Extending SHAC with Multiple Variance Reduction Techniques

Virgillio Strozzi
vstrozzi@ethz.ch

Timon Kick
tikick@ethz.ch

Abstract

This paper presents modifications and extensions to the SHAC (Short-Horizon Actor-Critic) policy learning algorithm [18], with the aim of improving its performance in reinforcement learning tasks. By incorporating alpha-order gradient estimators [13] and a TD(λ) formulation [15] in the policy loss, our objective is to improve sample efficiency, reduce the variance of stochastic gradients, and address the vanishing and exploding gradient problem. We implement our modifications and comparisons using the differentiable simulator code from the original SHAC paper [4]. Lastly, we compare our implementation in four different complexity robotic control problems: *Ant*, *Hopper*, *Humanoid*, from the high-performance implementations RL games[3], and *Muscle Tendon Humanoid*, from [10], with the vanilla SHAC algorithm. With the extension of TD(λ), we are able to achieve better results faster and with less noise, whereas with the alpha-order gradient, the results are worse compared to vanilla SHAC. All the implementations are available on the official GitHub repository [16].

1 Introduction

Reinforcement learning algorithms have shown remarkable success in various domains, but challenges such as sample inefficiency and gradient instability persist. In 2022, Xu et al. introduced the SHAC (Short-Horizon Actor-Critic) algorithm [18], which addresses some of these challenges by using parallel differentiable simulation [9, 17] for accelerated policy learning with short optimization windows over the trajectories. We hypothesize that further improvements are possible. This paper proposes modifications and extensions to the SHAC algorithm with a focus on decreasing the variance of stochastic gradients, thereby increasing the overall performance of the algorithm.
In particular, we extend SHAC with the alpha-order gradient estimator $\nabla^{[\alpha]}F(\theta)$ formulation from the work [13]. This approach convexly combines zeroth-order $\nabla^{[0]}F(\theta)$ [14, 12] and first-order $\nabla^{[1]}F(\theta)$ estimates of the gradient of the actor's policy in a differentiable simulator setting. The zeroth-order gradient is more robust, while the first-order gradient typically exhibits less variance, faster convergence to local minima, and greater efficiency [13]. For a continuous function F, these two gradients converge to the same quantity in expectation and depending on the variance of each gradient, their contribution to the convex combination is chosen accordingly.
The second extension incorporates the Temporal Difference learning with eligibility traces (TD(λ)) formulation, which is a value function estimation technique [15], in the policy loss. TD(λ) combines elements of Monte Carlo methods and dynamic programming methods. The goal of this extension is to balance the variance and bias of the policy's stochastic gradient estimates.

From the best of our knowledge, there are no known extensions of the original SHAC algorithm, outside from practical implementation such as in [2] and benchmarks such as in [1].

Foundations of Reinforcement Learning, FS2024. Department of Computer Science, ETH Zürich.

2 Background

In this section we present the SHAC algorithm and the variance reduction techniques used in this work.

2.1 SHAC

Xu et al. [18] model each control problem as a finite-horizon Markov decision process (MDP) with task horizon H. The transition function is given by the differentiable simulation.
SHAC consists of two main components: an actor network that learns a stochastic policy $\pi_\theta(\cdot|s) = \mathcal{N}(\mu_\theta(s), \sigma_\theta(s))$, and a critic network V_ϕ that approximates the state-action value function. To address the challenge of long horizons, SHAC divides the task horizon H into multiple sub-windows of smaller horizons h. During each learning episode, the algorithm concurrently samples N short-horizon trajectories $\{\tau_i\}$ of length h from the simulation. Then, the actor loss

$$L_\theta := -\frac{1}{Nh}\sum_{i=1}^{N}\left[\left(\sum_{t=t_0}^{t_0+h-1}\gamma^{t-t_0}r_t^i\right) + \gamma^h V_\phi(s_{t_0+h}^i)\right] \tag{1}$$

is computed by taking the average over trajectories of the h-step rewards within the sub-window plus a terminal value estimation from the learned critic network. Here, $r_t^i = R(s_t^i, a_t^i)$ is the reward at time t of the i-th trajectory. With this loss and the Adam optimizer, the parameters of the actor network are updated. Importantly, the differentiable simulation enables backpropagation of gradients through the states and actions within the sub-windows, providing an accurate policy gradient. The sampled trajectories are also used to train the critic network V_ϕ using the quadratic loss

$$\mathcal{L}_\phi = \mathop{\mathbb{E}}_{\mathbf{s}\in\{\tau_i\}}\left[\|V_\phi(\mathbf{s}) - \tilde{V}(\mathbf{s})\|^2\right], \tag{2}$$

