# Effective Linear Policy Gradient Search through Primal-Dual Approximation

Yiming Peng, Gang Chen, Mengjie Zhang
School of Engineering and Computer Science
Victoria University of Wellington
Wellington, New Zealand
Email: {yiming.peng, aaron.chen, mengjie.zhang}@ecs.vuw.ac.nz

***Abstract*—Recent research discovered that Reinforcement Learning (RL) algorithms with simple linear policies can achieve competitive performance as many state-of-the-art RL algorithms designed to train policies in the form of multi-layer neural networks. However, high learning performance is only achieved so far when policies are trained by jointly using multiple episodes of samples. An important research question remains as to whether linear policies can achieve cutting-edge performance when they are trained in a step-wise fashion (i.e., policies are iteratively updated based on every newly obtained sample). This paper presents a confirmatory answer to this question by developing a new RL algorithm called Primal-Dual Regular-gradient Actor-Critic (PD-RAC) as a generalization of RAC, which is a popular step-wise RL technique. Experiments on six benchmark control problems show that PD-RAC can achieve leading performance, in comparison to several recently developed baseline algorithms.**

***Index Terms*—Reinforcement Learning, Actor Critic, Episodic Learning, Step-wise Learning, Primal-Dual Approximation**

## I. Introduction

Reinforcement learning (RL) has been successfully applied to a wide range of tasks from intelligent game play [1] to locomotion control [2]. A main family of RL algorithms called *Policy Gradient Search* (PGS) is designed to directly train a policy model based on gradients of its trainable parameter. PGS is particularly effective at solving various control problems with continuous state spaces and continuous action spaces. Depending on whether the policy model is linear or non-linear, it can be implemented as either a linear combination of a group of parameters or a multi-layer neural network (NN). Linear policies enjoy three significant advantages over non-linear policies: (1) they are easier to be interpreted and understood [3]; (2) they are more efficient in terms of computational cost [4]; (3) they have strong theoretical guarantees of convergence [3]–[5]. However, linear policies are often considered not as competent as non-linear policies at solving complex problems [2], [6]–[8].

Against this common belief, Rajeswaran et al. [9] for the first time showed that linear policies trained via natural gradient learning can effectively solve continuous control tasks. Shortly after that, Mania et al. [10] proposed the Augmented Random Search (ARS) algorithm, which is capable of training linear policies to achieve state-of-the-art performance on many difficult benchmark control tasks. All these algorithms train linear policies by jointly using multiple episodes of environment samples. This is because policy gradient estimation based on a single sample alone can easily suffer from high-level of noise in step-wise learning systems [9]. Such noise can be noticeably reduced when estimating policy gradient over a collection of samples.

Nevertheless, step-wise learning is much more efficient to operate than episodic learning. It also improves the adaptability of an RL algorithm and enhances its sample efficiency [3]. Therefore, it is important to develop new step-wise RL algorithms with proven capability of training linear policies highly reliably and effectively. Driven by this understanding, we aim to address a critical challenge in this paper: upon updating a linear policy in a step-wise fashion along its policy gradient, no mechanism exists to re-use vital historical information that can improve the accuracy of the estimated policy gradient [6], [11]–[13]. Without explicitly maintaining multiple episodes of samples obtained in the past, we decide to tackle this issue by efficiently accumulating and re-using historical gradients while estimating policy gradients in future steps.

To systematically re-use historical gradients, we adopt a general *Primal-Dual* (PD) approximation technique [14] that converts challenging primal learning problems to simpler linear dual problems through averaged historical gradients accompanied with a strongly convex regularization term. In fact, the dual problems can be treated as locally linear estimations of the original primal problems and can often be solved analytically, resulting in low-variance learning performance [15]. Although PD has been utilized by the mirror descent technique to facilitate safe RL, to the best of our knowledge, this important method has seldom been applied to step-wise PGS algorithms.

Motivated by the great potential of PD at enhancing step-wise RL, our main goal is to develop a linear PGS algorithm that can tackle various control problems effectively and reliably. In particular, we aim to achieve four research objectives as follows:

1) To develop a new AC algorithm termed PD-RAC based on a newly derived dual formulation of the policy gradient search problem.
2) To prove that PD-SAC generalizes the basic RAC algorithm for step-wise RL by analyzing the dual formulation of PD-RAC.

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

3) To show that PD-RAC converges to local equilibrium by providing a theoretical analysis.
4) To empirically evaluate the learning performance of PD-RAC, in comparison to RAC and two cutting-edge PS algorithms (i.e., ARS and PPO-Linear) on six benchmark continuous control problems.

The paper is structured as follows. Section II presents the background and relate work to form a foundation of this paper. Next, Section III derives the new policy updating rule with the help of PD approximation, and develops the new algorithm PD-RAC. Meanwhile, an analysis is performed to testify the fact that RAC is in fact a special case of PD-RAC. The design of experiments and the discussion on results are given in Section V and Section VI respectively. The paper concludes in Section VII.

## II. Background and Related Work

In this section, we briefly describe the background including the concepts of reinforcement learning, Actor-Critic Architecture and the general Primal-Dual method. Following that, we present a discussion of related works in the literature.

### A. Reinforcement Learning

Reinforcement Learning is defined as a process where an agent interacts with an unknown environment through a series of actions [3]. The environment provides responsive feedback in the form of rewards to the actions it receives from the agent. Through RL, the agent has the goal to maximize its long-term pay-off, which is defined as

$$\mathcal{J}(\pi) = \mathbf{E}[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | \pi]. \tag{1}$$

This goal is realized by learning a policy $\pi$ that maximizes $\mathcal{J}$ in (1), i.e.,

$$\begin{aligned} \pi^* &= \text{argmax}_\pi \, \mathcal{J}(\pi) \\ &= \text{argmax}_\pi \, \mathbf{E}[\textstyle\sum_{t=0}^{\infty} \gamma^t r_{t+1} | \pi]. \end{aligned} \tag{2}$$

### B. Actor-Critic Architecture

The *Actor-Critic* (AC) architecture is popularly used to design PGS algorithms [3], [6], [16]. An AC algorithm explicitly maintains and learns both a policy (i.e., actor) and a value function (i.e., critic). The value function is used for policy evaluation. For this purpose, the state value function $V^\pi$ is trained via the *temporal-difference* (TD) error below,

$$\delta_t^\pi = \mathcal{R}(\vec{s}_t, a_t, \vec{s}_{t+1}) + \gamma V^\pi(\vec{s}_{t+1}) - V^\pi(\vec{s}_t), \tag{3}$$

where $V^\pi(\vec{s}_t)$ approximates the expected long-term pay-off for current state $\vec{s}_t$, and $\vec{s}_{t+1}$ refers to the next state reachable by taking action $a_t$ at state $\vec{s}_t$.

