future gradients, 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. In this paper, we show that RL can also be formulated as a predictable online learning problem, and we propose a novel first-order learning framework, PtcCoLO (PredICtor-COrrector PoLicy Optimization), for general predictable online learning problems. PtcCoLO is a meta-algorithm: it takes a standard online learning algorithm designed for adversarial problems (e.g. ADAGRAD (Duchi et al., 2011)) as input and returns a new hybrid algorithm that can use model information to accelerate convergence. This new “PtcCoLOed” algorithm optimizes the policy by alternating between Prediction and Correction steps. In the Prediction Step, the learner uses a predictive model to estimate the gradient of the next loss function and then uses it to update the policy; in the Correction Step, the learner executes the updated policy in the environment, receives the true gradient, and then corrects the policy using the gradient error. We note that PtcCoLO is orthogonal to control variate methods; it can still improve learning even in the noise-free setting (see Section 5.2).

Theoretically, we prove that PtcCoLO can improve the convergence rate of any base algorithm that can be written as mirror descent (Beck & Teboulle, 2003) or Follow-the-Regularizer-Leader (FTRL) (McMahan & Streeter, 2010). This family of algorithms is rich and covers most first-order algorithms used in RL and IL (Cheng et al., 2018). And, importantly, we show that PtcCoLO does not suffer from performance bias due to model error, unlike previous model-based approaches. To validate the theory, we “PtcCoLO” multiple algorithms in simulation. The experimental results show that the PtcCoLOed versions consistently surpass the base algorithm and are robust to model errors.

The design of PtcCoLO 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. We show that PtcCoLO includes and generalizes many existing algorithms, e.g., MoBIL, mirror-prox (Juditsky et al., 2011), and optimistic mirror descent (Rakhlin & Sridharan, 2013a) (Appendix A). Thus, we can treat PtcCoLO 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.

2. Problem Definition

We consider solving policy optimization problems: given state and action spaces \( S \) and \( A \), and a parametric policy class \( \Pi \), we desire a stationary policy \( \pi \in \Pi \) that solves

\[
\min_{\pi \in \Pi} J(\pi), \quad J(\pi) := \mathbb{E}_{(s,t) \sim d_\pi} \mathbb{E}_{a \sim \pi_s} [c_t(s, a)]
\]

where \( c_t(s, a) \) is the instantaneous cost at time \( t \) of state \( s \in S \) and \( a \in A \), \( \pi_s \) is the distribution of \( a \) at state \( s \) under policy \( \pi \), and \( d_\pi \) is a generalized stationary distribution of states generated by running policy \( \pi \) in a Markov decision process (MDP); the notation \( \mathbb{E}_{a \sim \pi_s} \) denotes evaluation when \( \pi \) is deterministic. The use of \( d_\pi \) in (1) abstracts different discrete-time RL/IL problems into a common setup. For example, an infinite-horizon \( \gamma \)-discounted problem with time-invariant cost \( c \) can be modeled by setting \( c_t = c \) and \( d_\pi(s, t) = (1 - \gamma)^t d_\pi(s) \), where \( d_\pi(s) \) is the state distribution visited by policy \( \pi \) at time \( t \) starting from some fixed but unknown initial state distribution.

For convenience, we will usually omit the random variable in expectation notation (e.g. we will write (1) as \( \mathbb{E}_{d_\pi, \mathbb{E}_\pi [c]} \)). For a policy \( \pi \), we overload the notation \( \mathbb{E}_\pi \) to also denote its parameter, and write \( Q_\pi, t \) and \( V_\pi, t := \mathbb{E}_\pi [Q_\pi, t] \) as its Q-function and value function at time \( t \), respectively.

3. IL and RL as Predictable Online Learning

We study policy optimization through the lens of online learning (Hazan et al., 2016), by treating a policy optimization algorithm as the learner in online learning and each intermediate policy that it produces as an online decision. This identification recasts the iterative process of policy optimization into a standard online learning setup: in round \( n \), the learner plays a decision \( \pi_n \), a per-round loss \( l_n \) is then selected, and finally some information of \( l_n \) is revealed to the learner for making the next decision. We note that the “rounds” considered here are the number of episodes that a single iteration of IL/RL problem. We then define regret as the sum of all losses and expressiveness of the policy class.