encouraging the critic network to match the estimated values computed from the trajectories using a TD(λ) formulation

$$\tilde{V}(\mathbf{s}_t) = (1-\lambda)\left(\sum_{k=1}^{h-t-1}\lambda^{k-1}G_t^k\right) + \lambda^{h-t-1}G_t^{h-t}, \tag{3}$$

where $G_t^k = \left(\sum_{l=0}^{k-1}\gamma^l r_{t+l}\right) + \gamma^k V_\phi(\mathbf{s}_{t+k})$ is the k-step return from time t. The estimated value $\tilde{V}(\mathbf{s})$ is treated as constant during critic training, as in other actor-critic RL methods. Further, the target value function technique is used [11], and the target value function $V_{\phi'}$ is used to compute the policy loss 1 and to estimate state values 2. Pseudo-code is provided in Algorithm 1

Algorithm 1 SHAC

1: Initialize policy π_θ, value function V_ϕ, and target value function $V_{\phi'} \leftarrow V_\phi$.
2: **for** learning episode $\leftarrow 1, 2, \ldots, M$ **do**
3: Sample N short-horizon trajectories of length h by the parallel differentiable simulation from the end states of the previous trajectories.
4: Compute the policy loss L_θ defined in 1 from the sampled trajectories and $V_{\phi'}$.
5: Compute the analytical gradient ∇L_θ and update the policy π_θ one step with Adam.
6: Compute estimated values for all the states in sampled trajectories with 3.
7: Fit the value function V_ϕ using the critic loss defined in 2.
8: Update target value function: $V_{\phi'} \leftarrow \alpha V_{\phi'} + (1-\alpha)V_\phi$.
9: **end for**

2.2 Alpha-order gradients

All the information reported in this paragraph comes directly from the paper [13].
The use of a differentiable simulator provides access to the first-order gradient of an objective to update an actor's policy. These simulators can be based on auto-differentiation frameworks [9] or analytic gradient calculation [17]. While this seems a fast and reliable alternative to the robust zeroth-order estimate, some characteristics of certain physical systems, such as stiffness and discontinuities,

may compromise their effectiveness. A natural idea is to analyze this phenomenon under bias and variance, and provide an adaptive convex combination of the zeroth-order and first-order gradients of an objective based on the current gradient landscape. This method is called the α-order gradient estimator, with $\alpha \in [0, 1]$ and combines the efficiency of first-order methods and the robustness of zeroth-order methods.

The setting is a discrete-time, finite-horizon, continuous-state control problem with states $s \in \mathbb{R}^n$, inputs $u \in \mathbb{R}^m$, transition function $\phi : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$, and horizon $H \in \mathbb{N}$. Given a sequence of costs $c_h : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, a family of policies $\pi_h : \mathbb{R}^n \times \mathbb{R}^d \to \mathbb{R}^m$ parameterized by $\boldsymbol{\theta} \in \mathbb{R}^d$, and a sequence of injected noise terms $w_{1:H} \in (\mathbb{R}^m)^H$, the cost-to-go function is defined as follows:

$$V_h(s_h, w_{h:H}, \boldsymbol{\theta}) = \sum_{h'=h}^{H} c_h(s_{h'}, u_{h'}),$$
$$\text{s.t. } s_{h'+1} = \phi(s_{h'}, u_{h'}),\ u_{h'} = \pi(s_{h'}, \boldsymbol{\theta}) + w_{h'},\ h' \geq h.$$

The aim of the actor is to derive a policy minimizing the following objective:

$$F(\boldsymbol{\theta}) := \mathbb{E}_{s_1 \sim \rho} \mathbb{E}_{w_h \overset{\text{iid}}{\sim} p} V_1(s_1, w_{1:H}, \boldsymbol{\theta}), \tag{4}$$

where ρ is a distribution with finite moments over the initial states s_1, and $w_1, \ldots, w_H$ are independent and identically distributed noise terms according to $p = \mathcal{N}(0, \sigma^2 I_n)$ for some $\sigma > 0$. These noise terms act as a *smoothing* regularizer of the optimization landscape [8, 6].