Based on $\delta_t^\pi$ in (3), $V^\pi$ can be iteratively updated according to,

$$\vec{v}_{t+1} = \vec{v}_t + \alpha_t \delta_t^\pi \vec{\phi}(\vec{s}_t), \tag{4}$$

Simultaneously, actor training in AC algorithms, such as RAC [16], is realized by using the estimated policy gradient, i.e., the gradient of cumulative rewards $\mathcal{J}$ with respect to all policy parameters $\vec{\theta}$,

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \beta_t \nabla_{\vec{\theta}} \mathcal{J}(\vec{\theta}_t), \tag{5}$$

where

$$\nabla_{\vec{\theta}} \mathcal{J}(\vec{\theta}_t) = \int_{\vec{s} \in \mathcal{S}} p^\pi(\vec{s}) \int_{a \in \mathcal{A}(\vec{s})} \frac{\partial \pi_{\vec{\theta}}(a|\vec{s})}{\partial \vec{\theta}} \tilde{Q}^\pi_{\vec{w}^\star}(\vec{s}, a) da d\vec{s} \tag{6}$$

### C. A General Primal-Dual Approximation Method

This section introduces the core notion of PD that paves the way for developing PD-RAC in Section III. [14] showed that it is possible to approximate a complicated primal learning problem through a simpler linear dual problem that can be solved immediately. Given a general optimization/learning problem (i.e., ***the primal problem***)

$$f(\vec{x}^\star) = \max_{\vec{x} \in \mathcal{D}} f(\vec{x}), \tag{7}$$

where $\mathcal{D} \subseteq \mathbb{R}^m$ is a convex subset of the real vector space, and the convex scalar objective function $f : \mathcal{D} \to \mathbb{R}$ is Lipschitz continuous. To solve (7), [14] reformulates (7) to a simpler ***dual problem*** based on $k$ ($k > 0$) candidate solutions $\{\vec{x}_i\}_{i=0}^k \subset \mathcal{D}$, as shown below:

$$\begin{aligned} l_k(\vec{x}^\star) = \max_{\vec{x} \in \mathcal{D}} [ & \frac{1}{k+1} \sum_{i=0}^{k} [f(\vec{x}_i) + \nabla_{\vec{x}} f(\vec{x}_i)^T \cdot (\vec{x} - \vec{x}_i)] \\ & - \mu_k d(\vec{x}, \vec{x}_0)], \end{aligned} \tag{8}$$

where $\mu_k = \frac{1}{2\beta(k+1)}$ is a scaling parameter, and $d(\vec{x}, \vec{x}_0)$ is an arbitrary distance measure between any two solutions. In order to provide $k$ candidate solutions, we can adopt an iterative process. Particularly, during each learning iteration, (8) can be solved directly to produce the next candidate solution as

$$\begin{aligned} \vec{x}_{k+1} &= \vec{x}_0 + \beta \tfrac{1}{k+1} \textstyle\sum_{i=0}^{k} \nabla_{\vec{x}} f(\vec{x}_i) \\ &= \vec{x}_k + \beta \tfrac{1}{k+1} \nabla_{\vec{x}} f(\vec{x}_k), \end{aligned} \tag{9}$$

where $\beta$ is the learning rate.

Unfortunately the dual formulation of (9) cannot effectively re-use historical gradients. Therefore, we must investigate other variations of the dual formulation, which have not been explored in-depth in the literature.

### D. Related Work

Since Deep-Q gained a great success on autonomously playing Atari games, surpassing human expert players [1], RL has received unprecedented research interests [17]. A variety of interesting algorithms have been developed recently to address many difficult RL problems. Among all these algorithms, PGS methods have attracted increasing attention [17]. Different from value function based methods, such as Deep-Q, PGS avoids the necessity of using the $\epsilon$-greedy action selection mechanism that may dramatically change an RL agent's behavior subject to a very small change in the value function [3]. This issue has been shown to substantially degrade the effectiveness of Deep-Q on problems with large number of possible actions [5].

Many PGS algorithms have achieved cutting-edge performance onto many difficult problems. For example, Schulman et al. introduced PPO and TRPO to directly train policies modeled as deep neural networks [7], [8]. The two algorithms can noticeably outperform many value function based algorithms in terms of both sample efficiency and effectiveness. Overwhelmed by the huge success of PGS algorithms on training non-linear policies, few people has ever attempted to re-consider linear policies which stand for a popular choice of policy model before the age of deep reinforcement learning.

Very recently, two brilliant PGS algorithms successfully reignite the interests in linear policies. The first algorithm is proposed by [9] where a natural-gradient based episodic learning approach is adopted to learn a linear policy effectively. A more recent algorithm is proposed by [10] to search policies in the linear space through a guided random search mechanism. Regardless of the policy model adopted, all the aforementioned methods are designed to train policies based on either the current episode of samples or a mixed collection of both current and historical samples. In other words, they all fall under the episodic learning paradigm.

The research question remains regarding whether it is possible to develop a step-wise learning algorithm that can train linear policies reliably and effectively. In this paper, we aim to address this question by proposing a new PGS algorithm that is capable of conducting step-wise learning of linear policies with highly competent performance. To achieve this goal, we will introduce a new PD-inspired mechanism to effectively reuse historical gradients rather than historical samples to improve the accuracy of estimated policy gradients.

## III. Primal-Dual Regular Actor-Critic Algorithm

In this section, we start with deriving a new and more useful dual formulation for the PD-RAC algorithm. Following that, an relationship analysis PD-RAC and RAC is presented. Lastly, an algorithmic description of PD-RAC is given.