\[
\text{Regret}_N(l) := \sum_{n=1}^N w_n l_n(\pi_n) - \min_{\pi \in \Pi} \sum_{n=1}^N w_n l_n(\pi)
\]

and an expressiveness measure of the policy class \( \Pi \)

\[
\epsilon_{\Pi,N}(l) := \frac{1}{w_{1:N}} \min_{\pi \in \Pi} \sum_{n=1}^N w_n l_n(\pi)
\]

where \( w_n > 0 \) and \( w_{1:n} := \sum_{m=1}^n w_m \). Moreover, we show that these online learning problems are predictable:
that is, the per-round losses are not completely adversarial but can be estimated from past information. We will use these ideas to design PtCColo in the next section.

3.1. IL as Online Learning

We start by reviewing the classical online learning approach to IL (online IL for short) (Ross et al., 2011) to highlight some key ideas. IL leverages domain knowledge about a policy optimization problem through expert demonstrations. Online IL, in particular, optimizes policies by letting the learner π query the expert π∗ for desired actions, so that a policy can be quickly trained to perform as well as the expert. At its heart, online IL is based on the following lemma, which relates the performance between π and π∗.

Lemma 1. (Kakade & Langford, 2002) Let π and π∗ be two policies and Aπ∗,t(s, a) := Qπ∗,t(s, a) − Vπ∗,t(s). Then J(π) = J(π∗) + EdπEπ[Aπ∗].

Given the equality in Lemma 1, the performance difference between π and π∗ can then be upper-bounded as


for some positive constant Cπ∗, and function D1, which is often derived from statistical distances such as KL divergence (Cheng et al., 2018). When Aπ∗,t is available, we can also set D1(π∗||πs) = Eπ−π[Aπ∗,t(s, a)], as in value aggregation (AGREVEATE) (Ross & Bagnell, 2014).

Without loss of generality, let us suppose D1(π∗||πs) = Eπ−π[π(s, a)] for some ππ. Online IL converts policy optimization into online learning with per-round loss

ln(π) := EdπEπ[π].

By the inequality above, it holds that J(πn) − J(π∗) ≤ Cπ∗ln(πn) for every n, establishing the reduction below.

Lemma 2. (Cheng et al., 2019) For ln defined in (4),

E[ Σn=1 wnlN(l) ] ≤ J(π∗) + Cπ∗E[ ϵ11N(l) + Regretω(n) ]

where the expectation is due to sampling ln.

That is, when a no-regret algorithm is used, the performance concentrates toward J(π∗) + Cπ∗E[ ϵ11N(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 1. Given a policy πn in round n, we define a per-round loss

ln(π) := EdπEπ[Aπn−1].

which describes how well a policy π performs relative to the previous policy πn−1 under the state distribution of πn. By Lemma 1, for ln defined in (5), ln(πn) = J(πn) − J(πn−1) for every n, similar to the pointwise inequality of ln that Lemma 2 is based on. With this observation, we derive the reduction below (proved in Appendix B).

Lemma 3. Suppose w n+k /w n ≤ w m+k /w m for all n ≥ m ≥ 1 and k ≥ 0. For (5) and any πn, E[ Σn=1 wnlN(l) ] ≤ J(πn) + Σn=1 wnlN(l) E[Regretω(n) ] + w1nn1n(l), where the expectation is due to sampling ln.

3.2.1. Interpretations

Lemma 3 is a policy improvement lemma, which shows that when the learning algorithm is no-regret, the policy sequence improves on-average from the initial reference policy π0 that defines l1. This is attributed to an important property of the definition in (5) that minπ∈Πln(π) ≤ 0. To see this, suppose E[ ϵ11N(π) ] ≤ −Ω(1) (i.e. there is a policy that is better than all previous n policies); this is true for small n or when the policy sequence is concentrated. Under this assumption, if w n = 1 and Regretω(n) ≤ O(√n), then the average performance improves roughly N E[ ϵ11N(l) ] away from J(π0).

While it is unrealistic to expect E[ ϵ11N(l) ] ≤ 0 for large n, we can still use Lemma 3 to comprehend global properties of policy improvement, for two reasons. First, the inequality in Lemma 3 holds for any interval of the policy sequence. Second, as we show in Appendix B, the Lemma 3 also applies to dynamic regret (Zinkevich, 2003), with respect to which E[ ϵ11N(l) ] is always negative. Therefore, if an algorithm is strongly-adaptive (Daniely et al., 2015) (i.e. it is no-regret for any interval) or has sublinear dynamic regret (Jadbabaie et al., 2015), then its generated policy sequence will strictly, non-asymptotically improve. In other words, for algorithms with a stronger notion of convergence, Lemma 3 describes the global improvement rate.