Given zeroth-order estimates of the policy gradient $\hat{\nabla}^{[0]} F_i(\boldsymbol{\theta})$ and first-order gradient estimates $\hat{\nabla}^{[1]} F_i(\boldsymbol{\theta})$, the zeroth-order batched gradient (*ZoBG*) $\bar{\nabla}^{[0]} F(\boldsymbol{\theta})$ and the first-order batched gradient (*FoBG*) $\bar{\nabla}^{[1]} F(\boldsymbol{\theta})$ are respectively defined as the mean of the previous N samples:

$$\hat{\nabla}^{[0]} F_i(\boldsymbol{\theta}) \coloneqq \frac{1}{\sigma^2} \left[V_1(\mathbf{s}_1, \mathbf{w}^i_{1:H}, \boldsymbol{\theta}) - b \right] \left[\sum_{h=1}^{H} \mathrm{D}_{\boldsymbol{\theta}} \pi(\mathbf{s}^i_h, \boldsymbol{\theta})^\intercal \mathbf{w}^i_h \right],$$
$$\hat{\nabla}^{[1]} F_i(\boldsymbol{\theta}) \coloneqq \nabla_{\boldsymbol{\theta}} V_1(\mathbf{s}_1, \mathbf{w}^i_{1:H}, \boldsymbol{\theta}),$$

where $\mathbf{s}^i_h$ is the state at time h of a trajectory induced by the noise $\mathbf{w}^i_{1:H}$, i is the index of the sample trajectory, $\mathrm{D}_{\boldsymbol{\theta}} \pi$ is the Jacobian matrix $\partial \pi / \partial \boldsymbol{\theta} \in \mathbb{R}^{m \times d}$, and $b = V_1(\mathbf{s}_1, \mathbf{0}_{1:H})$ is the zero-noise rollout used as a baseline for variance reduction.

Moreover, for both estimators, the *Empirical variance* is defined as:

$$\hat{\sigma}^2_k := \frac{1}{N-1} \sum_{i=1}^{N} \| \hat{\nabla}^{[k]} F_i(\boldsymbol{\theta}) - \bar{\nabla}^{[k]} F(\boldsymbol{\theta}) \|^2, \quad k \in \{0, 1\}.$$

Lastly, for a given $\alpha \in [0, 1]$, the α-order batched gradient (*AoBG*) is defined as:

$$\bar{\nabla}^{[\alpha]} F(\boldsymbol{\theta}) := \alpha \bar{\nabla}^{[1]} F(\boldsymbol{\theta}) + (1 - \alpha) \bar{\nabla}^{[0]} F(\boldsymbol{\theta}). \tag{5}$$

The value α is then chosen adaptively from the closed-form solution of the variance-minimization problem:

$$\min_{\alpha \in [0,1]} \quad \alpha^2 \hat{\sigma}^2_1 + (1 - \alpha)^2 \hat{\sigma}^2_0$$
$$\text{s.t.} \quad \epsilon + \alpha \underbrace{\| \bar{\nabla}^{[1]} F - \bar{\nabla}^{[0]} F \|}_{B} \leq \gamma.$$

where γ and the confidence ϵ can be chosen adaptively by the user. Namely, for $\gamma = \infty$, the optimal α is $\alpha_\infty := \frac{\hat{\sigma}^2_0}{\hat{\sigma}^2_1 + \hat{\sigma}^2_0}$, while for finite $\gamma \geq \epsilon$,

$$\alpha_\gamma := \begin{cases} \alpha_\infty & \text{if} \quad \alpha_\infty B \leq \gamma - \varepsilon \\ \frac{\gamma - \varepsilon}{B} & \text{otherwise .} \end{cases} \tag{6}$$

Using the adaptive rule at Equation 6 and the formulation of the *AoBG* at Equation 5, it is possible to overcome some of the limitations of using differentiable simulators for planning and control tasks.

2.3 TD(λ)

Temporal Difference learning with eligibility traces, often referred to as TD(λ), is a value function estimation technique [15]. It combines elements of Monte Carlo methods, which use full returns to update value estimates, and dynamic programming methods, which update estimates based on other bootstrap estimates. TD(λ) estimates the value function at a given state s_t by exponentially averaging different k-step returns:

$$\tilde{V}(\mathbf{s}_t) = (1-\lambda)\left(\sum_{k=1}^{T-t-1} \lambda^{k-1} G_t^k\right) + \lambda^{T-t-1} G_t^{T-t},$$

where G_t^k is the k-step return from time t and T is the task horizon. TD(λ) effectively balances the variance and bias of the estimation, as short k-step returns have high bias but low variance, whereas long k-step returns have high variance but low bias.

3 Methods

In this section we detail our extensions to the SHAC algorithm.

