Policy Gradient Methods with Adaptive Policy Spaces

Gianmarco Tedeschi
Politecnico di Milano
gianmarco.tedeschi@polimi.it

Matteo Papini
Politecnico di Milano
matteo.papini@polimi.it

Marcello Restelli
Politecnico di Milano
marcello.restelli@polimi.it

Abstract

Policy search is one of the most effective reinforcement learning classes of methods for solving continuous control tasks. These methodologies attempt to find a good policy for an agent by fixing a family of parametric policies and then searching directly for the parameters that optimize the long-term reward. However, this parametric policy space represents just a subset of all possible Markovian policies, and finding a good parametrization for a given task is a challenging problem in its own right, typically left to human expertise. In this paper, we propose a novel, model-free, *adaptive-space* policy search algorithm, GAPS. We start from a simple policy space; then, based on the observations we receive from the unknown environment, we build a sequence of policy spaces of increasing complexity, yielding more sophisticated optimized policies at each epoch. The final result is a parametric policy whose structure (including the number of parameters) is fitted on the problem at hand without any prior knowledge of the task. Finally, our algorithm is tested on a selection of continuous control tasks, evaluating the resulting policy sequence and comparing the results with traditional policy optimization methods that use a fixed policy space.

1 Introduction

Artificial intelligence, at its core, addresses the challenge of enabling an agent to make optimal decisions to accomplish specific tasks. Reinforcement Learning (RL) [25] tackles this by having the agent learn the best behavior through direct interaction with the environment and evaluating the performance by accumulating a reward signal. Among RL approaches, *policy gradient* (PG) methods have proven particularly effective in real-world control scenarios. PG methods have demonstrated success in continuous control tasks [22, 23, 24], robotics [26], and partially observable environments [16]. These algorithms work by directly searching within a space of parametric policies to find the one that maximizes a performance index.

However, determining an appropriate parametric policy space remains a significant challenge. A policy space that is too small may converge quickly but will likely result in poor performance due to its limited ability to represent the optimal behavior for the problem. On the other hand, a very large policy space might devise more effective policies, but it comes with high computational and sample costs.

Usually, this decision is based on domain-dependent practices or is left to human expertise, leveraging existing knowledge about the nature of the task to solve. However, in the real world, it is quite uncommon to have access to the information needed to suitably design the policy space *before* the learning process starts. Indeed, such a decision would require an apriori knowledge of the environment in which the agent operates, contrasting with the usual setting in which RL has proved beneficial.

38th Workshop on Aligning Reinforcement Learning Experimentalists and Theorists (ARLET 2024).

With the advent of Deep RL (DRL) [13], new methods have emerged that use complex neural networks to define parametric policies. These policies are often characterized by a very large number of parameters. Despite their capacity and flexibility, using neural networks to model policy spaces incurs a high computational cost and demands massive data to reduce the variance that affects policy gradient estimators. This makes DRL impractical for some applications, particularly those with limited resources or real-time constraints. Thus, the research question: *Can we shape the policy space during learning in an effective way?* remains open.

In this paper, we follow a different approach to identify a parametric space of appropriate complexity for the task at hand. We formulate the search for the most appropriate parametric space as a sequence of curricula by *starting small* [6] and gradually devising more complex policy spaces. The philosophy of starting small is widely embraced in several fields. Curriculum learning (CL) [28], is based on the concept of learning by iteratively moving from simpler instances to more complex ones. The original concept of CL [2] proposed "training from easier to harder data", i.e., training the machine learning model with subsets of easy examples from the dataset, then gradually increasing the difficulty until the whole training dataset is considered. Some works tried to adapt the concept of *curriculum* to RL, considering a sequence of control tasks of increasing complexity [e.g., 14]. On a very general level, curriculum can be thought of as an ordering of experience samples. The latter can be used to produce methods that start in a goal state and iteratively expand the start state distribution assuming reversible dynamics [12] or access to an approximate dynamics model [7]. Other approaches generate the curriculum from demonstration states [21] or through online exploration [5]. However, a curriculum can also be represented at a task level, organizing a set of tasks into a sequence [15] or in a graph of tasks which also specifies the order of learning [10].

An alternative approach, adopted in this paper, is to apply the curriculum concept at the *agent* level rather than at the task level, training a sequence of policies of increasing complexity. An existing example of agent curricula is [4], based on training and mixing a sequence of *given* policy networks of increasing complexity. The main limitation of this approach is that the sequence must be carefully designed in advance. In this work, we introduce the concept of *expanding policy spaces*. Embracing the philosophy of starting small, we begin with a simple policy parametrization and gradually expand the policy space, *automatically* adapting to the task's difficulty while learning the optimal parameters.

This is achieved by deploying copies of the same simple policy, with independent parameters, in each element of a state space partition, composing a piece-wise policy whose complexity depends only on the complexity of the partition. Starting from the whole state space, we partition it into finer and finer regions every time the optimization of the current parametrization reaches convergence. Each time, the partition is refined to maximize the potential performance improvement, and the optimization of the augmented piece-wise policy starts again. We can prove that each policy expansion leads to a potential performance improvement, even if the previous piece-wise policy has already been optimized to convergence. Eventually, the algorithm can find an appropriate policy space and a set of parameters without prior knowledge of the environment, with the advantage of focusing on simpler, faster-to-optimize policies at the beginning of the learning process.

Contribution. In this paper, we introduce a novel, model-free, actor-only policy optimization algorithm named Gradient-based Adaptive Policy Search (`GAPS`) that combines PG with the agent-curriculum idea to efficiently find (and optimize) the optimal policy parametrization for a given task. The contributions of this paper encompass theoretical, algorithmic, and experimental aspects. After providing some background on policy gradient in Section 2, we present the key insights motivating our algorithm in Section 3, and describe the details of `GAPS` in Section 4. In Section 5, we provide a one-step policy update analysis to show the improvement guarantees unlocked by policy space expansion. Finally, in section 6, we empirically assess the performance of our method on popular continuous RL tasks. The proofs of our theoretical results are reported in Appendix A.