3.2.2. Connections

The choice of per-round loss in (5) has an interesting relationship to both actor-critic in RL (Konda & Tsitsiklis, 2000) and AGREVEATE in IL (Ross & Bagnell, 2014).

Relationship to Actor-Critic

Although actor-critic methods, theoretically, use Edπn(∂πEπ[Aπn]|π=πn) to update policy πn, in practice, they use Edπn(∂πEπ[Aπn−1]|π=πn), because the advantage/value function estimate in round n is updated after the policy update in order to prevent bias due to over-fitting on finite samples (Sutton & Barto, 1998). This practical gradient is exactly ∇ln(πn), the sampled gradient of (5). Therefore, Lemma 3 explains the properties of these practical modifications.

Relationship to Value Aggregation

AGREVEATE (Ross & Bagnell, 2014) can be viewed as taking a policy improvement step from some reference policy: e.g., with the
per-round loss $E_{d_{π_n}} E_π [A_{π^n}]$, it improves one step from $π^*$. Realizing this one step improvement in AGGREvATE, however, requires solving multiple rounds of online learning, as it effectively solves an equilibrium point problem (Cheng & Boots, 2018). Therefore, while ideally one can solve multiple AGGREvATE problems (one for each policy improvement step) to optimize policies, computationally this can be very challenging. Minimizing the loss in (5) 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 switches to the next AGGREvATE problem with a new reference policy (i.e. the latest policy $π_{n-1}$). This connection is particularly tightened if we choose $π_0 = π^*$ and the bound in Lemma 3 becomes relative to $J(π^*)$.

3.3. Predictability

An important property of the above online learning problems is that they are not completely adversarial, as pointed out by Cheng & Boots (2018) for IL. This can be seen from the definitions of $l_n$ in (4) and (5), 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. That is, in an idealized case where the true dynamics and cost functions are exactly known, using the policy returned from a model-based RL algorithm would incur zero regret, since only the interactions with the real MDP environment, not the model, counts as rounds. We will exploit this property to design PicCoLO.

4. Predictor-Corrector Learning

We showed that the performance of RL and IL can be bounded by the regret of properly constructed predictable online learning problems. These results provide a foundation for designing policy optimization algorithms: efficient learning algorithms for policy optimization can be constructed from powerful online learning algorithms that achieve small regret. This perspective explains why common methods (e.g. mirror descent) based on gradients of (4) and (5) work well in IL and RL. However, the predictable nature of policy optimization problems suggests that directly applying these standard online learning algorithms designed for adversarial settings is suboptimal. The predictable information must be considered to achieve optimal convergence. One way to include predictable information is to develop specialized two-step algorithms based on, e.g., mirror-prox or FTRL-prediction (Juditsky et al., 2011; Rakhlin & Sridharan, 2013a; Ho-Nguyen & Kilinc-Karzan, 2018). For IL, MoBiL was recently proposed (Cheng et al., 2019), which updates policies by approximate Be-the-Leader (Kalai & Vempala, 2005) and provably achieves faster convergence than previous methods. However, these two-step algorithms often have obscure and non-sequential update rules, and their adaptive and accelerated versions are less accessible (Diakonikolas & Orecchia, 2017). This can make it difficult to implement and 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 first-order policy optimization algorithms belong to this family (Cheng et al., 2018), so we can PicCoLO 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. The PicCoLO Idea

The design of PicCoLO is based on the observation that an $N$-round predictable online learning problem can be written as a new adversarial problems with $2N$ rounds. To see this, let $\{l_n\}_{n=1}^N$ be the original predictable loss sequence. Suppose, before observing $l_n$, we have access to a model loss $\hat{l}_n$ that contains the predictable information of $l_n$. Define $δ_n = l_n - \hat{l}_n$. We can then write the accumulated loss (which regret concerns) as $\sum_{n=1}^N l_n(π_n) = \sum_{n=1}^N l_n(π_n) + δ_n(π_n)$. That is, we can view the predictable problem with $\{l_n\}_{n=1}^N$ as a new adversarial online learning problem with a loss sequence $\hat{l}_1, \hat{l}_2, \hat{l}_2, \ldots, l_N, δ_N$.