### A. Dual Formulation for PD-RAC

According to (7), the primal problem for policy training can be defined as

$$\begin{aligned}\mathcal{J}(\vec{\theta}^{\star}) &= \max_{\vec{\theta}\in\vec{\Theta}}\mathcal{J}(\vec{\theta}) \\ &= \max_{\vec{\theta}\in\vec{\Theta}}[\int_{\vec{s}\in\mathcal{S}} p^{\pi_{\vec{\theta}}}\pi_{\vec{\theta}}(a|\vec{s})\mathcal{R}(\vec{s},a,\vec{s}')dad\vec{s}]\end{aligned} \tag{10}$$

Instead of using the dual formulation given in (8), which is equivalent to RAC, we change the arithmetic averaging in (8) to the exponentially-weighted averaging in (11). Besides this, several possible variations can be considered. For example, we can set the norm of the gradients as the weights [14]. This implies that only the directions of the gradients will be considered during the learning process. However, we cannot find any empirical evidences that support its effective use in our RL algorithm.

Based on this understanding, the dual problem of (8) can be re-formulated as

$$\begin{aligned}l(\vec{\theta}^{\star}) = \max_{\vec{\theta}\in\Theta}[&\sum_{i=0}^{t}\rho^{t-i}[\mathcal{J}(\vec{\theta}_i)+\nabla_{\vec{\theta}}\mathcal{J}(\vec{\theta}_i)^T\cdot(\vec{\theta}-\vec{\theta}_i)] \\ &-\mu_t||\vec{\theta}-\vec{\theta}_0||_2^2].\end{aligned} \tag{11}$$

Clearly, exponentially-weighted averaging is applied to (11). Particularly, the weight paramter $\rho \in (0,1]$ measures the importance of historical gradients (i.e., gradients obtained from the prior $t$ time steps) at an exponential scale. In the meantime, we decide to use Euclidean distance as the regularization term for simplicity. To analytically solve (11), note that

$$\begin{aligned}\frac{\partial l_t'(\vec{\theta})}{\partial\vec{\theta}} &= \frac{\partial[\frac{1}{t+1}\sum_{i=0}^{t}[f(\vec{\theta}_i)+(\frac{\partial\mathcal{J}(\vec{\theta}_i)}{\partial\vec{\theta}_i})^T\cdot(\vec{\theta}-\vec{\theta}_i)]+\mu_t d(\vec{\theta})]}{\partial\vec{\theta}} \\ &= (\frac{1}{t+1})\sum_{i=0}^{t}(\frac{\partial\mathcal{J}(\vec{\theta}_i)}{\partial\vec{\theta}_i})+\mu_t\frac{\partial d(\vec{\theta})}{\partial\vec{\theta}}.\end{aligned} \tag{12}$$

By letting (12) equal to $\vec{0}$, we have

$$(\frac{1}{t+1})\sum_{i=0}^{t}(\frac{\partial\mathcal{J}(\vec{\theta}_i)}{\partial\vec{\theta}_i})+\mu_t\frac{\partial d(\vec{\theta})}{\partial\vec{\theta}}=\vec{0}, \tag{13}$$

Based on (13), we can generalize policy training by introducing a tunable parameter $0<\rho\leq 1$,

$$\frac{1}{(t+1)\sum_{i=0}^{t}\rho^{t-i}}\sum_{i=0}^{t}[\rho^{t-i}(\frac{\partial\mathcal{J}(\vec{\theta}_i)}{\partial\vec{\theta}_i})]=\mu_t\frac{\partial d(\vec{\theta})}{\partial\vec{\theta}}, \tag{14}$$

According to the definition of $\mu_t$ in [14] and (13), the scaling parameter $\mu_t$ can be determined as:

$$\mu_t=\frac{\sum_{i=0}^{t}\rho^{t-i}}{2t(t+1)\sum_{i=0}^{t}\rho^{t-i}}, \tag{15}$$

Thereby, (14) can be reformulated as,

$$\sum_{i=0}^{t}[\rho^{t-i}(\frac{\partial\mathcal{J}(\vec{\theta}_i)}{\partial\vec{\theta}_i})]=(\frac{\sum_{i=0}^{t}\rho^{t-i}}{2t})\frac{\partial d(\vec{\theta})}{\partial\vec{\theta}}. \tag{16}$$

(16) hence enables PD-RAC to generalize RAC. Specifically, when $\rho=1$, policy learning in PD-RAC degenerates to the average model in (13). Additionally, the scaling parameter is obtained as $\mu_t=\frac{\sum_{i=0}^{t}\rho^{t-i}}{2(t+1)}$. Following the above derivations, we can obtain the policy parameter updating rule for PD-RAC as:

$$\vec{\theta}_{t+1}\leftarrow\vec{\theta}_0+\beta_t(\frac{t+1}{\sum_{i=0}^{t}\rho^{t-i}})\sum_{i=0}^{t}[\rho^{t-i}\nabla_{\vec{\theta}}\mathcal{J}(\vec{\theta}_i)], \tag{17}$$

where $\beta_t$ is the learning rate at time step $t$.

### B. Analysis on Dual Formulation of PD-RAC

By using the exponentially-weighted averaging method, we emphasize more on recently obtained policy gradients. Furthermore, the updating of policy parameters can be made at a smaller scale, in comparison to value function learning. During RL, the value function is expected to become more and more precise, progressively improving the accuracy of estimated policy gradients. Hence, over $t$ consecutive learning

steps, the gradients obtained in early steps (e.g., when $t = 0$) may not be as important and accurate as those obtained in later steps. However, in (8), historical gradients are treated equally, ignoring their varied importance. To cope with this issue, we introduce the dual problem formulation in (11), where gradients obtained more recently are considered more important. Moreover, in the literature, several research works showed that adaptive changes of weights can potentially result in better convergence rate [18]. We also performed some preliminary experiments. Our study confirms that the exponentially-weighted averaging method can noticeably improve the convergence rate, in comparison to the arithmetic averaging method.