2 Preliminaries

Markov Decision Processes. A reinforcement learning task [25] can be modeled as a discrete-time Markov Decision Process (MPD), which is defined as a tuple $\mathcal{M} := (\mathcal{S}, \mathcal{A}, P, R, \gamma, \mu_0)$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P(\cdot|s,a)$, is a Markovian transition model which defines for each state-action pair (s,a) the probability to reach the next state s', $R(s,a) \in [-R_{\max}, R_{\max}]$ assigns the expected reward for performing action a in state s, $\gamma \in [0,1]$ is a discount factor

and μ_0 is the distribution of the initial state. In this framework, the behavior of an agent is described by a policy $\pi(\cdot|s)$ that assigns to each state s the probability of performing a certain action a. We consider episodic MDPs with effective horizon $H \in \mathbb{N}$. A trajectory $\tau \in \mathcal{T}$ is a a sequence of states and actions $\tau = (s_{\tau,0}, a_{\tau,0}, ..., s_{\tau,H-1}, a_{\tau,H-1}, s_{\tau,H})$ and $\mathcal{T}$ is the set of collected trajectories. We evaluate the performance of an agent in terms of the expected return, i.e., the expected discounted sum of the rewards collected along τ: $J(\pi) = \mathbb{E}_{\tau\sim p(\cdot|\pi)}[\mathcal{R}(\tau)]$, with $\mathcal{R}(\tau) = \sum_{t=0}^{H-1} \gamma^t r(s_t, a_t)$ and the expectation is computed over the trajectory distribution, whose density function is $p(\tau|\pi) := \mu_0(s_0) \prod_{t=0}^{H-1} \pi(a_t|s_t) P(s_{t+1}|s_t, a_t)$. The objective of an RL task is to find a policy $\pi^\star \in \arg\max_\pi \{J(\pi)\}$. In our work, we focus on policies that belong to a parametric policy space $\Pi_\Theta = \{\pi_\theta : \theta \in \Theta \subseteq \mathbb{R}^d\}$, Θ being the parameter space. In such a case, we abbreviate $J(\theta) \coloneqq J(\pi_\theta)$ and $p(\cdot|\theta) \coloneqq p(\cdot|\pi_\theta)$ for any $\theta \in \Theta$.

Policy Gradient Methods. Policy gradients are a class of methods for searching the best-performing policy over a class of parametrized policies Π_Θ. We will denote with $J(\boldsymbol{\theta})$ the performance of a parametric policy and with $p(\tau|\boldsymbol{\theta})$ the probability of a trajectory τ. The search of a locally optimal policy is performed through gradient ascent, where the policy gradient is [29]:

$$\nabla_\theta J(\boldsymbol{\theta}) = \mathbb{E}_{\tau\sim p(\cdot|\boldsymbol{\theta})} \left[\nabla_\theta \log_{\boldsymbol{\theta}}(\tau|\boldsymbol{\theta}) \mathcal{R}(\tau)\right]. \tag{1}$$

At each iteration $p > 0$, a dataset of trajectories $\mathcal{T}_N^p = \{\tau_i\}_{i=1}^N$, with $N \in \mathbb{N}$, is collected using policy π_θ. The policy parameter is iteratively updated as $\theta_{k+1} = \theta_k + \eta \widehat{\nabla}_{\theta_k} J(\theta_k)$, where $\eta > 0$ is the step size and $\widehat{\nabla}_{\theta_k} J(\theta_k)$ is an estimate of the policy gradient (Eq. (1)) using $\mathcal{T}_N^p$. The most common policy gradient estimator (e.g., REINFORCE [29] and GPOMDP [1]) can be expressed as $\widehat{\nabla}_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^N g(\tau_i|\theta)$ for $\tau_i \in \mathcal{T}_N^p$, where $g(\tau_i|\theta)$ is $\nabla \log p(\tau_i) \mathcal{R}(\tau_i)$. In this work, we refer to g as the GPOMDP estimator, which is preferred due to its smaller variance and is given by:

$$\widehat{\nabla}_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{H-1} \gamma^t r(s_{i,t}, a_{i,t}) \sum_{l=0}^{t} \nabla_\theta \log \pi_\theta(a_{i,l}|s_{i,l}), \qquad \tau_i \in \mathcal{T}_N^p. \tag{2}$$

3 Policy Space Expansion

In this section, we introduce key concepts of policy space expansion, the building blocks of GAPS. The high-level idea involves enlarging the policy space by progressively partitioning the environment's state space into finer regions and deploying a distinct *simple* policy π_θ within each region. We refer to this procedure as *policy space expansion* and the resulting overall policy as *piece-wise policy*.

State Space Partition and Piece-Wise Policies. We define a *region* as a subset of the state space $\mathcal{E} \subseteq \mathcal{S}$. We consider a *partition* $\boldsymbol{\mathcal{E}} = \{\mathcal{E}_i\}_{i=1}^M$ of the state space, i.e., a set of M regions $\mathcal{E}_i$, such that $\mathcal{S} = \cup_{i=1}^M \mathcal{E}_i$ and $\mathcal{E}_i \cap \mathcal{E}_j = \emptyset$ for every $i \neq j$. We denoted the set of indexes of the regions of the partition $\boldsymbol{\mathcal{E}}$ as $[M] = \{1, ..., M\}$. Let $\boldsymbol{\mathcal{E}}, \boldsymbol{\mathcal{E}}'$ be two partitions of $\mathcal{S}$, containing M and M' regions, respectively. We say that $\boldsymbol{\mathcal{E}}'$ is a *subpartition* of $\boldsymbol{\mathcal{E}}$, and denote it with $\boldsymbol{\mathcal{E}}' \sqsubseteq \boldsymbol{\mathcal{E}}$, if there exists a partition $\boldsymbol{\mathcal{I}} = \{\mathcal{I}_i\}_{i=1}^M$ of $[M']$ such that for every $i \in [M]$ we have $\cup_{\mathcal{E}'_j : j\in\mathcal{I}_i} \mathcal{E}'_j = \mathcal{E}_i$, i.e., we can reconstruct a bigger region $\mathcal{E}_i$ as the union of $\{\mathcal{E}'_j\}_{j\in\mathcal{I}_i}$. We will call the latter a *split* of $\mathcal{E}_i$. We consider a *base policy space* $\Pi_\Theta = \{\pi_\theta, \theta \in \Theta\}$, where Θ should be thought of (a subset of) a low-dimensional Euclidean space.[1] Given a partition $\boldsymbol{\mathcal{E}} = \{\mathcal{E}_i\}_{i=1}^M$, we construct the piece-wise policy space by associating to every region $\mathcal{E}_i$ a policy parameter $\theta_i \in \Theta$. Thus, the complete policy parametrization is given by $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_M)^\top$. This way, whenever the environment is in a state $s \in \mathcal{E}_i$, the piece-wise policy executes the simple policy π_{θ_i}, leading to the following definition.

Definition 1 (Piece-Wise Policy)**.** *Given a fixed policy space Π_Θ, a partition $\boldsymbol{\mathcal{E}} = \{\mathcal{E}_i\}_{i=1}^M$ of $\mathcal{S}$, and a policy parametrization $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_M)^\top \in \Theta^M$. The piece-wise policy $\pi_{\boldsymbol{\theta}}$ is defined as:*

$$\forall s \in \mathcal{S}: \qquad \pi_{\boldsymbol{\theta}}(\cdot|s) = \sum_{i=1}^{M} \mathbf{1}\{s \in \mathcal{E}_i\} \pi_{\theta_i}(\cdot|s).$$

[1] In all our examples, either $\Theta \subseteq \mathbb{R}$ or $\Theta \subseteq \mathbb{R}^{d_\mathcal{A}}$, where $d_\mathcal{A}$ is the action-space dimension.