The idea of PicCoLO is to apply standard online learning algorithms designed for adversarial settings to this new $2N$-round problem. This would create a new set of decision variables $\{\tilde{π}_{n}\}_{n=1}^N$, in which $\tilde{π}_n$ denotes the decision made before seeing $l_n$, and leads to the following sequence $π_1, δ_1, \tilde{π}_2, \hat{l}_2, \tilde{π}_2, δ_2, \ldots, \tilde{π}_N, δ_N$ (in which we define $δ_1 = l_1$). We show that when the base algorithm is optimal in adversarial settings, this simple strategy results in a decision sequence $\{π_n\}_{n=1}^N$ whose regret with respect to $\{l_n\}_{n=1}^N$ is optimal, just as those specialized two-step algorithms (Juditsky et al., 2011; Rakhlin & Sridharan, 2013a; Ho-Nguyen & Kilinc-Karzan, 2018). In Appendix A, we show PicCoLO unifies and generalize these two-step algorithms to be adaptive.
4.2. The Meta Algorithm PtcCoLO

We provide details to realize this reduction. We suppose, in round \( n \), the model loss is given as \( l_n(\pi) = \langle \hat{g}_n, \pi \rangle \) for some vector \( \hat{g}_n \), and stochastic first-order feedback \( g_n = \nabla \hat{l}_n(\pi_n) \) from \( l_n \) is received. Though this linear form of model loss seems restrictive, later in Section 4.2.3 we will show that it is sufficient to represent predictable information.

4.2.1. BASE ALGORITHMS

We first give a single description of different base algorithms for the formal definition of the reduction steps. Here we limit our discussions to mirror descent and postpone the FTRL case to Appendix C. We assume that \( \Pi \) is a convex compact 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 decisions based on proximal maps. In round \( n \), given direction \( g_n \) and weight \( w_n \), it executes

\[
\pi_{n+1} = \arg \min_{\pi \in \Pi} \{ w_n g_n, \pi \} + B_R(\pi||\pi_n) \tag{6}
\]

where \( R_n \) is a strongly convex function; (6) reduces to gradient descent with step size \( \eta_n \) when \( R_n(\cdot) = \frac{1}{2\eta_n} \| \cdot \|^2 \). More precisely, (6) is composed of two steps: 1) the update of the distance generator to \( R_n \), and 2) the update of the decision to \( \pi_{n+1} \); different mirror descent algorithms differ in how the regularization is selected and adapted.

PtcCoLO explicitly treats a base algorithm as the composition of two basic operations (this applies also to FTRL)

\[
H_n = \text{adapt}(h_n, H_{n-1}, g_n, w_n) \tag{7}
\]

\[
h_{n+1} = \text{update}(h_n, H_n, g_n, w_n)
\]

so that later it can recompose them to generate the new algorithm. For generality, we use \( h \) and \( H \) to denote the abstract representations of the decision variable and the regularization, respectively. In mirror descent, \( h \) is exactly the decision variable, \( H \) is the distance generator, and we can write

\[
\text{update}(h, H, g, w) = \arg \min_{\pi' \in \Pi} \{ w g, \pi' \} + B_H(\pi'||h). \tag{8}
\]

The operation \( \text{adapt} \) denotes the algorithm-specific scheme for the regularization update (e.g. changing the step size), which in general updates the size of regularization to grow slowly and inversely proportional to the norm of \( g_n \).

4.2.2. THE PticLooED ALGORITHM

PtcCoLO generates decisions by applying a given base algorithm in (7) to the new problem with losses \( \delta_1, \delta_2, \ldots \). This is accomplished by recomposing the basic operations in (7) into the Prediction and the Correction Steps:

\[
h_n = \text{update}(h_n, H_{n-1}, \hat{g}_n, w_n) \quad \text{[Prediction]}
\]

\[
H_n = \text{adapt}(h_n, H_{n-1}, e_n, w_n)
\]

\[
h_{n+1} = \text{update}(h_n, H_n, e_n, w_n) \quad \text{[Correction]}
\]

where \( \hat{h}_n \) is the abstract representation of \( \hat{\pi}_n \), and \( e_n = g_n - \hat{g}_n \) is the error direction. We can see that the Prediction and Correction Steps are exactly the update rules resulting from applying (7) to the new adversarial problem, except that only \( \hat{h}_n \) is updated in the Prediction Step, not the regularization (i.e. the step size). This asymmetry design is important for achieving optimal regret, because in the end we care only about the regret of \{\pi_n\} on the original loss sequence \{l_n\}.