However, there is a key issue of directly using (17). Specifically, when $t$ becomes very large, the influence of the policy gradients estimated at times closer to $t = 0$ will diminish completely, resulting in biased learning.

A simple example in Figure 1 helps to demonstrate this issue. Here, we use a 2D contour graph to represent the policy parameter space. Each parameter value is represented as a black point. The red dashed vectors represent the normal updating trajectories from $\vec{\theta}_0$ to $\vec{\theta}_{t+1}$, obtained by following RAC. On the other hand, considering the updating rule (17), its second part is depicted as a solid blue vector, i.e., $\frac{t+1}{\sum_{i=0}^{t}\rho^{t-i}}\sum_{i=0}^{t}\rho^{t-i}\nabla_{\vec{\theta}}\mathcal{J}(\vec{\theta}_t)$. This is because, when $t$ is very large, the policy gradient (the dashed blue vector) is largely determined by the recent gradients, namely the two parts A and B shown in the figure. Upon following the direction of the solid blue vector (i.e., the dashed blue vector) to update $\vec{\theta}_0$, the updated parameter $\vec{\theta}'_{t+1}$ will end up at an undesirable position in the figure. In fact, $\vec{\theta}'_{t+1}$ and $\vec{\theta}_0$ are located roughly at the same contour line, resulting in no performance improvement.

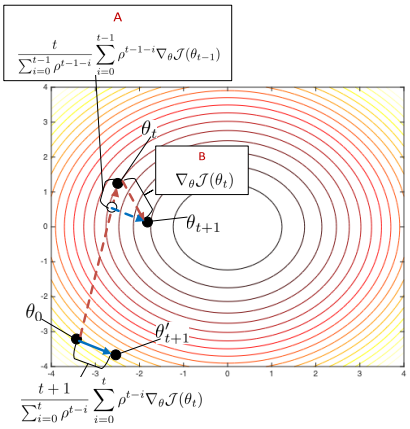


Fig. 1: An example of the biased learning following (17) when $t$ is very large.

To address the issue demonstrated in Figure 1, we develop a periodical updating process for learning the policy parameters. To be more specific, policy parameters are updated every $K$ (i.e.,the periodic interval) steps where $K > 0$ is a small constant. After every $K$ steps, we will apply the updating rule (17), as a result, $\vec{\theta}_0$ becomes $\vec{\theta}_K$. Accordingly, (17) can be re-written as a more general periodic updating rule below:

$$\vec{\theta}_{(n+1)K} \leftarrow \vec{\theta}_{nK} + \beta_{nK}\left(\frac{nK}{\sum_{i=0}^{nK}\rho^{n(K-i)}}\right)\sum_{i=0}^{nK}[\rho^{n(K-i)}\nabla_{\vec{\theta}}\mathcal{J}(\vec{\theta}_i)], \tag{18}$$

where $n \in \mathbb{N}$. Meanwhile, within one learning period, the step-wise updating rule below will be followed:

$$\vec{\theta}_{t+1} \leftarrow \vec{\theta}_t + \beta_t \nabla_{\vec{\theta}}\mathcal{J}(\vec{\theta}_i). \tag{19}$$

where $0 \leq t < K$. Note that, with periodical learning, RAC can still be viewed as a special case of PD-RAC. When $\rho = 1$, the proposed updating rules in (18) and (19) are equivalent to (5), which is adopted by RAC.

### C. PD-RAC Algorithm

Following (18) and (19) for policy learning, we present the complete PD-RAC algorithm in Algorithm 1. In the algorithm, the regular gradient estimator at each learning step is realized based on $\nabla_{\vec{\theta}}\mathcal{J}(\vec{\theta}_i) = \mathbf{E}_{\vec{\theta}}[\delta_i^{\pi_i}\Phi(\vec{s}_i, a_i)]$, where $\delta_i^{\pi_i}$ is obtained from (3) and $\Phi(\vec{s}, a) = \frac{\partial \ln \pi_{\vec{\theta}}(a|\vec{s})}{\partial \vec{\theta}}$ is the compatible feature [5].

**Algorithm 1** PD-RAC Algorithm

**Require:** an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, the periodic step interval $K$.
**Ensure:** $\vec{\theta}$, $\vec{v}^\pi$

*Initialization*:
$\vec{\theta} \leftarrow \vec{\theta}_0$, $\vec{v}^\pi \leftarrow v_0^\pi$, $\vec{s}_t \leftarrow \vec{s}_0$, $\vec{\tilde{g}} \leftarrow \vec{0}$, $\vec{g} \leftarrow \vec{0}$, $k \leftarrow 0$
*Learning Process for one episode:*
**for** $t = 0, 1, 2, \ldots$ **do**
  $a_t \sim \pi_{\vec{\theta}}(a|\vec{s}_t)$
  Take action $a_t$, observe reward $r_{t+1}$ and new state $\vec{s}_{t+1}$
  $\delta_t^\pi \leftarrow r_{t+1} + \gamma \vec{v}_t^{\pi T} \cdot \vec{\phi}(\vec{s}_{t+1}) - \vec{v}_t^{\pi T} \cdot \vec{\phi}(\vec{s}_t)$
  $\vec{v}_{t+1}^\pi \leftarrow \vec{v}_t^\pi + \vec{\alpha}\delta_t^\pi \vec{\phi}(\vec{s}_t)$
  $\vec{\tilde{g}} \leftarrow \vec{\tilde{g}} + \rho\delta_t^\pi \vec{\Phi}(\vec{s}_t, a_t)$
  $k \leftarrow k + 1$
  $\vec{g} \leftarrow \frac{k}{\sum_{i=0}^{k}\rho^{k-i}}\vec{\tilde{g}}$
  $\vec{\theta}_{t+1} \leftarrow \vec{\theta}_t + \beta\vec{\Phi}(\vec{s}_t, a_t)$
  **if** $k \geq K$ **then**
    $\vec{\theta}_{t+1} \leftarrow \vec{\theta}_0 + \beta\vec{g}$
    $\vec{\theta}_0 \leftarrow \vec{\theta}_{t+1}$
    $k \leftarrow 0$
  **end if**
**end for**
$\vec{\theta}_0 \leftarrow \vec{\theta}_{t+1}$, $\vec{v}^\pi \leftarrow v_0^\pi$, $\vec{s}_t \leftarrow \vec{s}_0$, $\vec{\tilde{g}} \leftarrow \vec{0}$, $\vec{g} \leftarrow \vec{0}$, $k \leftarrow 0$
**return** $\vec{\theta}$, $\vec{v}^\pi$

## IV. Theoretical Analysis

In this section, we briefly touch on the theoretical analysis, where Proposition 1 is presented to guarantee the convergence

of PD-RAC. Due to the space limitation, we here provide a general idea of proof rather than the actual rigorous mathematical proof.