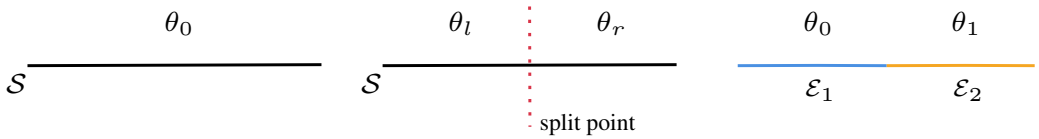


Figure 1: State space splitting and reallocation of parameters in a one-dimensional state space.

For policy $\pi_{\boldsymbol{\theta}}$, we straightforwardly extend the notations $J(\boldsymbol{\theta})$ for the expected return and $p(\cdot|\boldsymbol{\theta})$ for the trajectory density function. This construction of the piece-wise policy allows us to obtain a particularly convenient expression of the policy gradient. Indeed, for every $i \in [M]$, we have:

$$\nabla_{\theta_i} J(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim p(\cdot|\boldsymbol{\theta})} \left[\sum_{t=0}^{H-1} \gamma^t r(s_t, a_t) \sum_{l=0}^{t} \mathbf{1}\{s_l \in \mathcal{E}_i\} \nabla_{\theta_i} \log \pi_{\theta_i}(a_l|s_l) \right]. \tag{3}$$

In words, the terms in the sum of the score functions are filtered to retain only the scores where the policy π_{θ_i} is played, i.e., when $s_{n,l} \in \mathcal{E}_i$.

Policy Space Expansion. The idea behind GAPS consists in *splitting* the current state partition $\boldsymbol{\mathcal{E}}$ (made of M regions) to create a subpartion $\boldsymbol{\mathcal{E}}' \sqsubseteq \boldsymbol{\mathcal{E}}$ (made of $M' > M$ regions), whenever policy $\pi_{\boldsymbol{\theta}}$, with $\boldsymbol{\theta} \in \Theta^M$, approaches a stationary point of the performance measure for the policy space defined over partition $\boldsymbol{\mathcal{E}}$, i.e., whenever $\nabla J(\boldsymbol{\theta}) \approx 0$.[2] This amounts to deploying a new piece-wise policy $\pi_{\boldsymbol{\theta}'}$ where $\boldsymbol{\theta}' \in \Theta^{M'}$, replicating the parameters θ_i of $\boldsymbol{\theta}$ for the regions $\mathcal{E}_i$ that are split into $\{\mathcal{E}'_j\}_{j \in \mathcal{I}_i}$. Formally, for every $i \in [M]$, we have that $\theta'_j = \theta_i$ for every $j \in \mathcal{I}_i$. When this holds, we introduce the equivalence relation $\boldsymbol{\theta} \equiv_{\boldsymbol{\mathcal{I}}} \boldsymbol{\theta}'$. Consequently, even though the resulting policy remains the same (just with a different parametrization), the expanded policy space has the potential to achieve higher performance, being defined over a finer partition of the state space. Replicating the parameter values offers the convenient opportunity to *estimate the policy gradient of $\pi_{\boldsymbol{\theta}'}$ (post-splitting) using samples collected with $\pi_{\boldsymbol{\theta}}$ (pre-splitting) only*.

Lemma 3.1. *Let $\boldsymbol{\mathcal{I}} = \{I_i\}_{i=1}^M$ be a partition of $[M']$ with $M' > M$, $\boldsymbol{\theta} \in \Theta^M$, and $\boldsymbol{\theta}' \in \Theta^{M'}$. If $\boldsymbol{\theta} \equiv_{\boldsymbol{\mathcal{I}}} \boldsymbol{\theta}'$, for every $i \in [M]$ and $j \in \mathcal{I}_i$, it holds that:*

$$\nabla_{\theta'_j} J(\boldsymbol{\theta}') = \mathbb{E}_{\tau \sim p(\cdot|\boldsymbol{\theta})} \left[\sum_{t=0}^{H-1} \gamma^t r(s_t, a_t) \sum_{l=0}^{t} \mathbf{1}\{s_l \in \mathcal{E}'_j\} \nabla_{\theta_i} \log \pi_{\theta_i}(a_l|s_l) \right]. \tag{4}$$

Furthermore, it holds that:

$$\nabla_{\theta_i} J(\boldsymbol{\theta}) = \sum_{j \in \mathcal{I}_i} \nabla_{\theta'_j} J(\boldsymbol{\theta}'). \tag{5}$$

Thus, we can compute the gradient right after the splitting, $\nabla_{\theta'_j} J(\boldsymbol{\theta}')$, using samples collected just before the splitting with policy $\pi_{\boldsymbol{\theta}}$. This is crucial as it allows for the evaluation of multiple candidate splits (i.e., different subpartitions) *without the need to collect new samples*, facilitating the selection of the one that offers the most promising learning opportunities. Indeed, as can be seen from Equation (5), even when the pre-splitting gradient is zero, $\nabla_{\theta_i} J(\boldsymbol{\theta}) = \mathbf{0}$, the individual post-splitting terms $\nabla_{\theta'_j} J(\boldsymbol{\theta}')$ may be non-zero, potentially unlocking new learning opportunities. A graphical description of the splitting operation is shown in Figure 1.

4 Gradient-based Adaptive Policy Search

In this section, we discuss a novel model-free actor-only policy search algorithm named GAPS, explaining the structure and criteria to determine the optimal policy expansion (Algorithm 1).

Notation. We denote the parameter associated with the newly built region on the left side of the split point s with the subscript l and the superscript s, while its counterpart on the right side is denoted with the subscript r. The split points considered valid are stored in a set noted as $\mathcal{D}_v$. Eventually, with the notation $\theta \leftarrow (\theta_l^{s^\star}, \theta_r^{s^\star})$ and $\mathcal{E} \leftarrow (\mathcal{E}_l^{s^\star}, \mathcal{E}_r^{s^\star})$ we denote the policy and region expansion.

[2] This way, we may stop at a suboptimal *local* optimum. This is an inherent limitation of policy gradient methods, which attempt to solve a non-convex optimization problem via gradient descent. However, in many cases, locally optimal policies are actually (close to) global optima [3].

Splitting Criterion. A key point for the algorithm is to determine *when* and *where* to split the state space to maximize the potential performance improvement resulting from policy expansion. Before seeking a splitting point, we optimize until convergence of the current policy parameters $\boldsymbol{\theta}$. Ideally, in this scenario, $\nabla J(\boldsymbol{\theta}) = \mathbf{0}$, and the objective of splitting is to find a new parametrization $\boldsymbol{\theta}'$ with the greatest potential for improvement. To achieve this, we propose focusing on the single most promising region of the environment and then assessing the magnitude and direction of the resulting post-splitting gradients for each potential split point. This approach comprises four phases.