In round \( n \), the “PtcCoLOed” algorithm first performs the Prediction Step using \( \hat{g}_n \) to generate the learner’s decision (i.e. \( \pi_n \)) 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 done by first adapting the regularization to \( H_n \) and then updating \( \pi_n \) to \( \pi_{n+1} \) along the error \( e_n = g_n - \hat{g}_n \).

4.2.3. MODEL LOSSES AND PREDICTIVE MODELS

The Prediction Step of PicCoLO relies on the vector \( \hat{g}_n \) to approximate the future gradient \( g_n \). Here we discuss different ways to specify \( \hat{g}_n \) based on the concept of predictive models (Cheng et al., 2019). A predictive model \( \Phi_n \) is a first-order oracle such that \( \Phi_n(\cdot) \) approximates \( \nabla l_n(\cdot) \). In practice, a predictive model can be a simulator with an (online learned) dynamics model (Tan et al., 2018; Deisenroth & Rasmussen, 2011), or a neural network trained to predict the required gradients (Silver et al., 2017; Oh et al., 2017).

An even simpler heuristic is to construct predictive models by off-policy gradients \( \Phi_n(\cdot) = \sum_{m=n-K}^{n-1} \nabla l_m(\cdot) \) where \( K \) is the buffer size.

In general, we wish to set \( \hat{g}_n \) to be close to \( g_n \), as we will later show in Section 5 that the convergence rate of PtcCoLO depends on their distance. However, even when we have perfect predictive models, this is still a non-trivial task. We face a chicken-or-the-egg problem: \( g_n \) depends on \( \pi_n \), which in turn depends on \( \hat{g}_n \) from the Prediction Step.

Cheng et al. (2019) show one effective heuristic is to set \( \hat{g}_n = \Phi_n(\pi_n) \), because we may treat \( \pi_n \) as an estimate of \( \pi_n \). However, due to the mismatch between \( \hat{\pi}_n \) and \( \pi_n \) this simple approach has errors even when the predictive model is perfect. To better leverage a given predictive model, we propose to solve for \( \hat{g}_n \) and \( \pi_n \) simultaneously. That is, we wish to solve a fixed-point problem, finding \( \hat{h}_n \), such that

\[
h_n = \text{update}(h_n, H_{n-1}, \Phi_n(\pi_n(h_n)), w_n) \tag{8}
\]

The exact formulation of the fixed-point problem depends on the class of base algorithms. For mirror descent, it is a variational inequality: find \( \pi_n \in \Pi \) such that \( \forall \pi \in \Pi, \langle \Phi_n(\pi), \pi - \pi_n \rangle \geq 0 \). In a special case when \( \Phi_n = \nabla f_n \) for some function \( f_n \), the above variational inequality is equivalent to finding a stationary point of the optimization problem \( \min_{\pi \in \Pi} f_n(\pi) + B_{R_{n-1}}(\pi||\pi_n) \). In other words, one way to implement the
We provide a summary of the full algorithm for policy optimization in Algorithm 1. We see that PicCOLO uses the predicted gradient to take an extra step to accelerate learning, and, meanwhile, to prevent the error accumulation, based on the prediction error and corrects for the bias on the policy right away. To gain some intuition, let us consider ADAGRAD (Duchi et al., 2011) as a base algorithm:\(^1\)

\[
G_n = G_{n-1} + \text{diag}(w_n g_n) \circ w_n g_n \\
\pi_{n+1} = \arg\min_{\pi \in \Pi} \left( w_n g_n, \pi \right) + \frac{1}{2\eta} (\pi - \pi_n)^\top G_n^{1/2} (\pi - \pi_n)
\]

where \(G_0 = \epsilon I\) and \(\eta, \epsilon > 0\), and \(\circ\) denotes element-wise multiplication. This update has an adapt operation as adapt\((h, H, g, w) = G + \text{diag}(wg \circ wg)\) which updates the Bregman divergence based on the gradient size.

