ASYNCHRONOUS ADVANTAGE ACTOR CRITIC: NON-ASYMPTOTIC ANALYSIS AND LINEAR SPEEDUP

Anonymous authors
Paper under double-blind review

ABSTRACT

Asynchronous and parallel implementation of standard reinforcement learning (RL) algorithms is a key enabler of the tremendous success of modern RL. Among many asynchronous RL algorithms, arguably the most popular and effective one is the asynchronous advantage actor-critic (A3C) algorithm. Although A3C is becoming the workhorse of RL, its theoretical properties are still not well-understood, including the non-asymptotic analysis and the performance gain of parallelism (a.k.a. speedup). This paper revisits the A3C algorithm with TD(0) for the critic update, termed A3C-TD(0), with provable convergence guarantees. With linear value function approximation for the TD update, the convergence of A3C-TD(0) is established under both i.i.d. and Markovian sampling. Under i.i.d. sampling, A3C-TD(0) obtains sample complexity of $\mathcal{O}(\epsilon^{-2.5}/N)$ per worker to achieve ϵ accuracy, where N is the number of workers. Compared to the best-known sample complexity of $\mathcal{O}(\epsilon^{-2.5})$ for two-timescale AC, A3C-TD(0) achieves *linear speedup*, which justifies the advantage of parallelism and asynchrony in AC algorithms theoretically for the first time. Numerical tests on synthetically generated instances and OpenAI Gym environments have been provided to verify our theoretical analysis.

1 INTRODUCTION

Reinforcement learning (RL) has achieved impressive performance in many domains such as robotics [1, 2] and video games [3]. However, these empirical successes are often at the expense of significant computation. To unlock high computation capabilities, the state-of-the-art RL approaches rely on sampling data from massive parallel simulators on multiple machines [3, 4, 5]. Empirically, these approaches can stabilize the learning processes and *reduce training time* when they are implemented in an asynchronous manner. One popular RL method that often achieves the best empirical performance is the asynchronous variant of the actor-critic (AC) algorithm, which is referred to as A3C [3].

A3C builds on the original AC algorithm [6]. At a high level, AC simultaneously performs policy optimization (a.k.a. the actor step) using the policy gradient method [7] and policy evaluation (a.k.a. the critic step) using the temporal difference learning (TD) algorithm [8]. To ensure scalability, both actor and critic steps can combine with various function approximation techniques. To ensure stability, AC is often implemented in a two time-scale fashion, where the actor step runs in the slow timescale and the critic step runs in the fast timescale. Similar to other on-policy RL algorithms, AC uses samples generated from the target policy. Thus, data sampling is entangled with the learning procedure, which generates significant *overhead*. To speed up the sampling process of AC, A3C introduces multiple workers with a shared policy, and each learner has its own simulator to perform data sampling. The shared policy can be then updated using samples collected from multiple learners.

Despite the tremendous empirical success achieved by A3C, to the best of our knowledge, its theoretical property is not well-understood. The following *theoretical* questions remain unclear: **Q1)** Under what assumption does A3C converge? **Q2)** What is its convergence rate? **Q3)** Can A3C obtain benefit (or speedup) using parallelism and asynchrony?

For **Q3**), we are interested in the *training time linear speedup* with N workers, which is the ratio between the training time using a single worker and that using N workers. Since asynchronous parallelism mitigates the effect of stragglers and keeps all workers busy, the training time speedup

can be measured roughly by the sample (i.e., computational) complexity linear speedup [9], given by

$$\text{Speedup}(N) = \frac{\text{sample complexity when using one worker}}{\text{average sample complexity per worker when using } N \text{ workers}}. \tag{1}$$

If $\text{Speedup}(N) = \Theta(N)$, the speedup is linear, and the training time roughly reduces linearly as the number of workers increases. This paper aims to answer these questions, towards the goal of providing theoretical justification for the empirical successes of parallel and asynchronous RL.

1.1 Related works

Analysis of actor critic algorithms. AC method was first proposed by [6, 10], with asymptotic convergence guarantees provided in [6, 10, 11]. It was not until recently that the *non-asymptotic* analyses of AC have been established. The finite-sample guarantee for the batch AC algorithm has been established in [12, 13] with i.i.d. sampling. Later, in [14], the finite-sample analysis was established for the double-loop nested AC algorithm under the Markovian setting. An improved analysis for the Markovian setting with minibatch updates has been presented in [15] for the nested AC method. More recently, [16, 17] have provided the first finite-time analyses for the two-timescale AC algorithms under Markov sampling, with both $\tilde{O}(\epsilon^{-2.5})$ sample complexity, which is the best-known sample complexity for two-timescale AC. Through the lens of bi-level optimization, [18] has also provided finite-sample guarantees for this two-timescale Markov sampling setting, with global optimality guarantees when a *natural* policy gradient step is used in the actor. However, none of the existing works has analyzed the effect of the asynchronous and parallel updates in AC.

Empirical parallel and distributed AC. In [3], the original A3C method was proposed and became the workhorse in empirical RL. Later, [19] has provided a GPU-version of A3C which significantly decreases training time. Recently, the A3C algorithm is further optimized in modern computers by [20], where a large batch variant of A3C with improved efficiency is also proposed. In [21], an importance weighted distributed AC algorithm IMPALA has been developed to solve a collection of problems with one single set of parameters. Recently, a gossip-based distributed yet synchronous AC algorithm has been proposed in [5], which has achieved the performance competitive to A3C.