The convergence analysis of PD-RAC follows and extends the analysis of RAC presented in [16]. Similar to other convergence analysis in literature, the PR-RAC converges under six important assumptions. To avoid unnecessary repetitions, we refer readers to related papers [16], [19], [20] to detailed mathematical descriptions for all **six** essential assumptions. Following this, we propose the proposition below:

**Proposition 1.** *Under Assumptions 1 - 6 in [16], [19], [20], given some small $\eta > 0$ [1] and $\epsilon > 0$, $\exists \delta > 0$ such that for $\vec{\theta}_t$, $t \geq 0$ obtained from PD-RAC, if $\sum_{i=0}^{t} \sup_{\vec{\theta}_i} ||e^{\vec{\theta}_i}|| < t\delta$, also $\nabla_{\vec{\theta}} \mathcal{J}(\vec{\theta})$ and $\nabla^2_{\vec{\theta}} \mathcal{J}(\vec{\theta})$ are bounded, then $\vec{\theta}_t \to \Im^{\epsilon}$ as $t \to \infty$ with probability one.*

In what follows, we will discuss the key ideas of proof in a high-level, which is to find a way to categorize PD-RAC into the two time-scale learning process [3], [5], [16]. Distinct from the single-step parameter updating process of RAC, PD-RAC features a $k$-step parameter updating process where $k$ is a pre-defined step-length. Thus, we can start with a single-step updating to prove the convergence, then it can be extended to the case $k = 2$ and further to the case where $k \to \infty$.

*Firstly*, let us recall that the single-step policy parameter learning can be regarded as,

$$\vec{\theta}_{t+1} = \vec{\theta}_t - \beta_t \nabla_{\vec{\theta}_t} \mathcal{J}(\vec{\theta}_t). \tag{20}$$

Following (20), we can obtain a sequence of policy parameters over time, i.e.,

$$\vec{\theta}_0, \ldots, \vec{\theta}_t.$$

*Secondly*, let us expand the above pattern to think $k$-step as a single step shown as below,

$$\underbrace{\vec{\theta}_0, \ldots, \vec{\theta}_k}_{\hat{\vec{\theta}}_1}, \underbrace{\vec{\theta}_{k+1}, \ldots, \vec{\theta}_{k+k}}_{\hat{\vec{\theta}}_2}, \ldots, \underbrace{\vec{\theta}_{t-k}, \ldots, \vec{\theta}_t}_{\hat{\vec{\theta}}_t}.$$

Accordingly, we can have another variant single-step updating sequence of parameters, i.e.,

$$\hat{\vec{\theta}}_0, \ldots, \hat{\vec{\theta}}_t.$$

Thus, following the findings in previous research [16], we can easily understand the convergence of sequence $\hat{\vec{\theta}}_t$ if each $\hat{\vec{\theta}}_t$ is bounded.

*Thirdly*, with some derivation by following Taylor Expansion, we can show that if $\eta$ is reasonably small, the update to $\vec{\theta}_k$ when $k \geq 2$ can be bounded as,

$$\begin{aligned} \vec{\theta}_k \leq & \vec{\theta}_0 - (k-1)A(B+C+D)\nabla_{\vec{\theta}_0} \mathcal{J}(\vec{\theta}_0) \\ & - (k-1)A[\rho e^{\vec{\theta}_0}(\vec{s}_0) + e^{\vec{\theta}_1}(\vec{s}_1) \\ & + \cdots + e^{\vec{\theta}_{k-1}}(\vec{s}_{k-1})]. \end{aligned} \tag{21}$$

[1] $\eta$ is a new learning rate introduced to mitigate the affect of higher order terms, when expanding the first order gradient of $\mathcal{J}$, i.e., $\nabla_{\vec{\theta}} \mathcal{J}(\vec{\theta})$, with Taylor Series. Thus, the value must be assumed to be small enough.

*Lastly*, following similar steps in the proof for Theorem 2 in [16] and Lemma 6 in [16], we can draw the conclusion that the learning process of $\vec{\theta}$ converges to a local equilibrium.

## V. Design of Experiments

In this section, we evaluate the proposed PD-RAC on six benchmarked continuous control tasks including Bipedal Walker, LunarLander, Mountain Car Continuous, Inverted Pendulum, Inverted Double Pendulum, and Inverted Pendulum Swingup. We briefly describe each problem used in experiments in the first subsection below. More detailed descriptions about these problems can be found in [21]–[23]. Subsequently, the second subsection presents the detailed setups for experiments, including competing algorithms, representation of value function and policy, and hyper-parameter configurations. Lastly, the experiment design is presented.

### A. Benchmark Problems

Among the six benchmark problems, Mountain Car Continuous, Lunar Lander, and Bipedal Walker are implemented provided by GYM environment [24]. The other three problems are Inverted Pendulum, Inverted Double Pendulum and Inverted Pendulum Swingup provided by Bullet Physics Engine [25]. We briefly describe each problem used in experiments in the rest of the subsection, and more detailed descriptions about these problems can be found in [21]–[23].