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

\[
\pi_n = \arg\min_{\pi \in \Pi} \left( w_n g_n, \pi \right) + \frac{1}{2\eta} (\pi - \pi_n-1)^\top G_{n-1}^{1/2} (\pi - \pi_n-1)
\]

In the Correction Step, it performs

\[
G_n = G_{n-1} + \text{diag}(w_n e_n \circ w_n e_n) \\
\hat{\pi}_{n+1} = \arg\min_{\pi \in \Pi} \left( w_n e_n, \pi \right) + \frac{1}{2\eta} (\pi - \hat{\pi}_n)^\top G_n^{1/2} (\pi - \hat{\pi}_n)
\]

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 theory and in the experiments.

### 4.3. Summary: Why Does PicCOLO Work?

We provide a summary of the full algorithm for policy optimization in Algorithm 1. We see that PicCOLO uses the predicted gradient to take an extra step to accelerate learning, and, 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 (Duchi et al., 2011) as a base algorithm:\(^1\):

\[
G_n = G_{n-1} + \text{diag}(w_n g_n) \circ w_n g_n \\
\pi_{n+1} = \arg\min_{\pi \in \Pi} \left( w_n g_n, \pi \right) + \frac{1}{2\eta} (\pi - \pi_n)^\top G_n^{1/2} (\pi - \pi_n)
\]

where \(G_0 = \epsilon I\) and \(\eta, \epsilon > 0\), and \(\circ\) denotes element-wise multiplication. This update has an adapt operation as adapt\((h, H, g, w) = G + \text{diag}(wg \circ wg)\) which updates the Bregman divergence based on the gradient size.

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

\[
\pi_n = \arg\min_{\pi \in \Pi} \left( w_n g_n, \pi \right) + \frac{1}{2\eta} (\pi - \pi_n-1)^\top G_{n-1}^{1/2} (\pi - \pi_n-1)
\]

In the Correction Step, it performs

\[
G_n = G_{n-1} + \text{diag}(w_n e_n \circ w_n e_n) \\
\hat{\pi}_{n+1} = \arg\min_{\pi \in \Pi} \left( w_n e_n, \pi \right) + \frac{1}{2\eta} (\pi - \hat{\pi}_n)^\top G_n^{1/2} (\pi - \hat{\pi}_n)
\]

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 theory and in the experiments.

\(^1\)We provide another example in Appendix E.

\(^2\)Here we assume project is automatically performed inside PredictionStep and CorrectionStep.

### 5. Theoretical Analysis

In this section, 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 operation of a base algorithm.

**Assumption 1.** adapt chooses a regularization sequence such that, for some \(M_N = o(w_{1:N})\), \(\|H_0\|_R + \sum_{n=1}^N \|H_n - H_{n-1}\|_R \leq M_N\) for some norm \(\cdot\) which measures the size of regularization.

This assumption, which requires the regularization to increase 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 \(O\left(\frac{1}{n}\right)\) has \(M_N = O(\sqrt{N})\). In general, for stochastic problems, an optimal base algorithm would ensure \(M_N = O\left(\frac{w_{1:N}}{\sqrt{N}}\right)\).

#### 5.1. 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 F.

**Theorem 1.** Suppose \(H_n\) defines a strongly convex function with respect to \(\|\cdot\|_R\). Under Assumption 1, running PicCOLO ensures \(\sum_{n=1}^N \langle w_n g_n, \pi_n - \pi \rangle \leq M_N + \sum_{n=1}^N \frac{\sigma_n^2}{2} \|e_n\|_2^2, n - \frac{1}{2} \|\pi_n - \hat{\pi}_n\|_2^2, \forall \pi \in \Pi\), for all \(\pi \in \Pi\).

The term \(\|\pi_n - \hat{\pi}_n\|_2^2\) in Theorem 1 says that the performance of PicCOLO depends on how well the base algorithm adapts to the error \(e_n\) through the adapt operation in the Correction Step. Usually adapt updates \(H_n\) gradually (Assumption 1) while minimizing \(\frac{1}{2} \|e_n\|_2^2\), like we showed in ADAGRAD.