Asynchronous stochastic optimization. For solving general optimization problems, asynchronous stochastic methods have received much attention recently. The study of asynchronous stochastic methods can be traced back to 1980s [22]. With the batch size M, [23] analyzed asynchronous SGD (async-SGD) for convex functions, and derived a convergence rate of $\mathcal{O}(K^{-\frac{1}{2}}M^{-\frac{1}{2}})$ if delay K_0 is bounded by $\mathcal{O}(K^{\frac{1}{4}}M^{-\frac{3}{4}})$. This result implies linear speedup. [24] extended the analysis of [23] to smooth convex with nonsmooth regularization and derived a similar rate. Recent studies by [25] improved upper bound of K_0 to $\mathcal{O}(K^{\frac{1}{2}}M^{-\frac{1}{2}})$. However, all these works have focused on the single-timescale SGD with a single variable, which cannot capture the stochastic recursion of the AC and A3C algorithms. To best of our knowledge, non-asymptotic analysis of asynchronous two-timescale SGD has remained unaddressed, and its speedup analysis is even an uncharted territory.

1.2 This work

In this context, we revisit A3C with TD(0) for the critic update, termed A3C-TD(0). The hope is to provide *non-asymptotic* guarantee and *linear speedup* justification for this popular algorithm.

Our contributions. Compared to the existing literature on both the AC algorithms and the async-SGD, our contributions can be summarized as follows.

c1) We revisit two-timescale A3C-TD(0) and establish its convergence rates with both i.i.d. and Markovian sampling. To the best of our knowledge, this is the first non-asymptotic convergence result for *asynchronous parallel* AC algorithms.

c2) We characterize the sample complexity of A3C-TD(0). In i.i.d. setting, A3C-TD(0) achieves a sample complexity of $\mathcal{O}(\epsilon^{-2.5}/N)$ per worker, where N is the number of workers. Compared to the best-known complexity of $\mathcal{O}(\epsilon^{-2.5})$ for i.i.d. two-timescale AC [18], A3C-TD(0) achieves *linear speedup*, thanks to the parallelism and asynchrony. In the Markovian setting, if delay is bounded, the sample complexity of A3C-TD(0) matches the order of the non-parallel AC algorithm [17].

c3) We test A3C-TD(0) on the synthetically generated environment to verify our theoretical guarantees with both i.i.d. and Markovian sampling. We also test A3C-TD(0) on the classic control tasks and Atari Games from OpenAI Gym. **Code is available in the supplementary material.**

Technical challenges. Compared to the recent convergence analysis of nonparallel two-timescale AC in [16, 17, 18], several new challenges arise due to the parallelism and asynchrony.

Markovian noise coupled with asynchrony and delay. The analysis of two-timescale AC algorithm is non-trivial because of the Markovian noise coupled with both the actor and critic steps. Different from the nonparallel AC that only involves a single Markov chain, asynchronous parallel AC introduces multiple Markov chains (one per worker) that mix at different speed. This is because at a given iteration, workers collect different number of samples and thus their chains mix to different degrees. As we will show later, the worker with the slowest mixing chain will determine the convergence.

Linear speedup for SGD with two coupled sequences. Parallel async-SGD has been shown to achieve linear speedup recently [9, 26]. Different from async-SGD, asynchronous AC is a two-timescale stochastic *semi-gradient* algorithm for solving the more challenging *bilevel* optimization problem (see [18]). The errors induced by asynchrony and delay are intertwined with both actor and critic updates via a nested structure, which makes the sharp analysis more challenging. Our linear speedup analysis should be also distinguished from that of mini-batch async-SGD [27], where the speedup is a result of *variance reduction* thanks to the larger batch size generated by parallel workers.

2 PRELIMINARIES

2.1 MARKOV DECISION PROCESS AND POLICY GRADIENT THEOREM

RL problems are often modeled as an MDP described by $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma\}$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}(s'|s,a)$ is the probability of transitioning to $s' \in \mathcal{S}$ given current state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$, and $r(s,a,s')$ is the reward associated with the transition (s,a,s'), and $\gamma \in (0,1)$ is a discount factor. Throughout the paper, we assume the reward r is upper-bounded by a constant $r_{\max}$. A policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ is defined as a mapping from the state space $\mathcal{S}$ to the probability distribution over the action space $\mathcal{A}$.

Considering discrete time t in an infinite horizon, a policy π can generate a trajectory of state-action pairs $(s_0, a_0, s_1, a_1, \ldots)$ with $a_t \sim \pi(\cdot|s_t)$ and $s_{t+1} \sim \mathcal{P}(\cdot|s_t, a_t)$. Given a policy π, we define the state and state action value functions as

$$V_\pi(s) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) \mid s_0 = s\right], \; Q_\pi(s,a) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) \mid s_0 = s, a_0 = a\right] \quad (2)$$

where $\mathbb{E}$ is taken over the trajectory $(s_0, a_0, s_1, a_1, \ldots)$ generated under policy π. With the above definitions, the advantage function is $A_\pi(s,a) := Q_\pi(s,a) - V_\pi(s)$. With η denoting the initial state distribution, the discounted state visitation measure induced by policy π is defined as $d_\pi(s) := (1-\gamma)\sum_{t=0}^{\infty} \gamma^t \mathbb{P}(s_t = s \mid s_0 \sim \eta, \pi)$, and the discounted state action visitation measure is $d'_\pi(s,a) = (1-\gamma)\sum_{t=0}^{\infty} \gamma^t \mathbb{P}(s_t = s \mid s_0 \sim \eta, \pi)\pi(a|s)$.