Phase 1 — Region selection. Firstly, we identify a region in our current state space partition that offers the greatest potential for improvement. To gauge this potential, we consider the variance of the gradient estimate components for the current parametrization $\boldsymbol{\theta}$. A higher variance for a component indicates greater variability in the gradient for that specific parameter, suggesting the presence of distinctively different "left" and "right" gradients for some split points. Note that each parameter θ_i is directly associated with a region $\mathcal{E}_i$. The selection of the region is determined by:

$$i^* \in \arg\max_{i \in [M]} \mathbb{V}\mathrm{ar}[\widehat{\nabla}_{\theta_i} J(\boldsymbol{\theta})].$$

Phase 2 — Generation of split point candidates. In our work, we focus on state spaces $\mathcal{S} \subseteq \mathbb{R}^d$, thus each region is also $\mathcal{E}_i \subseteq \mathbb{R}^d$. We restrict the partition of the region $\mathcal{E}_{i^*}$ to a split along a single orthogonal coordinate $\xi \in [d]$. The coordinate ξ for each epoch is selected in a round-robin fashion across the state space dimensions. We now generate a set of *candidate splitting points*, which, with a little abuse of notation, we denote as $\mathcal{S}_{\text{split}} \in \mathcal{T} \cap \mathcal{E}_{i^*}$, i.e., the states appearing in the observed trajectories that belong to region $\mathcal{E}_{i^*}$. Since this may be a very large set and evaluating candidate split points is computationally expensive, we sample a subset of the visited states $\mathcal{S}_{\text{split}}$. To respect the (discounted) visitation probabilities, we sample a fixed number of states from each trajectory with replacement, where the probability of sampling the t-th state in the trajectory is $\gamma^t(1-\gamma)$.

Phase 3 — Selection of valid split points. After identifying the region to be split $\mathcal{E}_{i^*}$ and the set of candidate splitting points $\mathcal{S}_{\text{split}}$, we search for split points $s \in \mathcal{S}_{\text{split}}$ that induce two subregions with post-splitting gradients with *diverging directions*. We test the null hypothesis that the angle between the two gradient vectors is in $[0, \pi/2]$ against the alternative hypothesis that it is in $[\pi/2, \pi]$, indicating that the gradients point in contrasting directions. When the null hypothesis is rejected, it means that no direction allows for the simultaneous improvement in the two subregions, and the two corresponding sets of parameters will move to significantly different values, resulting in a more expressive policy. To perform this test, we convert the set of gradient pairs into a set of random angles [11], representing the angle between each pair of gradients.

We first test for uniformity using the Rayleigh test [11]. Rejecting the null hypothesis means that there is a preferred direction—the measured angles concentrate around some significant value. Next, we assume that the angular data follow a Von Mises distribution [11]. Based on this assumption, we construct a confidence interval at level α for the mean of the Von Mises distribution. A detailed description of how to compute the confidence interval can be found in Appendix B. If the confidence interval is entirely contained in $(\pi/2, \pi)$, the two gradients point in opposite directions, so we can conclude that the observed split has the potential to yield a performance improvement. We refer to such a point as *valid* split point s and we denote the corresponding set as $\mathcal{S}_{\text{valid}} \subseteq \mathcal{S}_{\text{split}}$.

Phase 4 — Split point selection. The fourth phase consists of maximizing the expected performance improvement derived from the division. After collecting the valid split points $\mathcal{S}_{\text{valid}}$ in the previous phase, we look for the optimal split point as

$$s^\star = \arg\max_{s \in \mathcal{S}_{\text{valid}}} \left\| \widehat{\nabla}_\theta J(\theta_l^s) \right\|_2 + \left\| \widehat{\nabla}_\theta J(\theta_r^s) \right\|_2 .$$

Using the coordinate ξ of $s^\star$, we generate a new pair of regions and their associated parameters.

Algorithm Structure. The algorithm is structured as a double loop, consisting of policy space expansion (Algorithm 2) and policy parameter optimization (Algorithm 1). The outer loop is repeated for a fixed number of epochs P, during which we optimize the current policy $\boldsymbol{\theta}$ until convergence. Using the latest estimate of the gradient $\widehat{\nabla}_{\theta_k} J(\boldsymbol{\theta})$ from policy optimization, we measure the variance of the gradient components to select the region to split. Once a region $\mathcal{E}$ has been selected, the inner loop searches for a valid split point. Using the batch of trajectories $\mathcal{T} \cap \mathcal{E}$ generated from experience with the policy $\pi_{\boldsymbol{\theta}}$ and a split point s, the algorithm computes the corresponding gradients. These gradients are evaluated based on their directional difference. If at least one valid split point is found

Algorithm 1 GAPS (Gradient-based Adaptive Policy Search)

Inputs: Max number of epochs P, initial policy space Θ_1, initial parameter $\boldsymbol{\theta}$

for $p = 1, 2, ..., P$ **do**
  Optimize $\pi_{\boldsymbol{\theta}}$ with GPOMDP until convergence, storing trajectories in $\mathcal{T}^p$
  Sort regions of state-space partition by descending order of $\mathbb{V}\mathrm{ar}[\widehat{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta})]$
  $k = 1$
  repeat
    $\Theta_{d+1} =$ EXPAND_POLICY_SPACE $(\Theta_d, \mathcal{E}_k, \theta_k, \mathcal{T})$ {Algorithm 2}
    $k \leftarrow k + 1$
  until $\Theta_{d+1} \neq \Theta_d$ **or** $k = d$
  if $\Theta_{d+1} = \Theta_d$ **then**
    return $\pi_{\boldsymbol{\theta}}$
  end if
end for

Algorithm 2 EXPAND_POLICY_SPACE

Inputs: Policy space Θ, region $\mathcal{E}$ to split and corresponding parameter θ, batch of trajectories $\mathcal{T}$
Outputs: new policy space Θ'

for $a = \texttt{shuffle}(0, 1, 2, ..., d_{\mathcal{S}})$ **do**
  Initialize $\mathcal{D}_v = \emptyset$ as the set of valid points
  for s in $\mathcal{T} \cap \mathcal{E}$ **do**
    Split space in s along axis ξ to obtain new regions $\mathcal{E}_l^s, \mathcal{E}_r^s$
    Initialize new parameters $\theta_l^s \leftarrow \theta$ and $\theta_r^s \leftarrow \theta$
    Compute $\widehat{\nabla}_\theta J(\theta_l^s)$ and $\widehat{\nabla}_\theta J(\theta_r^s)$ with GPOMDP
    if $\widehat{\nabla}_\theta J(\theta_l^s)$ and $\widehat{\nabla}_\theta J(\theta_r^s)$ are significantly different directions **then**
      $\mathcal{D}_v = \mathcal{D}_v \cup \{s\}$
    end if
  end for
  if $\mathcal{D}_v \neq \emptyset$ **then**
    $s^\star = \arg\max_{s \in \mathcal{D}_v} \left\| \widehat{\nabla}_\theta J(\theta_l^s) \right\| + \left\| \widehat{\nabla}_\theta J(\theta_r^s) \right\|_2$
    Update parameters $\theta_l^{s^*} \leftarrow \theta_l^{s^*} + \eta \widehat{\nabla}_\theta J(\theta_l^{s^*})$ and $\theta_r^{s^*} \leftarrow \theta_r^{s^*} + \eta \widehat{\nabla}_\theta J(\theta_r^{s^*})$
    Expand Θ into Θ' by splitting:
    $\theta \leftarrow (\theta_l^{s^\star}, \theta_r^{s^\star})$
    $\mathcal{E} \leftarrow (\mathcal{E}_l^{s^\star}, \mathcal{E}_r^{s^\star})$
    return Θ'
  end if
end for
return Θ

within the region, we then construct two new sub-regions and compute their associated parameters, thereby expanding the policy parametrization to $\boldsymbol{\theta}'$ as described in Section 3.