In general, when the base algorithm is adaptive and optimal for adversarial problems, we show in Appendix G that its PicCOLOed version guarantees that \(E[\sum_{n=1}^N \langle w_n g_n, \pi_n - \pi \rangle] \leq O(1) + C_{II, \Phi} \frac{\sigma_n^2}{\sqrt{N}}\), where \(C_{II, \Phi} = O(\|\Pi\| + E_{\Phi} + \sigma_2^2 + \sigma_3^2)\) is some constant related to the diameter of \(\Pi\) (denoted as \(\|\Pi\|\)), the model bias \(E_{\Phi}\), and the sampling variance \(\sigma_2^2\) and \(\sigma_3^2\) of \(g_n\) and \(\hat{g}_n\), respectively. Through Lemma 2 and 3, this bound directly implies accelerated and bias-free policy performance.

**Theorem 2.** Suppose \(\hat{l}_n\) is convex and \(w_n \geq \Omega(1)\). Then running PicCOLO yields \(E[\text{Regret}_n(\hat{l})/w_{1:N}] = O\left(C_{II, \Phi} \frac{\sqrt{N}}{\sqrt{N}}\right)\), where \(C_{II, \Phi} = O(\|\Pi\| + E_{\Phi} + \sigma_2^2 + \sigma_3^2) = O(1)\).

\(^3\)The convexity assumption is standard, as used in (Duchi et al., 2011; Ross et al., 2011; Kingma & Ba, 2015; Cheng & Boots,
Table 1: Upper bounds of the average regret of different policy optimization algorithms.

<table>
<thead>
<tr>
<th>ALGORITHMS</th>
<th>UPPER BOUNDS IN BIG-O</th>
</tr>
</thead>
<tbody>
<tr>
<td>PicCoLO</td>
<td>$\frac{1}{</td>
</tr>
<tr>
<td>MODEL-FREE</td>
<td>$\frac{1}{2} \left(</td>
</tr>
<tr>
<td>MODEL-BASED</td>
<td>$\frac{1}{2} \left(</td>
</tr>
<tr>
<td>DYNA</td>
<td>$\frac{1}{2} \left(</td>
</tr>
</tbody>
</table>

5.2. Comparison

To appreciate the advantages of PicCoLO, we review several policy optimization algorithms and compare their regret. We show that they can be viewed as incomplete versions of PicCoLO, which only either result in accelerated learning or are unbiased, but not both (see in Table 1).

We first consider the common model-free approach (Sutton et al., 2000; Kakade, 2002; Peters & Schaal, 2008; Peters et al., 2010; Silver et al., 2014; Sun et al., 2017; Cheng et al., 2018), i.e. applying the base algorithm with $g_n$. To make the comparison concrete, suppose $\|E[\hat{g}_n]\|^2_s \leq G_g^2$ for some constant $G_g$, where we recall $g_n$ is the sampled true gradient. As the model-free approach is equivalent to setting $g_n = 0$ in PicCoLO, by Theorem 1 (with $e_n = g_n$), the constant $C_H$ in Theorem 2 would become $O(|\mathcal{A}| + G_g^2 + \sigma_g^2)$. In other words, PicCoLOing the base algorithm improves the constant factor from $G_g^2$ to $\sigma_g^2 + E_g$. 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.\(^4\)

Next we consider the pure model-based approach with a model that is potentially learned online (Jacobson & Mayne, 1970; Todorov & Li, 2005; Deisenroth & Rasmussen, 2011; Pan & Theodorou, 2014; Levine & Koltun, 2013; Sun et al., 2018). As this approach is equivalent to only performing the Prediction Step\(^5\), its performance suffers from any modeling error. Specifically, suppose $\|E[\hat{g}_n]\|^2_s \leq G_g^2$ for some constant $G_g$. One can show that the bound in Theorem 2 would become $O((|\mathcal{A}| + G_g^2 + \sigma_g^2)/\sqrt{N} + E_g)$, introducing a constant bias in $O(E_g)$.

A hybrid heuristic to combine the model-based and model-free updates is DYNA (Sutton, 1991; Sutton et al., 2012), 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 2 would become $O((|\mathcal{A}| + G^2 + \sigma^2)/\sqrt{N} + E_g)$, where $G^2 = \frac{1}{2}(G_g^2 + G_g^2)$ and $\sigma^2 = \frac{1}{2}(\sigma_g^2 + \sigma_g^2)$. Therefore, DYNA is effectively twice as fast as the pure model-free approach when the model is accurate. However, it would eventually suffer from the performance bias due model error, as reflected in the term $E_g$. We will demonstrate this property experimentally in Figure 1.