3.1 Alpha-order Gradients

In this subsection, we explain two different *AoBG* methods used in our work. Both methods derive an approximation of the gradient ∇L_θ, with the idea to pass this value to the underlying optimizer of the critic. The first method, *Theoretical AoBG*, strictly adapts the definitions of *AoBG* from the paper [13] to vanilla SHAC, while the second method, *Empirical AoBG*, tries to empirically overcome some memory limitations of the previous approach.

Theoretical AoBG

Recall the actor loss defined in 1 and consider the notation in 4. We slightly modify the whole SHAC formulation problem, by introducing gaussian smoothing terms w_i. In particular, the actor loss at time step t_0 of window h, with initial state distribution μ, becomes as follow to match the theoretical premises of the formulation of the *AoBG*:

$$\begin{aligned}
L_\theta &:= -\frac{1}{Nh}\sum_{i=1}^{N}\left[\left(\sum_{t=t_0}^{t_0+h-1}\gamma^{t-t_0}R(s_t^i,a_t^i)\right)+\gamma^h V_\phi(s_{t_0+h}^i)\right] \\
&\approx \mathbb{E}\left[-\frac{1}{h}\left(\sum_{t=t_0}^{t_0+h-1}\gamma^{t-t_0}R(s_t,a_t)\right)+\gamma^h V_\phi(s_{t_0+h})\middle|\tau\sim\text{Gauss. Env}\right] \\
&= \mathbb{E}\left[-\frac{1}{h}\left(\sum_{t=t_0}^{t_0+h-1}\gamma^{t-t_0}R(s_t,a_t)\right)+\gamma^h V_\phi(s_{t_0+h})\middle|s_{t_0}\sim\rho,\right. \\
&\quad \left. s_{t\neq t_0}\sim P(s_{t-1},a_{t-1}+w_{t-1}), a_t\sim\pi_\theta(s_t), w_t\overset{\text{iid}}{\sim}p\right] \\
&= \mathbb{E}_{s_{t_0}\sim\rho}\mathbb{E}_{w_h\overset{\text{iid}}{\sim}p}\mathbb{E}\left[-\frac{1}{h}\left(\sum_{t=t_0}^{t_0+h-1}\gamma^{t-t_0}R(s_t,a_t)\right)+\gamma^h V_\phi(s_{t_0+h})\middle|\right. \\
&\quad \left. s_{t\neq t_0}\sim P(s_{t-1},a_{t-1}+w_{t-1}), a_t\sim\pi_\theta(s_t)\right] \\
&= \mathbb{E}_{s_{t_0}\sim\rho}\mathbb{E}_{w_h\overset{\text{iid}}{\sim}p}V_{t_0}(s_{t_0},w_{t_0:t_0+h},\boldsymbol{\theta}) \\
&= F(\boldsymbol{\theta}),
\end{aligned}$$

where

$$\rho = \begin{cases}\mu, s_{t_0}=s_1 \\ P(s_{t-1},a_{t-1}+w_{t-1}),\ s_{t_0}\neq s_1\end{cases}$$

From this formulations, there's still the need to investigate the theoretical guarantees of the results when $t_0 \neq t_1$, since the theoretical guarantees of *AoBG* hold if the initial distribution ρ has finite

moments. This might not be the case, since for $s_{t_0} \neq s_1$ we have $\rho = P(s_{t_0-1}, a_{t_0-1} + w_{t_0-1})$, which, due to the continuity of the environment, is likely not to have finite moments. The authors of the paper [13] prove their theoretical results for every ρ with finite moments but claim that more general conditions could be established. This means that further theoretical investigation in this direction could be interesting to derive concrete theoretical guarantees in our setting.
Nevertheless, by considering our formulation and applying all the definitions of *Subsection* 2.2, the formulation of *Theoretical AoBG* comes naturally for our setting.

Empirical AoBG

We reformulate the definition of *AoBG* as follows:

$$\bar{\nabla}^{[\alpha]} L_\theta := \alpha \nabla^{[1]} L_\theta + (1-\alpha) \nabla^{[0]} L_\theta.$$

Here, $\nabla^{[1]} L_\theta$ is the first-order gradient as evaluated in normal SHAC, while $\nabla^{[0]} L_\theta$ is a zeroth-order gradient method from the paper [5]. In our case, we choose $\nabla^{[0]} L_\theta$ as the central Gaussian Smoothed Gradient (*cGSG* [5]), where the directions u_i are random directions from a standard Gaussian distribution with standard deviation σ. In the work [5], this method, compared to other zeroth-order methods, performs well in most applications with a sufficient number of evaluations and, more importantly, does not strictly require at least n evaluations of the function. Here, n represents the number of parameters of the Actor neural network. For a fixed input action on L_θ, we have:

$$\nabla^{[0]} L_\theta = \frac{1}{\sigma} \mathbb{E}_{u \sim N(0,I)} \left[(L_{\theta+\sigma u} - L_{\theta-\sigma u}) u \right] \approx \sum_{i=1}^{N} \frac{L_{\theta+\sigma u_i} - L_{\theta-\sigma u_i}}{2N\sigma} u_i$$