5 Guaranteed Policy Improvement after Policy Expansion

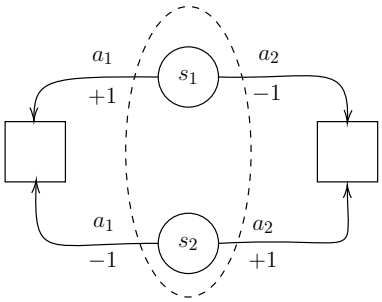
In this section, we outline the policy improvement guarantees that result from expanding the policy space. We present a one-step performance improvement analysis, starting from a scenario where the current policy $\boldsymbol{\theta}$ has converged, resulting in a gradient $\nabla J(\boldsymbol{\theta}) = \mathbf{0}$, indicating no further room for improvement. From this initial condition, we perform a state space split, leading to an associated expansion of the policy $\boldsymbol{\theta}$ and a subsequent policy update $\boldsymbol{\theta'}$.

Lemma 5.1. *Let $\boldsymbol{\theta}_{old} \in \Theta$ be the policy parametrization before the splitting operation, $\boldsymbol{\theta} \in \Theta$ be the expanded but not yet updated policy, and $\boldsymbol{\theta}' \in \Theta$ the expanded policy parametrization after a single update. Then, the improvement given by the new policy can be expressed as:*

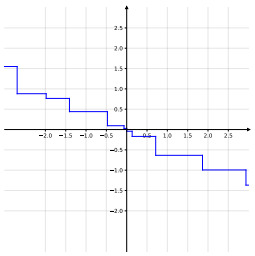
$$J(\boldsymbol{\theta}') - J(\boldsymbol{\theta}_{old}) = J(\boldsymbol{\theta}') - J(\boldsymbol{\theta}) = (\boldsymbol{\theta}' - \boldsymbol{\theta})^T \nabla J(\boldsymbol{\theta}) + \frac{1}{2}(\boldsymbol{\theta}' - \boldsymbol{\theta})^T \nabla^2 J(\widetilde{\boldsymbol{\theta}})(\boldsymbol{\theta}' - \boldsymbol{\theta}),$$

where $\widetilde{\boldsymbol{\theta}} = \lambda\boldsymbol{\theta} + (1 - \lambda)\boldsymbol{\theta}'$ for some $\lambda \in [0, 1]$ and $\nabla^2 J(\cdot)$ denotes the Hessian matrix.

This result follows directly from the mean value theorem and the observation that $J(\boldsymbol{\theta}_{\mathrm{old}}) = J(\boldsymbol{\theta})$, as the policy space expansion does not alter the policy itself, but only its parametrization.



(a) Differentiating the states can reveal a non-zero policy gradient in a two-state MDP.



(b) Piece-wise constant policy learned in 1-dimensional LQR.

The following assumptions on the smoothness of the performance function and the bounded variance of policy gradient estimators are standard in the study of sample complexity [30] and monotonic improvement [17]. We will justify both assumptions for our choice of policies in the following.

Assumption 1 (Smoothness)**.** *For any value of $\boldsymbol{\theta}$ there exists a constant $L < \infty$ such that: $\left\|\nabla^2 J(\boldsymbol{\theta})\right\|_2 \leq L$, where $\|\cdot\|_2$ denotes the spectral norm.*

Assumption 2 (Bounded variance)**.** *For any policy $\pi_{\boldsymbol{\theta}}$ there exists a constant $V < \infty$ such that: $\mathbb{V}\mathrm{ar}[g(\cdot|\boldsymbol{\theta})] \leq V$.*

Thus, under Assumption 2, the policy gradient estimated from a batch of N trajectories has a variance bounded by V/N. We can now state the policy improvement guarantees for GAPS.

Theorem 5.2. *Assume the REINFORCE or GPOMDP gradient estimator is used in GAPS (see Equation 2). Under Assumption 1 and 2, the expanded parameter vector $\boldsymbol{\theta}'$ computed by Algorithm 2 has, for an ideal step size η defined as:*

$$\eta = \frac{\|\nabla J(\boldsymbol{\theta})\|^2}{L\left(\|\nabla J(\boldsymbol{\theta})\|^2 + \frac{V}{N}\right)},$$

the following inequality holds:

$$J(\boldsymbol{\theta}') - J(\boldsymbol{\theta}_{\mathrm{old}}) \geq \frac{\|\nabla J(\boldsymbol{\theta})\|^2}{2L\left(1 + \frac{V}{N\|\nabla J(\boldsymbol{\theta})\|^2}\right)}.$$

This shows that if the gradient is non-zero, as it is *after* any valid expansion performed by our algorithm, a non-zero performance improvement is guaranteed by a sufficiently small update to the expanded policy's parameters. Refer to Appendix A for a detailed proof of Theorem 5.2.

Example. We now provide a minimal example illustrating how expanding the policy space by splitting the state space can transform a zero gradient into a nonzero one. A more concrete example can be found in Appendix D. Consider the two-state, two-action deterministic MDP from Figure 2a (squares denote terminal states), with a uniform starting state distribution and $\gamma = 1$. Let $\mathcal{S} = \{s_1, s_2\}$ and $\mathcal{A} = \{a_1, a_2\}$. We start with a policy space that does not differentiate between the two states, specifically using a softmax policy of the following form:

$$\pi_\theta(a_1) = \frac{\exp(\theta)}{1 + \exp(\theta)}, \qquad \pi_\theta(a_2) = 1 - \pi_\theta(a_1). \tag{6}$$

Using Equation 1, we verify that $\nabla J(\theta) = 0$ regardless of the value of θ. Now, imagine the state space is split into two regions $\mathcal{E}_1 = \{s_1\}$ and $\mathcal{E}_2 = \{s_2\}$. The resulting piecewise softmax policy is:

$$\pi_{\boldsymbol{\theta}}(a_1|s) = \frac{\exp(\theta_1)}{1 + \exp(\theta_1)}\mathbf{1}\{s = s_1\} + \frac{\exp(\theta_2)}{1 + \exp(\theta_2)}\mathbf{1}\{s = s_2\}, \qquad \pi_{\boldsymbol{\theta}}(a_2|s) = 1 - \pi_{\boldsymbol{\theta}}(a_1|s). \tag{7}$$

It is easy to check that the gradient now becomes:

$$\nabla J(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\exp(\theta_1)}{(1+\exp(\theta_1))^2} & -\frac{\exp(\theta_2)}{(1+\exp(\theta_2))^2} \end{bmatrix}^\top, \tag{8}$$

which is nonzero even if $\theta_1 = \theta_2 = \theta$, which is the case right after a split. For instance, $\theta = 0$, which is (locally) optimal before the split, yields $\nabla J(\boldsymbol{\theta}) = [1/4, -1/4]$ after the split. Additionally, note that the two derivatives have opposite signs, qualifying this as a valid split according to our method.