The goal of RL is to find a policy that maximizes the expected accumulative reward $J(\pi) := \mathbb{E}_{s\sim\eta}[V_\pi(s)]$. When the state and action spaces are large, finding the optimal policy π becomes computationally intractable. To overcome the inherent difficulty of learning a function, the policy gradient methods search the best performing policy over a class of parameterized policies. We parameterize the policy with parameter $\theta \in \mathbb{R}^d$, and solve the optimization problem as

$$\max_{\theta \in \mathbb{R}^d} J(\theta) \quad \text{with} \quad J(\theta) := \mathop{\mathbb{E}}_{s\sim\eta}[V_{\pi_\theta}(s)]. \quad (3)$$

To maximize $J(\theta)$ with respect to θ, one can update θ using the policy gradient direction given by [7]

$$\nabla J(\theta) = \mathop{\mathbb{E}}_{s,a\sim d'_\theta}\left[A_{\pi_\theta}(s,a)\psi_\theta(s,a)\right], \quad (4)$$

where $\psi_\theta(s,a) := \nabla \log \pi_\theta(a|s)$, and $d'_\theta := (1-\gamma)\sum_{t=0}^{\infty} \gamma^t \mathbb{P}(s_t = s \mid s_0, \pi_\theta)\pi_\theta(a|s)$. Since computing $\mathbb{E}$ in (4) is expensive if not impossible, popular policy gradient-based algorithms iteratively update θ using stochastic estimate of (4) such as REINFORCE [28] and G(PO)MDP [29].

2.2 ACTOR-CRITIC ALGORITHM WITH VALUE FUNCTION APPROXIMATION

Both REINFORCE and G(PO)MDP-based policy gradient algorithms rely on a Monte-Carlo estimate of the value function $V_{\pi_\theta}(s)$ and thus $\nabla J(\theta)$ by generating a trajectory per iteration. However, policy gradient methods based on Monte-Carlo estimate typically suffer from high variance and large sampling cost. An alternative way is to recursively refine the estimate of $V_{\pi_\theta}(s)$. For a policy π_θ, it is known that $V_{\pi_\theta}(s)$ satisfies the Bellman equation [30], that is

$$V_{\pi_\theta}(s) = \mathop{\mathbb{E}}_{a\sim\pi_\theta(\cdot|s),\, s'\sim\mathcal{P}(\cdot|s,a)} \left[r(s,a,s') + \gamma V_{\pi_\theta}(s')\right], \quad \forall s \in \mathcal{S}. \tag{5}$$

In practice, when the state space $\mathcal{S}$ is prohibitively large, one cannot afford the computational and memory complexity of computing $V_{\pi_\theta}(s)$ and $A_{\pi_\theta}(s,a)$. To overcome this curse-of-dimensionality, a popular method is to approximate the value function using function approximation techniques. Given the state feature mapping $\phi(\cdot) : \mathcal{S} \to \mathbb{R}^{d'}$ for some $d' > 0$, we approximate the value function linearly as $V_{\pi_\theta}(s) \approx \hat{V}_\omega(s) := \phi(s)^\top \omega$, where $\omega \in \mathbb{R}^{d'}$ is the critic parameter.

Given a policy π_θ, the task of finding the best ω such that $V_{\pi_\theta}(s) \approx \hat{V}_\omega(s)$ is usually addressed by TD learning [8]. Defining the kth transition as $x_k := (s_k, a_k, s_{k+1})$ and the corresponding TD-error as $\hat{\delta}(x_k, \omega_k) := r(s_k, a_k, s_{k+1}) + \gamma\phi(s_{k+1})^\top\omega_k - \phi(s_k)^\top\omega_k$, the parameter ω is updated via

$$\omega_{k+1} = \Pi_{R_\omega}\big(\omega_k + \beta_k g(x_k, \omega_k)\big) \quad \text{with} \quad g(x_k, \omega_k) := \hat{\delta}(x_k, \omega_k)\nabla_{\omega_k}\hat{V}_{\omega_k}(s_k) \tag{6}$$

where β_k is the critic stepsize, and Π_{R_ω} is the projection with R_ω being a pre-defined constant.

Using the definition of advantage function $A_{\pi_\theta}(s,a) = \mathbb{E}_{s'\sim\mathcal{P}}[r(s,a,s') + \gamma V_{\pi_\theta}(s')] - V_{\pi_\theta}(s)$, we can also rewrite (4) as $\nabla J(\theta) = \mathbb{E}_{s,a\sim d'_\theta, s'\sim\mathcal{P}}\left[\left(r(s,a,s') + \gamma V_{\pi_\theta}(s') - V_{\pi_\theta}(s)\right)\psi_\theta(s,a)\right]$. Leveraging the value function approximation, we can then approximate the policy gradient as

$$\widehat{\nabla} J(\theta) = \Big(r(s,a,s') + \gamma\hat{V}_\omega(s') - \hat{V}_\omega(s)\Big)\,\psi_\theta(s,a) = \hat{\delta}(x,\omega)\psi_\theta(s,a) \tag{7}$$

which gives rise to the policy update

$$\theta_{k+1} = \theta_k + \alpha_k v(x_k, \theta_k, \omega_k) \quad \text{with} \quad v(x_k, \theta_k, \omega_k) := \hat{\delta}(x_k, \omega_k)\psi_{\theta_k}(s_k, a_k) \tag{8}$$

where α_k is the stepsize for the actor update.

To ensure convergence when simultaneously performing critic and actor updates, the stepsizes α_k and β_k often decay at two different rates, which is referred to the two-timescale AC [17, 18].

3 ASYNCHRONOUS ADVANTAGE ACTOR CRITIC WITH TD(0)

To speed up the training process, we implement AC over N workers in a shared memory setting without coordinating among workers — a setting similar to that in A3C [3]. Each worker has its own simulator to perform sampling, and then collaboratively updates the shared policy π_θ using AC updates. As there is no synchronization after each update, the policy used by workers to generate samples may be outdated, which introduces staleness.