Notice that the main difference with the previous approach lies in the evaluation of $\nabla^{[0]} L_\theta$, where the perturbations are applied weights-wise and not actions-wise. This formulation allows us not to evaluate the Jacobian of the policy $\mathrm{D}_{\boldsymbol{\theta}} \pi(\mathbf{s}, \boldsymbol{\theta})$, which results in more efficient sampling and memory-usage. We then choose the α-value as in Equation 6.

3.2 TD(λ)

This extension augments the actor loss L_θ with a TD(λ) formulation. Specifically, we replace

$$-\frac{1}{Nh} \sum_{i=1}^{N} \left[\left(\sum_{t=t_0}^{t_0+h-1} \gamma^{t-t_0} r_t^i \right) + \gamma^h V_\phi(s_{t_0+h}^i) \right] =: L_0$$

with

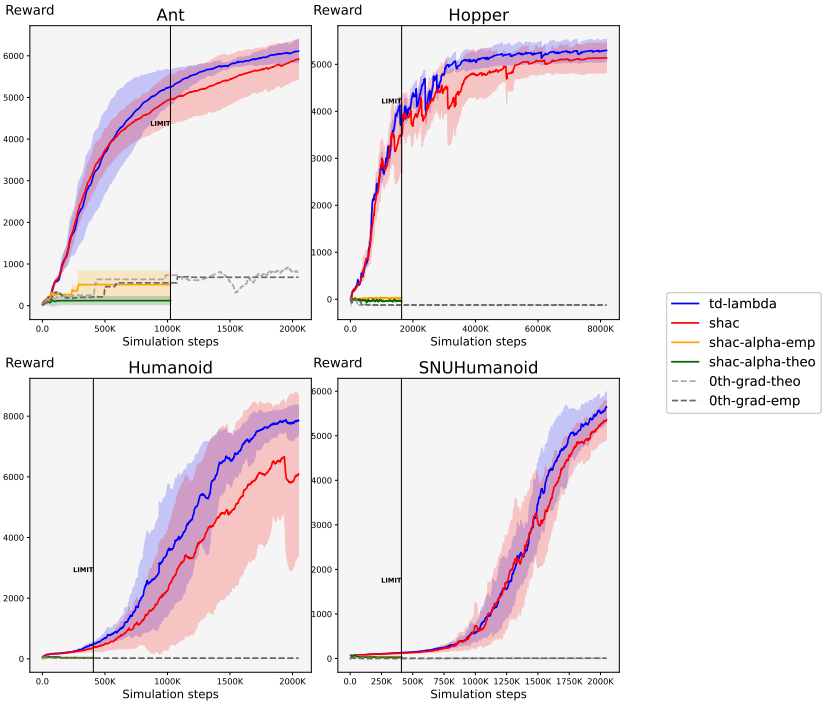
$$-\frac{1}{Nh} \sum_{i=1}^{N} \left[(1-\lambda) \left(\sum_{k=1}^{h-1} \lambda^{k-1} G_k^i \right) + \lambda^{h-1} G_h^i \right] =: L_\lambda,$$

where

$$G_k^i = \left(\sum_{t=t_0}^{t_0+k-1} \gamma^{t-t_0} r_t^i \right) + \gamma^k V_\phi(s_{t_0+k}^i)$$

is the k-step reward from time t_0 of the i-th trajectory. In practice, additional care is required to handle individual trajectories that might terminate early due to agent failure.
The TD(λ) formulation reduces the variance of policy gradient estimates at the cost of increased bias. Furthermore, this extension likely helps mitigate vanishing gradients through the summation and manage exploding gradients via the λ factor.
First experiments, however, showed that L_λ alone yields poor performance. We hypothesize that L_λ might at times be too biased, leading to suboptimal performance. To mitigate this, we take a convex combination $\alpha L_0 + (1-\alpha) L_\lambda$, which has shown better results. The hyperparameter α can be tuned for each task individually. This convex combination approach can be generally adopted in RL tasks to better balance the tradeoff between bias and variance. While fine-tuning λ could achieve similar effects, we kept λ constant to ensure that the observed improvement was due to the modified loss function rather than an optimized λ for value function estimation.
In terms of time complexity, while computing this combined loss is more resource-intensive, the overall increase in runtime is minimal. This is due to the reuse of TD(λ) computation results from the actor loss in critic learning, where TD(λ) is also used. Thus, the computational overhead effectively amounts to a zero-sum addition.