Finally, we note that the idea of using $\Phi_n$ as control variate (Chebotar et al., 2017; Grathwohl et al., 2018; Papini et al., 2018) is orthogonal to the setups considered above, and it can be naturally combined with PicCoLO. For example, we can also use $\Phi_n$ to compute a better sampled gradient $g_n$ with smaller variance (line 5 of Algorithm 1). This would improve $\sigma_g^2$ in the bounds of PicCoLO to a smaller $\sigma_g^2$, the size of reduced variance.

6. Experiments

We corroborate our theoretical findings with experiments\(^6\) in learning neural network policies to solve robot RL tasks (CartPole, Hopper, Snake, and Walker3D) from OpenAI Gym (Brockman et al., 2016) with the DART physics engine (Lee et al., 2018).\(^7\) The aim is to see if PicCoLO improves the performance of a base algorithm, even though in these experiments the convexity assumption in the theory does not hold. We choose several popular first-order mir-

\(^4\)The codes are available at https://github.com/gtrll/rlfamily.
\(^5\)The environments are defined in DartEnv, hosted at https://github.com/DartEnv.
In Figure 1, we first use CartPole to study Theorem 2, which suggests that PicCOLO is unbiased and improves the performance when the prediction is accurate. Here we additionally consider an extremely bad model, ADVERSARIAL, that predicts the gradients adversarially. It still converges when the model is adversarially attacking the algorithm, whereas DYN A fails completely. In Figure 1 (b), we conduct a finer comparison of the effects of different model accuracies (BIASED DYN-FP), when \( \hat{g}_n \) is computed using (8). To realize inaccurate dynamics models to be used in the Prediction Step, we change the mass of links of the robot by a certain factor, e.g. BIASED DYN0.8 indicates that the mass of each individual link is either increased or decreased by 80% with probability 0.5, respectively. We see that the fixed-point formulation (8), which makes multiple queries of \( \Phi_n \) for computing \( \hat{g}_n \), performs much better than the heuristic of setting \( \hat{g}_n = \Phi(\tilde{\pi}_n) \), even when the latter is using the true model (TRUE DYN). Overall, we see PicCOLO with BIASED DYN-FP is able to accelerate learning, though with a degree varying with model accuracies; but even for models with a large bias, it still converges unbiasedly, as we previously observed in Figure 1 (a),

In Figure 2, we study the performance of PicCOLO in a range of environments. In general, we find that PicCOLO indeed improves the performance\(^{10}\) though the exact degree depends on how \( \hat{g}_n \) is computed. In Figure 2 (a) and (b), we show the results of using ADAM as the base algorithm. We observe that, while setting \( \hat{g}_n = \Phi_n(\tilde{\pi}_n) \) is already an effective heuristic, the performance of PicCOLO can be further and largely improved if we adopt the fixed-point strategy in (8), as the latter allows the learner to take more globally informed update directions. Finally, to demonstrate the flexibility of the proposed framework, we also “PicCOLO” two other base algorithms, NATGRAD and TRPO, in Figure 2 (c) and (d), respectively. The complete set of experimental results can be found in Appendix H.

7. Conclusion

PicCOLO is a general reduction-based framework for solving predictable online learning problems. It can be viewed as an automatic strategy for generating new algorithms that can leverage prediction to accelerate convergence. Furthermore, PicCOLO uses the Correction Step to recover from the mistake made in the Prediction Step, so the presence of modeling errors does not bias convergence, as we show in both the theory and experiments. The design of PicCOLO leaves open the question of how to design good 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 can make non-trivial predictions. In the experiments, we found that off-policy and simulated gradients are often useful, but they are not perfect. It would be interesting to see whether a predictive model that is trained to directly minimize the prediction error can further help policy learning. Finally, we note that, despite the focus of this paper on policy optimization, PicCOLO can naturally be applied to other optimization and learning problems.

---

\(^{8}\)In implementation, we solve the corresponding optimization problem with a few number of iterations. For example, BIASED DYN-FP is approximately solved with 5 iterations.

\(^{9}\)We set \( g_{n+1} = -\left( \max_{m=1,\ldots,n} \| g_m \| / \| g_n \| \right) g_n \).

\(^{10}\)Note that different base algorithms are not directly comparable, as further fine-tuning of step sizes is required.
References