- *Mountain Car Continuous* is a continuous version of the classic Mountain Car Problem [3] where a car is positioned between two "mountains" in a one-dimensional axis. The goal is to drive the car up to the mountain on the right side. To make the problem difficult, the car's engine is not strong enough to reach the goal in a single pass. The instant reward is the distance from the current car position to the goal region, and a +10 is directly given once the car reaches the goal region. It is designed to have one continuous action in [-1.0, 1.0].
- *Lunar Lander* is to control an agent accepting 8-dimensional continuous sensor input to produce a two-dimensional continuous action ranging from $-1.0$ to $1.0$. It aims to smoothly and accurately guide the lander robot to land on a target pad which is always set at the origin $(0.0,\ 0.0)$. While moving from the top of the screen to the target pad with zero speed, the agent will be awarded a reward ranging from $100$ to $140$. However, it loses rewards due to its moving away from the pad. As long as the lander crashes or comes to rest, it receives an additional $-100$ or $100$, and the episode completes.
- *Bipedal Walker* is to drive a robot move along flat terrain. It is constituted of 24-dimensional continuous state space and 4-dimensional continuous action space. The agent is rewarded $+1$ point by moving forward, and total $+300$ points is given at the far end. The fallen of the robot will cause a $-100$ penalty; also motor torque costs a small number of points.
- *Inverted Pendulum* is the classical pole balancing problem, where a pole is attached by a joint to a cart moving

horizontally. It aims to find a plot that balances the pole to the upright angle as long as possible. It is designed to have four state inputs and one continuous action in $[-1.0, 1.0]$. As long as the pole maintains upright, it receives a $+1$ reward.

- *Inverted Pendulum Swingup* is an analogy to Inverted Pendulum, but it requires additional swing to balance the pole to the upright position.
- *Inverted Double Pendulum* is a hardcore version of Inverted Pendulum, as it contains two joints connecting two poles to a fixed point. The controller actuates the joint to swing the end of the lower pole to a given height from the initial situation where both poles are hanging downwards.

### B. Experiment Setup

We describe the overall experimental setups in this subsection. Our discussion covers (1) all competing algorithms; (2) the stochastic policy model; (3) the representation of value function and policy in the experiments; and (4) configurations of important hyper-parameters for all algorithms, including the learning rate and the discount factor.

*1) Competing Algorithms:* In our experiments, we consider three competing algorithms, including RAC, ARS, and PPO-Linear, because of two main reasons. First, to evaluate the effectiveness of PD-RAC, we need to compare its performance to its counterpart, i.e., RAC. Second, to position PD-RAC in the context of state-of-the-art algorithms, we compare with two cutting-edge algorithms closely related to this paper, i.e., ARS [10] and PPO-Linear [7]. PPO has been reported as the best-performing algorithms on challenging control problems in comparison to many state-of-the-art PGS algorithms [7]. Thus, we have decided to modify the PPO algorithm [9] to support linear policy representation for our experiments. Empirically, we found that the adapted PPO (i.e., PPO-Linear) remains highly effective on most control problems, which satisfies our experimental requirements. To ensure good performance of all competing algorithms, we rely on high-quality algorithm implementations provided by OpenAI Baselines [2] [26].

*2) Value Function and Policy Representations:* Most of PGS algorithms rely heavily on precise learning of value functions. Due to this reason, we decide to model the value function through an NN with proven effectiveness in the literature [3], [9]. The NN architecture for value function will be presented first in this subsection. Following that, we describe the stochastic policy implementation where the policy is represented as a linear parametric function. For fair comparisons, we consistently use the same network architecture for the value function across all competing algorithms except for ARS which does not require a value function. The architecture adopted is depicted in Figure 2.

Following many existing studies [6], [27], we implement the Gaussian policy for all experiments which is parameterized by $\vec{\theta}$ as

$$\pi_{\vec{\theta}}(a|\vec{s}) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(a-\mu)^2}{2\sigma^2}}, \tag{22}$$

where $\mu = \vec{\theta}^T \cdot \vec{\phi}(\vec{s})$ is the mean action output from policy $\pi_{\vec{\theta}}$ in state $\vec{s}$, which can be adjusted by changing policy parameters $\vec{\theta}$. Based on the common setup in the literature [27], the exploration meta-parameter (i.e., the standard deviation) is fixed for all experiments, i.e., $\sigma = 1.0$. Note that, $\pi$ at the RHS of (22) stands for the regular circumference ratio.

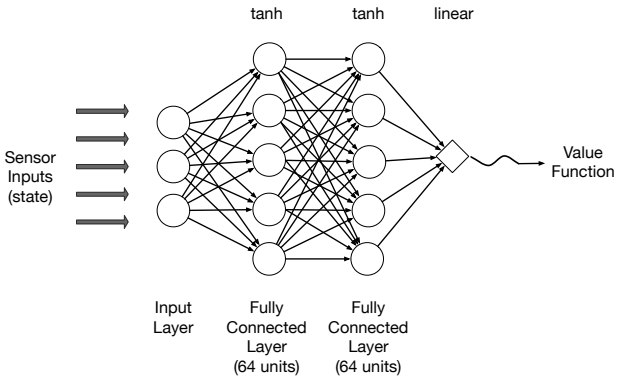


Fig. 2: The Architecture of NN for representing Value Function for PD-RAC, RAC and PPO-Linear.

TABLE I: The Hyper-parameter settings of all algorithms including RAC, PD-RAC, ARS and PPO-Linear used for all benchmark problems.

| Algorithms | Hyper-parameters | | | | | | |
|---|---|---|---|---|---|---|---|
| | $\alpha$ | $\beta$ | $\gamma$ | $\kappa$ | $\rho$ | $K$ | $\sigma$ |
| RAC | 3e-4 | 3e-5 | 0.99 | N/A | N/A | N/A | N/A |
| PD-RAC | 3e-4 | 3e-5 | 0.99 | N/A | 0.95 | 5 | N/A |
| ARS | N/A | 0.025 | 0.99 | N/A | N/A | N/A | 0.1 |
| PPO-Linear | 3e-4 | 3e-4 | 0.99 | N/A | N/A | N/A | N/A |

*3) Hyper-parameter Configurations:* We adopt the hyper-parameter configurations differently for different algorithms, according to the best reported settings for each competing algorithms in the literature [7], [10], [16]. For PD-RAC, we followed the same settings as RAC. All the important configurations can be found in Table I. We refer readers to our algorithm implementation in Github for more detailed configurations.