Figure 1: Reward curves comparison of the different implementations. Each curve represents the average of five individual runs for both the TD(λ) and original SHAC implementations, with the shaded area indicating the standard deviation. For both the *Empirical* and *Theoretical AoBG*, the shaded area is shown only for the *Ant* environment, and the iterations are stopped prematurely up to the vertical lime *LIMIT*.



4 Experiments

In this section, we describe the experimental setup of our work, and expose and comment our results. The code can be found at the official GitHub repository [16].

4.1 Experimental Setup

The experiments are all executed and tested on a machine hosting`Debian GNU/Linux 8.10 (jessie)`, and composed of an `Intel(R) Xeon(R) CPU @ 2.20GHz` and a `Tesla P100` with 16GB of RAM provided by Kaggle.
All the implementations kept the original hyperparameters unchanged for a fair comparison, and have custom hyper-parameters tuned individually to their best.

4.2 Benchmark Control Problems

We chose four out of six control tasks from the original SHAC paper to evaluate our modifications: three classical RL tasks, *Ant*, *Hopper*, *Humanoid* from [3], and *Muscle Tendon Humanoid* from [10], a high-dimensional control task with a large action space. The different task complexities ensure comprehensive evaluations.

4.3 Results

4.3.1 Alpha-order Gradients

We comment on the results for both the *Theoretical AoBG* and *Empirical AoBG*. Due to limited resources and the extended runtime of both versions, we were unable to test all implementations across all simulation steps.

For *Theoretical AoBG*, we fixed the noise parameter $\sigma = 10^{-6}$ and set the bound $\gamma - \epsilon = 0.05$ for the choice of the α-value. Observing the global view of the results in Figure 1, it is evident that the results are inferior compared to normal SHAC. Specifically, examining the zeroth-gradient method for *Theoretical AoBG* (*zero-th-grad-theo*), we observe that the actor fails to learn an effective policy outside of the *Ant* environment, resulting in inferior performance compared to other methods. Hence, this performance issue reflects on *AoBG*, which relies on the effectiveness of the zeroth-order method. Nevertheless, some learning does occur in the *Ant* environment, as depicted more clearly in the zoom-in Figure 2. From our experiments, we notice that the α-value selection fluctuates, weighting both first-order and zeroth-order gradients, albeit with a tendency towards the first-order gradient. While the adaptive rule appears effective, the strong fluctuations likely hinder convergence. Even adjusting parameters to favor the first-order gradient does not improve convergence, accuracy, or reward sufficiently compared to vanilla SHAC.
Another significant issue with this implementation relates to the zeroth-order gradient formulation from the paper [13], which includes the policy's Jacobian. Despite theoretical guarantees, this formulation leads to extensive memory usage due to the large size of the Jacobian ($\forall s, \boldsymbol{\theta}. |\mathrm{D}_{\boldsymbol{\theta}} \pi(s, \boldsymbol{\theta})| = |\theta| \cdot |\text{Actions}| > |\theta| = |\nabla_{\boldsymbol{\theta}} V(s, \boldsymbol{\theta})|$) and slower computation times compared to normal SHAC, as each environment step requires Jacobian computation. This increased computational demand is the primary reason we could not conduct longer or more extensive experiments with *Theoretical AoBG*, exhausting computational resources.

For *Empirical AoBG*, we fixed hyperparameters $\sigma = 0.1$, $\gamma - \epsilon = 0.2$, and nr_query $= 5$. Figures 1 and 2 show similar inferior results compared to vanilla SHAC. Figure 2 show that its zeroth-order gradient (*zero-th-grad-emp*) produces an effective policy only for the *Ant* environment and its results are slightly improved in all the environments other than *Ant*, compared to *zero-th-grad-theo*. Overall, *Empirical AoBG* outperforms *Theoretical AoBG* in all tasks in the final iterations, even if the results are not promising. The problem is also highlighted in the work [5], which shows that achieving accurate approximations with zeroth-order methods requires multiple function evaluations with varied noise or directions, scaling with the dimensionality n of the input, corresponding here to the neural network's parameter count. For large networks, this approach becomes costly, limiting the effectiveness of zeroth-order approximation methods. Our method, *cGSG*, does not require at least n evaluations as in the other coordinate-wise zeroth-order methods, which uses a per-parameter direction such as Central Finite Difference (*CFD*) and Forward Finite Difference (*FFD*), but still suffers from too few queries. Moreover, the work at [7] underscores the superiority of coordinate-wise gradient estimation over randomized vector-wise gradient estimation for deep model training in terms of efficiency and results, suggesting future exploration of this approach could be beneficial. The need for more queries to the underlying environment is why we could not conduct longer or more extensive experiments with *Empirical AoBG*.