Piecewise-constant Gaussian policies. We specialize our theoretical analysis to Gaussian policies with piecewise *constant* mean, the class used in our numerical simulations.

Definition 2. *Given the definition of piecewise policies in Definition 1, we can write piecewise-constant Gaussian policy as a linear Gaussian policy with state features $\phi : \mathcal{S} \to \mathbb{R}^M$ representing a one-hot encoding of the M regions:*

$$\pi_{\boldsymbol{\theta}}(\cdot|s) = \mathcal{N}(\boldsymbol{\theta}^T \phi(s); \sigma^2), \qquad \phi_i(s) \coloneqq \mathbf{1}\{s \in \mathcal{E}_i\}, \qquad i \in [M],$$

where $\sigma > 0$ is the (constant) standard deviation of the Gaussian policy, and $\theta_i \in \mathbb{R}$ is the constant expected action for the region $\mathcal{E}_i$.[3]

From Definition 2, we apply the bound on the smoothness constant L for linear Gaussian policies [30]:

$$\left\|\nabla^2 J(\boldsymbol{\theta})\right\|_2 \leq \frac{2\mathcal{R}_{\max} \sup_{s\in\mathcal{S}} \|\phi(s)\|^2}{\sigma^2(1-\gamma)^2} = \frac{2\mathcal{R}_{\max}}{\sigma^2(1-\gamma)^2}. \tag{9}$$

Similarly, we can use an upper bound on the variance of the GPOMDP estimator for Gaussian linear policies [17, Lemma 29]:

$$\mathbb{V}\mathrm{ar}\left[\widehat{\nabla} J(\boldsymbol{\theta})\right] \leq \frac{\sup_{s\in\mathcal{S}} \|\phi(s)\|^2 \mathcal{R}_{\max}^2 (1-\gamma^H)}{\sigma^2 N(1-\gamma)^3} = \frac{\mathcal{R}_{\max}^2 (1-\gamma^H)}{\sigma^2 N(1-\gamma)^3}. \tag{10}$$

Note that neither the smoothness constant (Eq. 9) nor the variance upper bound (Eq. 10) depend on the number of parameters M of the piece-wise policy. This implies that the curvature of the objective function and the variance of the gradient estimates remain stable across epochs. In other words, expanding the policy space does not disrupt the ongoing optimization process.

6 Numerical Simulations

In this section, we evaluate the performance of GAPS and compare it with policy gradient on well-known continuous control tasks: *Linear Quadratic Regulator* (LQR) ([9]) in both its scalar and multi-dimensional formulation, *Swimmer*, *Half-Cheetah* and *Ant* from the MuJoCo ([27]) and the *Minigolf* ([18]) environment. We consider GPOMDP as the estimator due to its smaller variance, also employing baselines ([19]) to reduce gradient variance further. Details on the implementation of both GPOMDP and GAPS can be found in Appendix C. Additional experimental results and further details on the experimental setting are provided in Appendix D.

We run both GPOMDP and GAPS for a variable number P iterations, with a batch size N, an an epoch length H tailored to the specific task (a detailed description of parameters used for training is provided in Appendix C). We consider both an adaptive step size η computed using Adam ([8]) algorithm and a constant learning rate chosen based on the observed problem. For the policy gradient algorithm, we adopted a Gaussian policy linear in the state, while for GAPS we use Piecewise-constant Gaussian policies defined in Section 5. We keep a fixed $\sigma^2 = 0.1$ for both algorithm policies.

From Figure 2, we observe that for a linear problem like LQR, the policy learned from GAPS converges faster, reaching the same performance as a linear policy. Consistent with the explanation in Section 5, we can observe that each policy space expansion (highlighted with a red dotted line) leads to performance improvements. For the one-dimensional LQR case, we also visually represent the policy in Figure 2b (which eventually resembles the optimal linear solution). The same considerations are also valid in the multidimensional LQR (2)

We also observe how, in settings where a linear policy is not suitable, such as the Minigolf environment, the policy produced by GAPS with a small number of splits manages to outperform GPOMDP (Figure 3). Additionally, the Swimmer experiment in Figure 3 provides an interesting observation on the policy complexity. We can appreciate the trade-off between GPOMDP and GAPS w.r.t. the parameter dimensionality $d_{\boldsymbol{\theta}}$. In Figure 3, we note that GAPS with just one iteration of policy expansion not only finds a policy that yields positive performance but also surpasses GPOMDP with a linear policy

[3] For $d_{\mathcal{A}}$-dimensional actions, each θ_i corresponds to a vector of $d_{\mathcal{A}}$ action variables, and the total dimension of $\phi(s)$ is $Md_{\mathcal{A}}$.

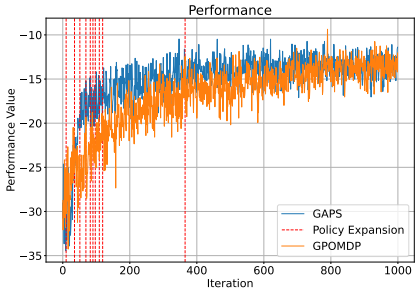
Table 1: Policy space dimension and mean performance for each environment

(a) Results for GPOMDP

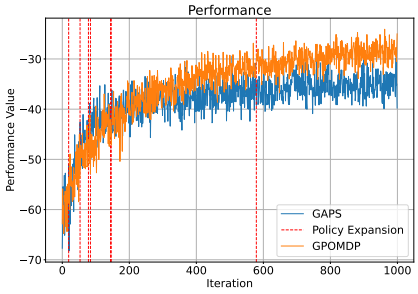
Environment	d_Θ	$J_{\boldsymbol{\theta}}$	95% C.I.
LQR 1-dimensional	1	-16.8	[-16.97, -16.74]
LQR 2-dimensional	4	-35.1	[-35.35, -34.68]
Minigolf	1	-73.16	[-74.73, -71.59]
Swimmer	16	24.2	[23.59, 24.80]
Half-Cheetah	102	176.58	[153.06, 200.09]
Ant	216	331.5	[315.42, 347.57]