Notations on transition (s, a, s')**.** Since each worker will maintain a separate Markov chain, we thereafter use subscription t in (s_t, a_t, s_{t+1}) to indicate the tth transition on a Markov chain. We use k to denote the global counter (or iteration), which increases by one whenever a worker finishes the actor and critic updates in the shared memory. We use subscription (k) in $(s_{(k)}, a_{(k)}, s'_{(k)})$ to indicate the transition used in the kth update.

Specifically, we initialize θ_0, ω_0 in the shared memory. Each worker will initialize the simulator with initial state s_0. Without coordination, workers will read θ, ω in the shared memory. From each worker's view, it then generates sample (s_t, a_t, s_{t+1}) by either sampling s_t from $\mu_\theta(\cdot)$, where $\mu_\theta(\cdot)$ is the stationary distribution of an artificial MDP with transition probability measure $\widetilde{\mathcal{P}}(\cdot|s_t, a_t) := \gamma\mathcal{P}(\cdot|s_t, a_t) + (1-\gamma)\eta(\cdot)$, or, sampling s_t from a Markov chain under policy π_θ. In both cases, each worker obtains $a_t \sim \pi_\theta(\cdot|s_t)$ and $s_{t+1} \sim \widetilde{\mathcal{P}}(\cdot|s_t, a_t)$. Sampling s_{t+1} from $\widetilde{\mathcal{P}}(\cdot|s_t, a_t)$ can be achieved by sampling s_{t+1} from $\eta(\cdot)$ with probability $1-\gamma$ and from $\mathcal{P}(\cdot|s_t, a_t)$ otherwise. Once

Algorithm 1 Asynchronous advantage AC with TD(0): each worker's view.

Global initialize: Global counter $k = 0$, initial θ_0, ω_0 in the shared memory.
Worker initialize: Local counter $t = 0$. Obtain initial state s_0.
for $t = 0, 1, 2, \cdots$ **do**
  Read θ, ω in the shared memory.
  Option 1 (i.i.d. sampling):
    Sample $s_t \sim \mu_\theta(\cdot)$, $a_t \sim \pi_\theta(\cdot|s)$, $s_{t+1} \sim \widetilde{\mathcal{P}}(\cdot|s_t, a_t)$.
  Option 2 (Markovian sampling):
    Sample $a_t \sim \pi_\theta(\cdot|s_t)$, $s_{t+1} \sim \widetilde{\mathcal{P}}(\cdot|s_t, a_t)$.
  Compute $\hat{\delta}(x_t, \omega) = r(s_t, a_t, s_{t+1}) + \gamma \hat{V}_\omega(s_{t+1}) - \hat{V}_\omega(s_t)$.
  Compute $g(x_t, \omega) = \hat{\delta}(x_t, \omega) \nabla_\omega \hat{V}_\omega(s_t)$.
  Compute $v(x_t, \theta, \omega) = \hat{\delta}(x_t, \omega) \psi_\theta(s_t, a_t)$.
  In the shared memory, perform update (9).
end for

obtaining $x_t := (s_t, a_t, s_{t+1})$, each worker locally computes the policy gradient $v(x_t, \theta, \omega)$ and the TD(0) update $g(x_t, \omega)$, and then updates the parameters in shared memory asynchronously by

$$\omega_{k+1} = \Pi_{R_\omega} \left(\omega_k + \beta_k g(x_{(k)}, \omega_{k-\tau_k}) \right), \tag{9a}$$

$$\theta_{k+1} = \theta_k + \alpha_k v(x_{(k)}, \theta_{k-\tau_k}, \omega_{k-\tau_k}), \tag{9b}$$

where τ_k is the delay in the kth actor and critic updates. See the A3C with TD(0) in Algorithm 1.

Parallel sampling. The AC update (6) and (8) uses samples generated "on-the-fly" from the target policy π_θ, which brings overhead. Compared with (6) and (8), the A3C-TD(0) update (9) allows parallel sampling from N workers, which is the key to linear speedup. We consider the case where only one worker can update parameters in the shared memory at the same time and the update cannot be interrupted. In practice, (9) can also be performed in a mini-batch fashion.

Minor differences from A3C [3]. The A3C-TD(0) algorithm resembles the popular A3C method [3]. With $n_{\max}$ denoting the horizon of steps, for $n \in \{1, ..., n_{\max}\}$, A3C iteratively uses n-step TD errors to compute actor and critic gradients. In A3C-TD(0), we use the TD(0) method which is the 1-step TD method for actor and critic update. When $n_{\max} = 1$, A3C method reduces to A3C-TD(0). We here focus on A3C with TD(0) just for ease of exposition, and we believe that the techniques here can also be useful for analyzing A3C with multi-step TD, which is left as our future work.

4 Convergence Analysis of Two-Timescale A3C-TD(0)

In this section, we analyze the convergence of A3C-TD(0) in both i.i.d. and Markovian settings. Throughout this section, the notation $\mathcal{O}(\cdot)$ contains constants that are independent of N and K_0.

To analyze the performance of A3C-TD(0), we make the following assumptions.

Assumption 1. *There exists K_0 such that the delay at each iteration is bounded by $\tau_k \leq K_0, \forall k$.*

Assumption 1 ensures the viability of analyzing the asynchronous update; see the same assumption in e.g., [5, 25]. In practice, the delay usually scales as the number of workers, that is $K_0 = \Theta(N)$.

With $\mathcal{P}_{\pi_\theta}(s'|s) = \sum_{a \in \mathcal{A}} \mathcal{P}(s'|s, a)\pi_\theta(a|s)$, we define that:

$$A_{\theta,\phi} := \mathop{\mathbb{E}}_{s \sim \mu_\theta, s' \sim \mathcal{P}_{\pi_\theta}} [\phi(s)(\gamma\phi(s') - \phi(s))^\top], \qquad b_{\theta,\phi} := \mathop{\mathbb{E}}_{s \sim \mu_\theta, a \sim \pi_\theta, s' \sim \mathcal{P}} [r(s, a, s')\phi(s)]. \tag{10}$$

It is known that the stationary point ω_θ^* of the TD(0) update in (6) satisfies $A_{\theta,\phi}\omega_\theta^* + b_{\theta,\phi} = 0$.