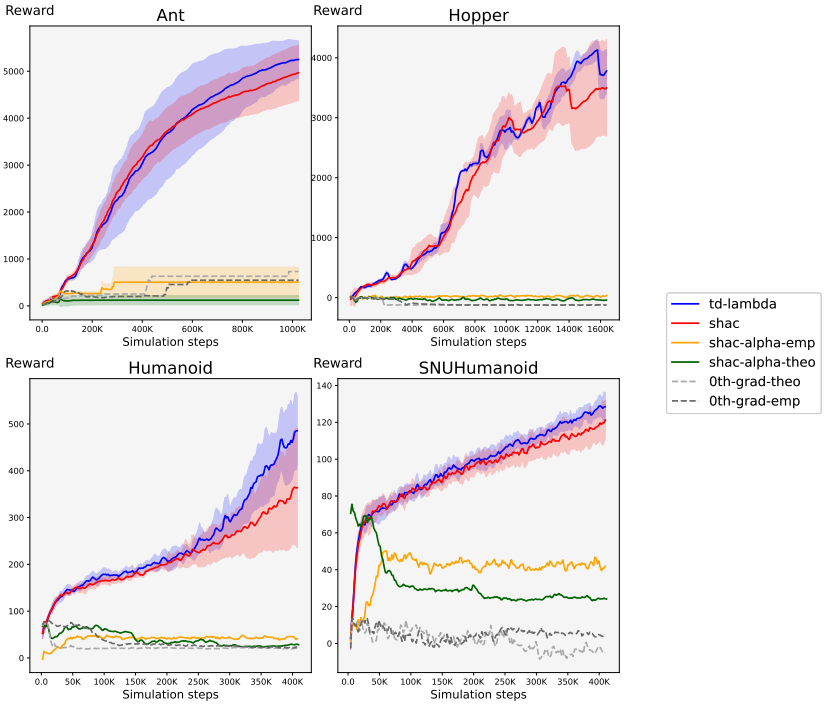
For both zeroth-order methods (*0th-grad-theo*, *0th-grad-emp*), results exhibit higher noise levels (Figure 2) in the *SNUHumanoid* environment, likely due to the environment's larger action-space and complexity. In the *Hopper* and *Humanoid* environments, they frequently become trapped in local minima, producing constant results.

Another noticeable drawback of both *AoBG* methods is the necessity of evaluating the first-order batched gradient (*FoBG*). While evaluating a batch of first-order gradients provides meaningful statistics on optimization landscapes like $\hat{\sigma}_1^2$, aiding in the α-value selection, it necessitates multiple optimizer backward calls, resulting in slower execution compared to evaluating a single first-order batched gradient.

4.3.2 TD(λ)

The results for the TD(λ) extension are presented in Figure 1. For the *Ant* task, the TD(λ) extension demonstrates slightly better performance and substantially reduced reward variance, particularly in the later stages of training. Similar results are achieved in the *Hopper* task. The benefits of the TD(λ) extension are most pronounced in the *Humanoid* task, where it significantly outperforms vanilla SHAC, quickly achieving higher rewards and, most importantly, showing greatly diminished variance. Although the *SNUHumanoid* task does not showcase lower variance, it does have a slightly better mean performance. Overall, the TD(λ) extension seems to provide lower reward variance, which aligns with its aim of reducing the variance of policy gradient estimates, suggesting a more stable

Figure 2: Reward curves comparison of the different implementations. This figure is a zoomed-in version of Figure 1, specifically focusing on the *Empirical* and *Theoretical AoBG* implementations up to the vertical line denoted as *LIMIT*, which marks the point where iterations were stopped.



and reliable learning process. Additionally, the results show a consistent, though small, improvement in performance as indicated by the means.

5 Conclusion and Future Work

So far, we have updated all the dependencies of the entire code simulator at [4], and we have successfully implemented running versions of *AoBG* (*Empirical* and *Theoretical*) and TD(λ), with the latter showing promising results. Despite initial challenges, we believe further exploration into the direction of *AoBG* remains compelling. We were constrained by resources, but we propose several avenues for future extension.