(b) Results for GAPS

Environment	d_Θ	$J_{\boldsymbol{\theta}}$	95% C.I.
LQR 1-dimensional	10	-15.3	[-15.76, -15.07]
LQR 2-dimensional	14	-41.0	[-44.22, -37.81]
Minigolf	6	-6.2	[-7.45, -4.98]
Swimmer	4	27.26	[26.63, 27.88]
Half-Cheetah	12	98.46	[97.86, 99.05]
Ant	16	134.58	[116.07, 152.68]



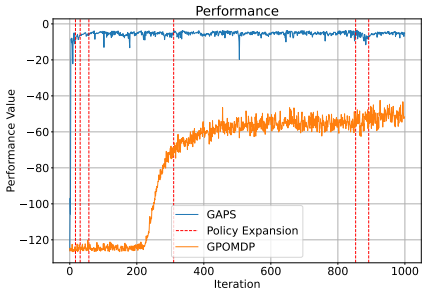
(a) 1-dimensional LQR



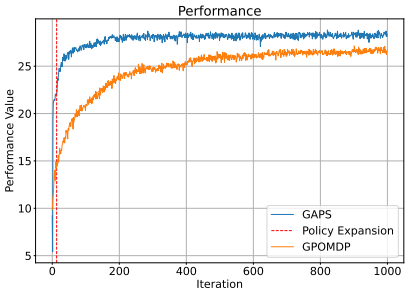
(b) 2-dimensional LQR

Figure 2: Learning curve in 1d and 2d LQR

that exploits a higher number of parameters. The exact number of parameters adopted for each environment and their relative performance are shown in Table 1.



(a) Minigolf environment



(b) Swimmer environment

Figure 3: Learning curves of Minigolf and Swimmer environments

7 Conclusions

In this paper, we introduced GAPS, a novel model-free policy search algorithm. The algorithm is based on the concept of *policy space expansion*, following a *starting small* philosophy. By adaptively learning from a simple policy, we construct an efficient policy space and derive an optimal policy without prior knowledge of the environment or task. Theoretical analysis demonstrates how expanding the policy space can improve performance, and extensive numerical simulations validate our claims. Our work paves the way for several promising research directions. Firstly, our implementation could be coupled with the design of an adaptive step size mechanism capable of handling an expanding parameter space. Furthermore, our approach is versatile enough to accommodate other types of policy, thereby extending its applicability beyond piecewise constant policies. Lastly, a parameter-based approach that autonomously adapts the parameter space represents another research direction.

References

[1] Jonathan Baxter and Peter L Bartlett. Infinite-horizon policy-gradient estimation. *journal of artificial intelligence research*, 15:319–350, 2001.

[2] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[3] Jalaj Bhandari and Daniel Russo. Global optimality guarantees for policy gradient methods. *Operations Research*, 2024.

[4] Wojciech Marian Czarnecki, Siddhant M. Jayakumar, Max Jaderberg, Leonard Hasenclever, Yee Whye Teh, Simon Osindero, Nicolas Heess, and Razvan Pascanu. Mixmatch - agent curricula for reinforcement learning, 2018.

[5] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. First return, then explore. *Nature*, 590(7847):580–586, February 2021. arXiv:2004.12919 [cs].

[6] Jeffrey L. Elman. Learning and development in neural networks: the importance of starting small. *Cognition*, 48(1):71–99, July 1993.

[7] Boris Ivanovic, James Harrison, Apoorva Sharma, Mo Chen, and Marco Pavone. BaRC: Backward Reachability Curriculum for Robotic Reinforcement Learning, September 2018. arXiv:1806.06161 [cs].

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

[9] Vladimír Kucera. Optimal control: Linear quadratic methods: Brian d. o. anderson and john b. moore. *Autom.*, 28(5):1068–1069, 1992.

[10] Patrick MacAlpine and Peter Stone. Overlapping layered learning. *Artificial Intelligence*, 254:21–43, January 2018.

[11] Kantilal V. Mardia and Peter E. Jupp. *Directional statistics*. Wiley series in probability and statistics. Wiley, Chichester, new ed. edition, 2000.

[12] Stephen McAleer, Forest Agostinelli, Alexander Shmakov, and Pierre Baldi. Solving the Rubik's Cube Without Human Knowledge, May 2018. arXiv:1805.07470 [cs].

[13] 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.

[14] Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey, September 2020. arXiv:2003.04960 [cs, stat].

[15] Sanmit Narvekar, Jivko Sinapov, and Peter Stone. Autonomous Task Sequencing for Customized Curriculum Design in Reinforcement Learning. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, pages 2536–2542, Melbourne, Australia, August 2017. International Joint Conferences on Artificial Intelligence Organization.

[16] Andrew Y Ng and Michael I Jordan. Pegasus: A policy search method for large mdps and pomdps. *arXiv preprint arXiv:1301.3878*, 2013.

[17] Matteo Papini, Matteo Pirotta, and Marcello Restelli. Smoothing policies and safe policy gradients. *Machine Learning*, 111(11):4081–4137, November 2022.

[18] AR Penner. The physics of putting. *Canadian Journal of Physics*, 80(2):83–96, 2002.

[19] Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

[20] Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697, May 2008.

[21] Cinjon Resnick, Roberta Raileanu, Sanyam Kapoor, Alexander Peysakhovich, Kyunghyun Cho, and Joan Bruna. Backplay: "man muss immer umkehren", 2022.

[22] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

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

[24] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[25] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[26] Russ Tedrake, Teresa Weirui Zhang, and H Sebastian Seung. Stochastic policy gradient reinforcement learning on a simple 3d biped. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2849–2854. IEEE, 2004.

[27] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[28] Xin Wang, Yudong Chen, and Wenwu Zhu. A survey on curriculum learning, 2021.

[29] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

[30] Rui Yuan, Robert M. Gower, and Alessandro Lazaric. A general sample complexity analysis of vanilla policy gradient, November 2022. arXiv:2107.11433 [cs, math, stat].

A Proofs

Lemma 3.1. *Let $\boldsymbol{\mathcal{I}} = \{I_i\}_{i=1}^{M}$ be a partition of $[M']$ with $M' > M$, $\boldsymbol{\theta} \in \Theta^M$, and $\boldsymbol{\theta'} \in \Theta^{M'}$. If $\boldsymbol{\theta} \equiv_{\boldsymbol{\mathcal{I}}} \boldsymbol{\theta'}$, for every $i \in [M]$ and $j \in \mathcal{I}_i$, it holds that:*

$$\nabla_{\theta'_j} J(\boldsymbol{\theta'}) = \mathbb{E}_{\tau \sim p(\cdot|\boldsymbol{\theta})} \left[\sum_{t=0}^{H-1} \gamma^t r(s_t, a_t) \sum_{l=0}^{t} \mathbf{1}\{s_l \in \mathcal{E}'_j\} \nabla_{\theta_i} \log \pi_{\theta_i}(a_l|s_l) \right]. \tag{4}$$

Furthermore, it holds that:

$$\nabla_{\theta_i} J(\boldsymbol{\theta}) = \sum_{j \in \mathcal{I}_i} \nabla_{\theta'_j} J(\boldsymbol{\theta'}). \tag{5}$$