In Table I, $\alpha$ and $\beta$ are learning rates for training value functions and policies respectively; $\gamma$ is the future reward discount factor; $\kappa$ represents the coefficient for Fisher Information matrix; $\rho$ controls the importance level of historical gradients in (17); $K$ is the periodic interval for PD-RAC; $\sigma$ is the standard deviations for noise in ARS.

### C. Experiment Design

In the experiments, all algorithms are compared in terms of their learning effectiveness. Following the standard setting in the literature [7], [8], [26], effectiveness is defined as the average total rewards of the last 100 episodes.

[2]Our implementations of all algorithms can be found at https://github.com/yimingpeng/primal_dual_baseline

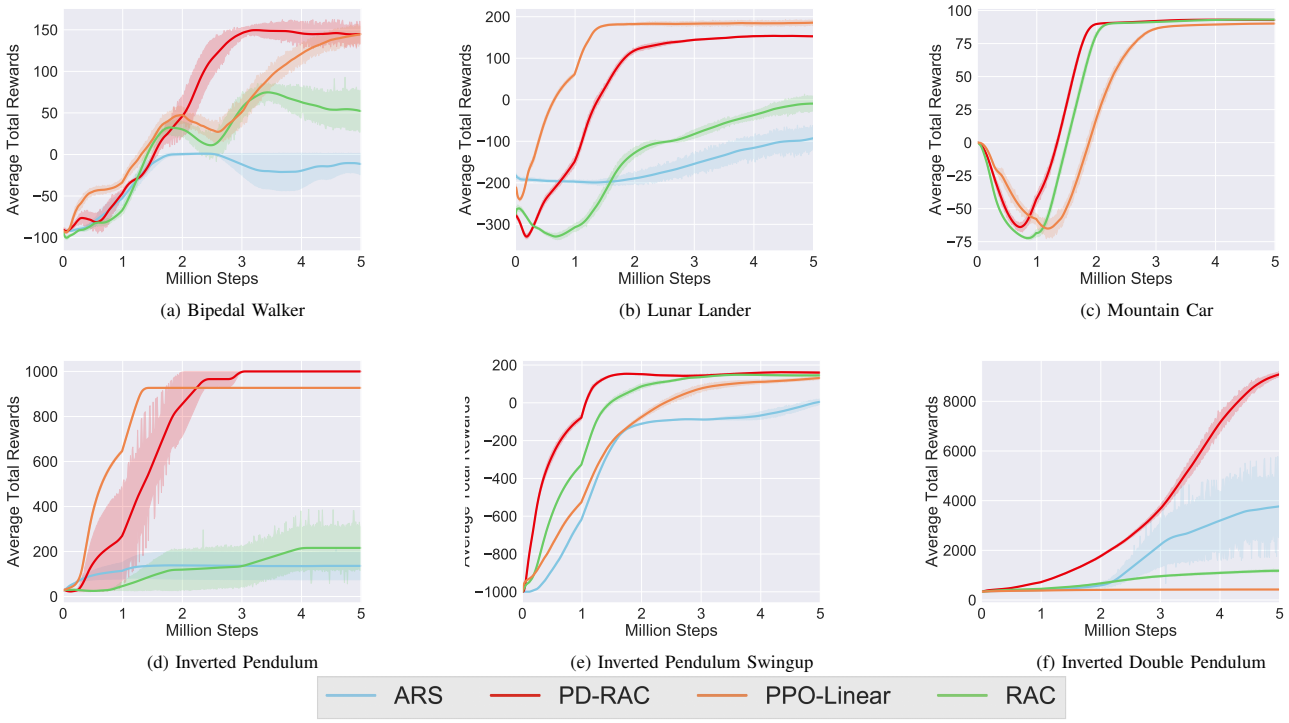


Fig. 3: A performance comparison of PD-RAC against the competing algorithms including RAC, ARS [10] and PPO-Linear [7] on six benchmark problems.

To determine any significant differences in performance, we perform 30 independent runs for all algorithms on each continuous control task. At every 10,000 samples, we conduct one independent test with a deterministic policy. Testing is performed in a separated testing environment with the same random seed as the one used for training. In doing so, we can also identify the true effectiveness as well as sample efficiency of each algorithm. All these independent tests are carried out by using the best policy learned so far till the testing point (i.e., every 10,000 samples). For all experiments and all algorithms, we have performed policy training for only 5,000,000 samples (i.e., 5 million steps), due to the computational resource limitation.

## VI. Results and Discussion

Experimental results for four algorithms on the six benchmark problems will be presented in this section. We firstly present the learning curves of all algorithms with respect to each problem individually in Figure 3. Next, we show the final performance of trained policies after all algorithms completed their training processes.

### A. Discussion on Learning Effectiveness

To evaluate the general effectiveness, we illustrate the learning curves of the proposed PD-RAC in comparison to the competing algorithms in Figure 3. As can be seen from Figure 3, the proposed algorithm PD-RAC performed clearly better than RAC across all six problems. In addition, when compared to the cutting-edge algorithms, PD-RAC outperform ARS on all six problems. In comparison to PPO-Linear, in addition to Inverted Pendulum and Inverted Double Pendulum, PD-RAC also achieved competitive performance on Bipedal Walker and Inverted Pendulum Swingup.

An interesting finding is that ARS did not manage to achieve good performance on all six problems [3]. This is because fitness evaluation in ARS requires a large number of samples. In our case, 5,000,000 learning steps may not be sufficient for the algorithm to converge. In the original paper of ARS [10], the amount of samples required for solving any control task is consistently above 1,000,000. To reduce training time, 100 CPU cores have been utilized in parallel. The hardware requirement is far more than what we can support in our experiments.

To sum up, we can confirm that PD-RAC performed mostly better than the competing algorithms in terms of learning effectiveness.

### B. Discussion on Final Performance

To further analyze the effectiveness of PD-RAC, we present the final performance for all algorithm in Table II. We specifically measured the performance achievable after learning 5M steps in this table, in order to identify any statistically significant difference in performances.

[3]Note that, ARS's learning curve overlaps with X-axis and is hardly distinguished, because it completely fails on the Mountain Car problem as reported in Table II where it obtains 0 total rewards at the final episode.