Firstly, as highlighted in the introduction of *Theoretical AoBG*, investigating the theoretical guarantees of *AoBG* in our specific settings and deriving concrete proofs would be valuable. It would verify whether the theory holds and provide more insights on *AoBG*.

Secondly, as mentioned in the Results section, exploring other zeroth-order methods such as CFD and FFD, following insights from [7], could provide deeper insights. It is difficult to assess the effectiveness of *AoBG* if one of its foundational method, the zeroth-order gradient, does not compare well as its first-order counterpart. Therefore, investigating stronger zeroth-order methods would be an initial step. Moreover, comparing the effective time per step and results across different methods could yield better comparisons, as *AoBG* notably requires more time per step. Understanding these trade-offs is crucial to verify whether *AoBG* has a future in the SHAC setting.

Another avenue for exploration involves extending and refining the adaptive rule for selecting α values, perhaps without needing hyperparameter tuning on ϵ and γ. While the current rule has theoretical grounding, it may perform better empirically with adjustments and refinements.

Lastly, it's evident that currently *AoBG* does not integrate well with SHAC. Despite the potential for future research, current experimental drawbacks outweigh the benefits. The performance of zeroth-order methods does not justify their reduced efficiency in our findings.

However, the results from TD(λ) appear promising and contribute meaningfully to the SHAC algorithm. In general, there's a faster convergence to higher reward values, with often very diminished variance and little-to-no additional cost compared to the original SHAC algorithm.

References

[1] DaXBench: Benchmarking Deformable Object Manipulation with Differentiable Physics — arxiv.org. https://arxiv.org/abs/2210.13066, . [Accessed 28-03-2024].

[2] Task-optimal data-driven surrogate models for eNMPC via differentiable simulation and optimization — arxiv.org. https://arxiv.org/abs/2403.14425, . [Accessed 28-03-2024].

[3] GitHub - Denys88/rlgames: RL implementations — github.com. https://github.com/Denys88/rl_games, . [Accessed 28-03-2024].

[4] GitHub - NVlabs/DiffRL: [ICLR 2022] Accelerated Policy Learning with Parallel Differentiable Simulation — github.com. https://github.com/NVlabs/DiffRL, . [Accessed 28-03-2024].

[5] Albert S. Berahas, Liyuan Cao, Krzysztof Choromanski, and Katya Scheinberg. A theoretical and empirical comparison of gradient approximations in derivative-free optimization, 2019.

[6] Albert S. Berahas, Liyuan Cao, Krzysztof Choromanski, and Katya Scheinberg. A theoretical and empirical comparison of gradient approximations in derivative-free optimization. *arXiv: Optimization and Control*, 2019.

[7] Aochuan Chen, Yimeng Zhang, Jinghan Jia, James Diffenderfer, Konstantinos Parasyris, Jiancheng Liu, Yihua Zhang, Zheng Zhang, Bhavya Kailkhura, and Sijia Liu. Deepzero: Scaling up zeroth-order optimization for deep model training. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=qBWhjsNPEY.

[8] John Duchi, Peter Bartlett, and Martin Wainwright. Randomized smoothing for stochastic optimization. *SIAM Journal on Optimization*, 22, 03 2011. doi: 10.1137/110831659.

[9] C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax - a differentiable physics engine for large scale rigid body simulation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

[10] Seunghwan Lee, Moonseok Park, Kyoungmin Lee, and Jehee Lee. Scalable muscle-actuated human simulation and control. *ACM Trans. Graph.*, 38(4), July 2019.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[12] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[13] H. J. Terry Suh, Max Simchowitz, Kaiqing Zhang, and Russ Tedrake. Do differentiable simulators give better policy gradients? 2022.

[14] Richard Sutton, David Mcallester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Adv. Neural Inf. Process. Syst*, 12, 02 2000.

[15] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[16] Virgilio Strozzi Timon Kick. Extending shac with multiple variance reduction techniques. https://github.com/vstrozzi/FRL-SHAC-Extension, 2024.

[17] Keenon Werling, Dalton Omens, Jeongseok Lee, Ioannis Exarchos, and C. Karen Liu. Fast and Feature-Complete Differentiable Physics Engine for Articulated Rigid Bodies with Contact Constraints. In *Proceedings of Robotics: Science and Systems*, Virtual, July 2021. doi: 10.15607/RSS.2021.XVII.034.

[18] Jie Xu, Viktor Makoviychuk, Yashraj Narang, Fabio Ramos, Wojciech Matusik, Animesh Garg, and Miles Macklin. Accelerated policy learning with parallel differentiable simulation. In *International Conference on Learning Representations*, 2021.