Assumption 2. *For all $s \in \mathcal{S}$, the feature vector $\phi(s)$ is normalized so that $\|\phi(s)\|_2 \leq 1$. For all $\theta \in \mathbb{R}^d$, $A_{\theta,\phi}$ is negative definite and its maximum eigenvalue is upper bounded by constant $-\lambda$.*

Assumption 2 is common in analyzing TD with linear function approximation; see e.g., [17, 31, 32]. With this assumption, $A_{\theta,\phi}$ is invertible, so we have $\omega_\theta^* = -A_{\theta,\phi}^{-1} b_{\theta,\phi}$. Define $R_\omega := r_{\max}/\lambda$, then we have $\|\omega_\theta^*\|_2 \leq R_\omega$. It also justifies the projection introduced in Algorithm 1.

Assumption 3. *For any $\theta, \theta' \in \mathbb{R}^d$, $s \in \mathcal{S}$ and $a \in \mathcal{A}$, there exist constants such that: i) $\|\psi_\theta(s,a)\|_2 \leq C_\psi$; ii) $\|\psi_\theta(s,a) - \psi_{\theta'}(s,a)\|_2 \leq L_\psi \|\theta - \theta'\|_2$; iii) $|\pi_\theta(a|s) - \pi_{\theta'}(a|s)| \leq L_\pi \|\theta - \theta'\|_2$.*

Assumption 3 is common in analyzing policy gradient-type algorithms which has also been made by e.g., [33, 34, 35]. This assumption holds for many policy parameterization methods such as tabular softmax policy [34], Gaussian policy [36] and Boltzmann policy [37].

Assumption 4. *For any $\theta \in \mathbb{R}^d$, the Markov chain under policy π_θ and transition kernel $\mathcal{P}(\cdot|s,a)$ or $\widetilde{\mathcal{P}}(\cdot|s,a)$ is irreducible and aperiodic. Then there exist constants $\kappa > 0$ and $\rho \in (0,1)$ such that*

$$\sup_{s \in \mathcal{S}} \quad d_{TV}\left(\mathbb{P}(s_t \in \cdot | s_0 = s, \pi_\theta), \mu_\theta\right) \leq \kappa \rho^t, \quad \forall t \tag{11}$$

where μ_θ is the stationary state distribution under π_θ, and s_t is the state of Markov chain at time t.

Assumption 4 assumes the Markov chain mixes at a geometric rate; see also [31, 32]. The stationary distribution μ_θ of an artificial Markov chain with transition $\widetilde{\mathcal{P}}$ is the same as the discounted visitation measure d_θ of the Markov chain with transition $\mathcal{P}$ [6]. This means that if we sample according to $a_t \sim \pi_\theta(\cdot|s_t)$, $s_{t+1} \sim \widetilde{\mathcal{P}}(\cdot|s_t, a_t)$, the marginal distribution of (s_t, a_t) will converge to the discounted state-action visitation measure $d'_\theta(s,a)$, which allows us to control the gradient bias.

4.1 Linear speedup result with i.i.d. sampling

In this section, we consider A3C-TD(0) under the i.i.d. sampling setting, which is widely used for analyzing RL algorithms; see e.g., [13, 18, 38].

We first give the convergence result of critic update as follows.

Theorem 1 (Critic convergence)**.** *Suppose Assumptions 1–4 hold. Consider Algorithm 1 with i.i.d. sampling and $\hat{V}_\omega(s) = \phi(s)^\top \omega$. Select step size $\alpha_k = \frac{c_1}{(1+k)^{\sigma_1}}$, $\beta_k = \frac{c_2}{(1+k)^{\sigma_2}}$, where $0 < \sigma_2 < \sigma_1 < 1$ and c_1, c_2 are positive constants. Then it holds that*

$$\frac{1}{K}\sum_{k=1}^{K} \mathbb{E}\left\|\omega_k - \omega^*_{\theta_k}\right\|_2^2 = \mathcal{O}\left(\frac{1}{K^{1-\sigma_2}}\right) + \mathcal{O}\left(\frac{1}{K^{2(\sigma_1-\sigma_2)}}\right) + \mathcal{O}\left(\frac{K_0^2}{K^{2\sigma_2}}\right) + \mathcal{O}\left(\frac{K_0}{K^{\sigma_1}}\right) + \mathcal{O}\left(\frac{1}{K^{\sigma_2}}\right). \tag{12}$$

Different from async-SGD (e.g., [9]), the optimal critic parameter ω^*_θ is constantly drifting as θ changes at each iteration. This necessitates setting $\sigma_1 > \sigma_2$ to make the policy change slower than the critic, which can be observed in the second term in (12). If $\sigma_1 > \sigma_2$, then the policy is static relative to the critic in an asymptotic sense.

To introduce the convergence of actor update, we first define the critic approximation error as

$$\epsilon_{app} \coloneqq \max_{\theta \in \mathbb{R}^d} \sqrt{\mathop{\mathbb{E}}_{s \sim \mu_\theta} |V_{\pi_\theta}(s) - \hat{V}_{\omega^*_\theta}(s)|^2}, \tag{13}$$

where μ_θ is the stationary distribution under π_θ and $\widetilde{\mathcal{P}}$. This error captures the quality of the critic function approximation; see also [14, 15, 17]. Now we are ready to give the actor convergence.