TABLE II: The final episode performance comparison of four algorithms (i.e., PD-RAC, RAC, ARS and PPO-Linear) on six benchmark problems (i.e., Bipedal Walker, Inverted Double Pendulum, Inverted Pendulum, Inverted Pendulum Swingup, Lunar Lander Continuous, and Mountain Car Continuous).

| Algorithms/Problems | Bipedal Walker | Inverted Double Pendulum | Inverted Pendulum | Inverted Pendulum Swingup | Lunar Lander Continuous | Mountain Car Continuous |
|---|---|---|---|---|---|---|
| ARS | -27.78±51.43 | 4000.07±4017.15 | 135.63±231.02 | 19.30±25.84 | -77.04±83.59 | 0.00±0.00 |
| PD-RAC | 145.13±30.54 | **9271.13±131.03** | **1000.00±0.00** | **156.14±7.54** | 155.77±4.37 | **92.79±0.07** |
| PPO-Linear | **146.32±28.03** | 426.51±6.43 | 931.19±3.56 | 140.48±29.33 | **188.04±20.80** | 90.19±0.08 |
| RAC | 39.76±63.33 | 1184.29±116.89 | 215.41±356.46 | 149.52±14.31 | -16.73±37.79 | 91.10±0.06 |

As can be seen from Table II, PD-RAC has leading final performance across four problems including Mountain Car, Inverted Pendulum, Inverted Pendulum Swingup and Inverted Double Pendulum. Moreover, it shows no significant difference comparing to PPO-Linear on Bipedal Walker problem.

Interestingly, a small performance gap can be spotted between PD-RAC and PPO-Linear on the Lunar Lander problem that is highly sensitive to precise control signals. This implies that the changes to the policy cannot be too large to avoid unexpected behaviors. PPO naturally can keep the policy changes properly bounded by a gradually reducing threshold. However, this ability may prevent PPO-Linear from exploring effectively on other problems such as Mountain Car Continuous and Inverted Double Pendulum.

In summary, based on the analysis on the final learning performance, we can conclude that PD-RAC is sample efficient in comparison to both RAC and other state-of-the-art algorithms, i.e., PPO-Linear and ARS.

## VII. Conclusions

In this paper, we have successfully achieved the main goal of developing an effective linear policy PGS algorithm with step-wise learning. More specifically, we have utilized weighted historical gradients to obtain more accurate policy gradient estimations for effective policy learning. Instead of treating all historical gradients equally, PD-RAC systematically reduces the influence of historical gradients obtained long time in the past in future policy parameter updates. In addition, the experimental results demonstrate the effectiveness of applying PD methods for policy training on several difficult benchmark problems, in comparison to RAC and other competing algorithms. As many RL algorithms with linear policy representations had been proposed in the past decades, a possible future work is to conduct more revisits to those ancient algorithms, which may bring new discoveries.

## References


[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[2] 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," *arXiv*, Sept. 2015.

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

[4] R. Parr, L. Li, G. Taylor, C. Painter-Wakefield, and M. L. Littman, "An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning.," *arXiv*, 2008.

[5] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation.," *NIPS*, 1999.

[6] M. P. Deisenroth, G. Neumann, and J. Peters, "A survey on policy search for robotics," *Foundations and Trends® in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. R. a. p. arXiv, and 2017, "Proximal policy optimization algorithms," *arxiv.org*.

[8] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz, "Trust Region Policy Optimization.," *arXiv*, 2015.

[9] A. Rajeswaran, K. Lowrey, E. V. Todorov, and S. M. Kakade, "Towards Generalization and Simplicity in Continuous Control," pp. 6550–6561, 2017.

[10] H. Mania, A. Guy, and B. Recht, "Simple random search provides a competitive approach to reinforcement learning.," *arXiv*, vol. cs.LG, 2018.

[11] E. Greensmith, P. L. Bartlett, and J. Baxter, "Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning.," *Journal of Machine Learning Research*, 2004.

[12] J. Peters and S. Schaal, "Natural Actor-Critic," *Neurocomputing*, vol. 71, pp. 1180–1190, Mar. 2008.

[13] L. A. Prashanth and M. Ghavamzadeh, "Variance-constrained actor-critic algorithms for discounted and average reward MDPs," *Mach Learn*, vol. 105, pp. 367–417, Aug. 2016.

[14] Y. Nesterov, "Primal-dual subgradient methods for convex problems," *Math. Program.*, vol. 120, pp. 221–259, June 2007.

[15] L. Xiao, "Dual averaging methods for regularized stochastic learning and online optimization," *Journal of Machine Learning Research*, vol. 11, no. Oct, pp. 2543–2596, 2010.

[16] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee, "Natural actor-critic algorithms.," *Automatica*, vol. 45, no. 11, pp. 2471–2482, 2009.

[17] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A Brief Survey of Deep Reinforcement Learning," *IEEE Signal Process. Mag.*, pp. 26–38, Aug. 2017.

[18] N. Le Roux, M. Schmidt, and F. Bach, "A Stochastic Gradient Method with an Exponential Convergence Rate for Finite Training Sets," *arXiv*, Feb. 2012.

[19] V. S. Borkar, "Stochastic approximation with two time scales," *Systems & Control Letters*, vol. 29, pp. 291–294, Feb. 1997.

[20] H. Kushner and G. Yin, *Stochastic Approximation and Recursive Algorithms and Applications*. Springer Science & Business Media, Nov. 2013.

[21] T. Erez, Y. Tassa, and E. Todorov, "Infinite-Horizon Model Predictive Control for Periodic Tasks with Contacts.," *Robotics - Science and Systems*, 2011.

[22] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization.," *IROS*, pp. 4906–4913, 2012.

[23] P. Henderson, W.-D. Chang, F. Shkurti, J. Hansen, D. Meger, and G. Dudek, "Benchmark Environments for Multitask Learning in Continuous Domains.," *arXiv*, vol. cs.AI, 2017.

[24] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv*, June 2016.

[25] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-Real: Learning Agile Locomotion For Quadruped Robots," *arXiv preprint arXiv:1804.10332*, 2018.

[26] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "OpenAI Baselines," tech. rep., 2017.

[27] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural Networks*, vol. 21, pp. 682–697, May 2008.