Theorem 2 (Actor convergence)**.** *Under the same assumptions of Theorem 1, select step size $\alpha_k = \frac{c_1}{(1+k)^{\sigma_1}}$, $\beta_k = \frac{c_2}{(1+k)^{\sigma_2}}$, where $0 < \sigma_2 < \sigma_1 < 1$ and c_1, c_2 are positive constants. Then it holds that*

$$\frac{1}{K}\sum_{k=1}^{K} \mathbb{E}\left\|\nabla J(\theta_k)\right\|_2^2 = \mathcal{O}\left(\frac{1}{K^{1-\sigma_1}}\right) + \mathcal{O}\left(\frac{K_0}{K^{\sigma_1}}\right) + \mathcal{O}\left(\frac{K_0^2}{K^{2\sigma_2}}\right) + \mathcal{O}\left(\frac{1}{K}\sum_{k=1}^{K} \mathbb{E}\left\|\omega_k - \omega^*_{\theta_k}\right\|_2^2\right) + \mathcal{O}(\epsilon_{app}). \tag{14}$$

Different from the analysis of async-SGD, in actor update, the stochastic gradient $v(x, \theta, \omega)$ is biased because of inexact value function approximation. The bias introduced by the critic optimality gap and the function approximation error correspond to the last two terms in (14).

In Theorem 1 and Theorem 2, optimizing σ_1 and σ_2 gives the following convergence rate.

Corollary 1 (Linear speedup)**.** *Given Theorem 1 and Theorem 2, select $\sigma_1 = \frac{3}{5}$ and $\sigma_2 = \frac{2}{5}$. If we further assume $K_0 = \mathcal{O}(K^{\frac{1}{5}})$, then it holds that*

$$\frac{1}{K}\sum_{k=1}^{K} \mathbb{E}\left\|\nabla J(\theta_k)\right\|_2^2 = \mathcal{O}\left(K^{-\frac{2}{5}}\right) + \mathcal{O}(\epsilon_{app}) \tag{15}$$

where $\mathcal{O}(\cdot)$ contains constants that are independent of N and K_0.

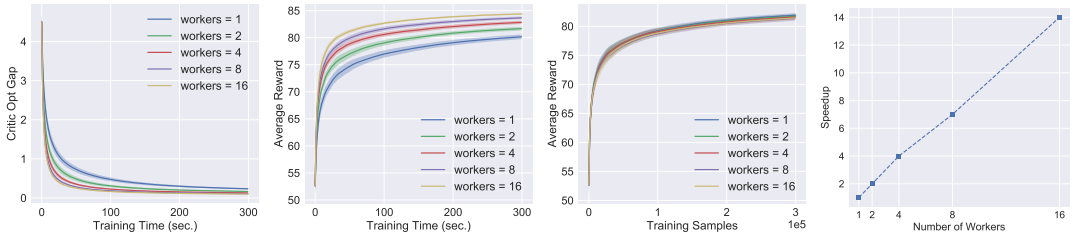

Figure 1: Convergence results of A3C-TD(0) with i.i.d. sampling in synthetic environment.

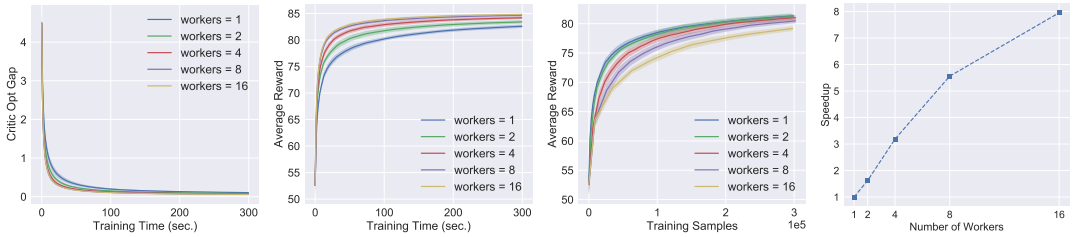

Figure 2: Convergence results of A3C-TD(0) with Markovian sampling in synthetic environment.

By setting the first term in (15) to ϵ, we get the total iteration complexity to reach ϵ-accuracy is $\mathcal{O}(\epsilon^{-2.5})$. Since each iteration only uses one sample (one transition), it also implies a total sample complexity of $\mathcal{O}(\epsilon^{-2.5})$. Then the average sample complexity per worker is $\mathcal{O}(\epsilon^{-2.5}/N)$ which indicates linear speedup in (1). Intuitively, the negative effect of parameter staleness introduced by parallel asynchrony vanishes asymptotically, which implies linear speedup in terms of convergence.

4.2 Convergence result with Markovian sampling

Theorem 3 (Critic convergence)**.** *Suppose Assumptions 1–4 hold. Consider Algorithm 1 with Markovian sampling and $\hat{V}_\omega(s) = \phi(s)^\top \omega$. Select step size $\alpha_k = \frac{c_1}{(1+k)^{\sigma_1}}$, $\beta_k = \frac{c_2}{(1+k)^{\sigma_2}}$, where $0 < \sigma_2 < \sigma_1 < 1$ and c_1, c_2 are positive constants. Then it holds that*

$$\frac{1}{K}\sum_{k=1}^{K}\mathbb{E}\left\|\omega_k - \omega^*_{\theta_k}\right\|_2^2 = \mathcal{O}\left(\frac{1}{K^{1-\sigma_2}}\right)+\mathcal{O}\left(\frac{1}{K^{2(\sigma_1-\sigma_2)}}\right)+\mathcal{O}\left(\frac{K_0^2}{K^{2\sigma_2}}\right)+\mathcal{O}\left(\frac{K_0^2\log^2 K}{K^{\sigma_1}}\right)+\mathcal{O}\left(\frac{K_0\log K}{K^{\sigma_2}}\right). \tag{16}$$

The following theorem gives the convergence rate of actor update in Algorithm 1.

Theorem 4 (Actor convergence)**.** *Under the same assumptions of Theorem 3, select step size $\alpha_k = \frac{c_1}{(1+k)^{\sigma_1}}$, $\beta_k = \frac{c_2}{(1+k)^{\sigma_2}}$, where $0 < \sigma_2 < \sigma_1 < 1$ and c_1, c_2 are positive constants. Then it holds that*

$$\frac{1}{K}\sum_{k=1}^{K}\mathbb{E}\left\|\nabla J(\theta_k)\right\|_2^2 = \mathcal{O}\left(\frac{1}{K^{1-\sigma_1}}\right)+\mathcal{O}\left(\frac{K_0^2\log^2 K}{K^{\sigma_1}}\right)+\mathcal{O}\left(\frac{K_0^2}{K^{2\sigma_2}}\right)+\mathcal{O}\left(\frac{1}{K}\sum_{k=1}^{K}\mathbb{E}\left\|\omega_k - \omega^*_{\theta_k}\right\|_2^2\right)+\mathcal{O}(\epsilon_{app}). \tag{17}$$

Assume $K_0 = \mathcal{O}(K^{\frac{1}{5}})$. Given Theorem 3, select $\sigma_1 = \frac{3}{5}$ and $\sigma_2 = \frac{2}{5}$, then it holds that

$$\frac{1}{K}\sum_{k=1}^{K}\mathbb{E}\left\|\nabla J(\theta_k)\right\|_2^2 = \widetilde{\mathcal{O}}\left(K_0 K^{-\frac{2}{5}}\right) + \mathcal{O}(\epsilon_{app}), \tag{18}$$

where $\widetilde{\mathcal{O}}(\cdot)$ hides constants and the logarithmic order of K.

With Markovian sampling, the stochastic gradients $g(x,\omega)$ and $v(x,\theta,\omega)$ are biased, and the bias decreases as the Markov chain mixes. The mixing time corresponds to the logarithmic term $\log K$ in (16) and (17). Because of asynchrony, at a given iteration, workers collect different number of samples and their chains mix to different degrees. The worker with the slowest mixing chain will determine the rate of convergence. The product of K_0 and $\log K$ in (16) and (17) appears due to the slowest mixing chain. As the last term in (16) dominates other terms asymptotically, the convergence rate reduces as the number of workers increases. While the theoretical linear speedup is difficult to establish in the Markovian setting, we will empirically demonstrate it in Section 5.2.

Figure 3: Speedup of A3C-TD(0) in OpenAI gym classic control task (Carpole).

Figure 4: Speedup of A3C-TD(0) in OpenAI Gym Atari game (Seaquest).

5 Numerical Experiments

We test the speedup performance of A3C-TD(0) on both synthetically generated and OpenAI Gym environments. **The settings, parameters, and codes are provided in supplementary material**.

5.1 A3C-TD(0) in synthetic environment

To verify the theoretical result, we tested A3C-TD(0) with linear value function approximation in a synthetic environment. We use tabular softmax policy parameterization [34], which satisfies Assumption 3. The MDP has a state space $|\mathcal{S}| = 100$, an discrete action space of $|\mathcal{A}| = 5$. Each state feature has a dimension of 10. Elements of the transition matrix, the reward and the state features are randomly sampled from a uniform distribution over $(0, 1)$. We evaluate the convergence of actor and critic respectively with the running average of test reward and critic optimality gap $\|\omega_k - \omega^*_{\theta_k}\|_2$.

Figures 1 and 2 show the training time and sample complexity of running A3C-TD(0) with i.i.d. sampling and Markovian sampling respectively. The speedup plot is measured by the number of samples needed to achieve a target running average reward under different number of workers. All the results are average over 10 Monte-Carlo runs. Figure 1 shows that the sample complexity of A3C-TD(0) stays about the same with different number of workers under i.i.d. sampling. Also, it can be observed from the speedup plot of Figure 1 that the A3C-TD(0) achieves roughly linear speedup with i.i.d. sampling, which is consistent with Corollary 1. The speedup of A3C-TD(0) with Markovian sampling shown in Figure 2 is roughly linear when number of workers is small.

5.2 A3C-TD(0) in OpenAI Gym environments

We have also tested A3C-TD(0) with neural network parametrization in the classic control and the Atari game environments. Figures 3 and 4 show the speedup of A3C-TD(0) under different number of workers, where the average reward is computed by taking the running average of test rewards. The speedup and runtime speedup plots are respectively measured by the number of samples and training time needed to achieve a target running average reward under different number of workers. Although not justified theoretically, Figures 3 and 4 suggest that the sample complexity speedup is roughly linear, and the runtime speedup slightly degrades when the number of workers increases. This is partially due to our hardware limit. Similar observation has also been obtained in async-SGD [9].

6 Conclusions

This paper revisits the A3C algorithm with TD(0) for the critic update, termed A3C-TD(0). With linear value function approximation, the convergence of the A3C-TD(0) algorithm has been established under both i.i.d. and Markovian sampling settings. Under i.i.d. sampling, A3C-TD(0) achieves linear speedup compared to the best-known sample complexity of two-timescale AC, theoretically justifying the benefit of parallelism and asynchrony for the first time. Under Markov sampling, such a linear speedup can be observed in most classic benchmark tasks.

REFERENCES

[1] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *Proc. of International Conference on Learning Representations*, 2016.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[3] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning." in *Proc. of International Conference on Machine Learning*, 2016.

[4] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen *et al.*, "Massively parallel methods for deep reinforcement learning," *arXiv preprint:1507.04296*, 2015.

[5] M. Assran, J. Romoff, N. Ballas, J. Pineau, and M. Rabbat, "Gossip-based actor-learner architectures for deep reinforcement learning," in *Proc. of Advances in Neural Information Processing Systems*, 2019.

[6] V. Konda, *Actor-critic algorithms.* PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 2002.

[7] R. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation." in *Proc. of Advances in Neural Information Processing Systems*, 2000.

[8] R. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, 1988.

[9] X. Lian, H. Zhang, C. Hsieh, Y. Yijun Huang, and J. Liu, "A comprehensive linear speedup analysis for asynchronous stochastic parallel optimization from zeroth-order to first-order," in *Proc. of the Advances in Neural Information Processing Systems*, 2016.

[10] V. Borkar and V. Konda, "The actor-critic algorithm as multi-time-scale stochastic approximation," *Sadhana*, vol. 22, no. 4, pp. 525–543, 1997.

[11] S. Bhatnagar, R. Sutton, M. Ghavamzadeh, and M. Lee, "Natural actor critic algorithms," *Automatica*, vol. 45, pp. 2471–2482, 2009.

[12] Z. Yang, K. Zhang, M. Hong, and T. Başar, "A finite sample analysis of the actor-critic algorithm," in *Proc. of IEEE Conference on Decision and Control*, 2018, pp. 2759–2764.

[13] H. Kumar, A. Koppel, and A. Ribeiro, "On the sample complexity of actor-critic method for reinforcement learning with function approximation," *arXiv preprint:1910.08412*, 2019.

[14] S. Qiu, Z. Yang, J. Ye, and Z. Wang, "On the finite-time convergence of actor-critic algorithm," in *Optimization Foundations for Reinforcement Learning Workshop at Advances in Neural Information Processing Systems*, 2019.

[15] T. Xu, Z. Wang, and Y. Liang, "Improving sample complexity bounds for (natural) actor-critic algorithms," in *Proc. of Advances in Neural Information Processing Systems*, 2020.

[16] ——, "Non-asymptotic convergence analysis of two time-scale (natural) actor-critic algorithms," *arXiv preprint:2005.03557*, 2020.

[17] Y. Wu, W. Zhang, P. Xu, and Q. Gu, "A finite time analysis of two time-scale actor critic methods," in *Proc. of Advances in Neural Information Processing Systems*, 2020.

[18] M. Hong, H.-T. Wai, Z. Wang, and Z. Yang, "A two-timescale framework for bilevel optimization: Complexity analysis and application to actor-critic," *arXiv preprint:2007.05170*, 2020.

[19] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz, "Reinforcement learning through asynchronous advantage actor-critic on a gpu," in *Proc. of International Conference on Learning Representations*, 2017.

[20] A. Stooke and P. Abbeel, "Accelerated methods for deep reinforcement learning," *arXiv preprint:1803.02811*, 2019.

[21] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," *arXiv preprint:1802.01561*, 2018.

[22] D. Bertsekas and J. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Prentice-Hall, 1989.

[23] A. Agarwal and J. Duchi, "Distributed delayed stochastic optimization," in *Proc. of Advances in Neural Information Processing Systems*, 2011.

[24] H. Feyzmahdavian, A. Aytekin, and M. Johansson, "An asynchronous mini-batch algorithm for regularized stochastic optimization," *arXiv preprint:1505.04824*, 2015.

[25] X. Lian, Y. Huang, Y. Li, and J. Liu, "Asynchronous parallel stochastic gradient for nonconvex optimization." in *Proc. of Advances in Neural Information Processing Systems*, 2015.

[26] T. Sun, R. Hannah, and W. Yin, "Asynchronous coordinate descent under more realistic assumptions," in *Proc. of Advances in Neural Information Processing Systems*, 2017.

[27] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent," in *Proc. of Advances in Neural Information Processing Systems*, 2017.

[28] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, May 1992.

[29] J. Baxter and P. L. Bartlett, "Infinite-horizon policy-gradient estimation," *J. Artificial Intelligence Res.*, vol. 15, pp. 319–350, 2001.

[30] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT Press, 2018.

[31] J. Bhandari, D. Russo, and R. Singal, "A finite time analysis of temporal difference learning with linear function approximation." in *Proc. of Conference on Learning Theory*, 2018.

[32] T. Xu, Z. Wang, Y. Zhou, and Y. Liang, "Reanalysis of variance reduced temporal difference learning," in *Proc. of International Conference on Learning Representations*, 2020.

[33] K. Zhang, A. Koppel, H. Zhu, and T. Başar, "Global convergence of policy gradient methods to (almost) locally optimal policies," *arXiv preprint:1906.08383*, 2019.

[34] A. Agarwal, S. M. Kakade, J. D. Lee, and G. Mahajan, "Optimality and approximation with policy gradient methods in markov decision processes." in *Proc. of Thirty Third Conference on Learning Theory*, 2020.

[35] S. Zou, T. Xu, and Y. Liang, "Finite-sample analysis for SARSA with linear function approximation," in *Proc. of Advances in Neural Information Processing Systems*, 2019.

[36] K. Doya, "Reinforcement learning in continuous time and space," *Neural Computation*, vol. 12, no. 1, pp. 219–245, 2000.

[37] V. Konda and V. Borkar, "Actor-critic–type learning algorithms for markov decision processes," *SIAM Journal on Control and Optimization*, vol. 38, no. 1, pp. 94–123, 1999.

[38] R. Sutton, H. Maei, D. Precup, S. Bhatnagar, D. Silver, and E. Szepesvári, C.and Wiewiora, "Fast gradient-descent methods for temporal-difference learning with linear function approximation," in *Proc. of International Conference on Machine Learning*, 2009.

[39] Dgriff, "Pytorch implementation of a3c," https://github.com/dgriff777/rl_a3c_pytorch, 2018.