Proof. First of all, since $\boldsymbol{\theta} \equiv_{\boldsymbol{\mathcal{I}}} \boldsymbol{\theta'}$, we have that $\theta'_j = \theta_i$ and, consequently:

$$\nabla_{\theta'_j} \log \pi_{\theta'_j}(a|s) = \nabla_\theta \log \pi_\theta(a|s)|_{\theta=\theta'_j} = \nabla_\theta \log \pi_\theta(a|s)|_{\theta=\theta_i} = \nabla_{\theta_i} \log \pi_{\theta_i}(a|s). \tag{11}$$

The first statement follows from the policy gradient expression (Eq. 1) recalling that $p(\cdot|\boldsymbol{\theta'}) = p(\cdot|\boldsymbol{\theta})$. For the second statement, we simply observe that $\sum_{j \in \mathcal{I}_i} \mathbf{1}\{s \in \mathcal{E}'_j\} = \mathbf{1}\{s \in \mathcal{E}_i\}$. □

Theorem 5.2. *Assume the REINFORCE or GPOMDP gradient estimator is used in GAPS (see Equation 2). Under Assumption 1 and 2, the expanded parameter vector $\boldsymbol{\theta'}$ computed by Algorithm 2 has, for an ideal step size η defined as:*

$$\eta = \frac{\|\nabla J(\boldsymbol{\theta})\|^2}{L\left(\|\nabla J(\boldsymbol{\theta})\|^2 + \frac{V}{N}\right)},$$

the following inequality holds:

$$J(\boldsymbol{\theta'}) - J(\boldsymbol{\theta}_{\text{old}}) \geq \frac{\|\nabla J(\boldsymbol{\theta})\|^2}{2L\left(1 + \frac{V}{N\|\nabla J(\boldsymbol{\theta})\|^2}\right)}.$$

Proof. To better display our reasoning, we first present a proof for the simpler but unrealistic case in which the *exact* gradient is used in the policy parameter update:

$$\boldsymbol{\theta'} = \boldsymbol{\theta} + \eta(\boldsymbol{\theta}),$$

then we move to the stochastic gradient case:

$$\boldsymbol{\theta'} = \boldsymbol{\theta} + \eta \widehat{\nabla} J(\boldsymbol{\theta}),$$

where $\widehat{\nabla} J(\boldsymbol{\theta})$ is estimated from a batch of N trajectories, proving our main result.

Exact gradient. Starting from the result of Lemma 5.1,

$$J(\boldsymbol{\theta'}) - J(\boldsymbol{\theta_{old}}) = J(\boldsymbol{\theta'}) - J(\boldsymbol{\theta}) = (\boldsymbol{\theta'} - \boldsymbol{\theta})^\top \nabla J(\boldsymbol{\theta}) + \frac{1}{2}(\boldsymbol{\theta'} - \boldsymbol{\theta})^\top \nabla^2 J(\tilde{\boldsymbol{\theta}})(\boldsymbol{\theta'} - \boldsymbol{\theta}),$$

we can apply the Cauchy–Schwarz inequality and Assumption 1:

$$J(\boldsymbol{\theta'}) - J(\boldsymbol{\theta}) \geq (\boldsymbol{\theta'} - \boldsymbol{\theta})^\top \nabla J(\boldsymbol{\theta}) - \frac{L}{2} \|\boldsymbol{\theta'} - \boldsymbol{\theta}\|^2 \tag{12}$$

$$= \eta \|\nabla J(\boldsymbol{\theta})\|^2 - \eta^2 \frac{L}{2} \|\nabla J(\boldsymbol{\theta})\|^2 \tag{13}$$

$$= \|\nabla J(\boldsymbol{\theta})\|^2 \left(\eta - \frac{L}{2}\eta^2\right) \tag{14}$$

$$= \frac{1}{2L} \|\nabla J(\boldsymbol{\theta})\|^2 \geq \frac{C^2}{2L}. \tag{15}$$

where the η that maximizes (14) was computed as

$$\frac{\partial}{\partial \eta}[\|\nabla J(\boldsymbol{\theta})\|^2 (\eta - \frac{L}{2}\eta^2)] = \|\nabla J(\boldsymbol{\theta})\|^2 - 2\eta \frac{L}{2} \|\nabla J(\boldsymbol{\theta})\|^2 = 0, \tag{16}$$

$$\implies \eta = \frac{1}{L}. \tag{17}$$

Stochastic gradient. Starting from the result of Lemma 5.1, under Assumption 1 and 2, applying the Cauchy–Schwarz inequality on Lemma 5.1:

$$J(\boldsymbol{\theta'}) - J(\boldsymbol{\theta}) \geq (\boldsymbol{\theta'} - \boldsymbol{\theta})^\top \nabla J(\boldsymbol{\theta}) - \frac{L}{2} \left\| \boldsymbol{\theta'} - \boldsymbol{\theta} \right\|^2 \tag{18}$$

$$= \eta \widehat{\nabla} J(\boldsymbol{\theta})^\top J(\boldsymbol{\theta}) - \nabla J(\boldsymbol{\theta}) - \eta^2 \frac{L}{2} \left\| \widehat{\nabla} J(\boldsymbol{\theta}) \right\|^2. \tag{19}$$

We apply the expectation (conditional on $\boldsymbol{\theta}$) on both sides:

$$\mathbb{E}[J(\boldsymbol{\theta'}) - J(\boldsymbol{\theta})] \geq \eta \mathbb{E}[\widehat{\nabla} J(\boldsymbol{\theta})]^\top \nabla J(\boldsymbol{\theta}) - \eta^2 \frac{L}{2} \mathbb{E}\left[\left\| \widehat{\nabla} J(\boldsymbol{\theta}) \right\|^2 \right] \tag{20}$$

$$= \eta \left\| \nabla J(\boldsymbol{\theta}) \right\|^2 - \eta^2 \frac{L}{2} \left\| \nabla J(\boldsymbol{\theta}) \right\|^2 - \eta^2 \frac{L}{2} \mathbb{V}\mathrm{ar}[\widehat{\nabla} J(\boldsymbol{\theta})], \tag{21}$$

since the gradient estimate is unbiased, and from the definition of $\mathbb{V}\mathrm{ar}[\widehat{\nabla} J(\boldsymbol{\theta})]$:

$$\mathbb{V}\mathrm{ar}[\widehat{\nabla} J(\boldsymbol{\theta})] = \mathrm{Tr}(\mathbb{C}\mathrm{ov}[\widehat{\nabla} J(\boldsymbol{\theta})]) \tag{22}$$

$$= \mathbb{E}\left[\left\| \widehat{\nabla} J(\boldsymbol{\theta}) - \nabla J(\boldsymbol{\theta}) \right\|^2 \right] \tag{23}$$

$$= \mathbb{E}\left[\left\| \widehat{\nabla} J(\boldsymbol{\theta}) \right\|^2 \right] - \left\| \nabla J(\boldsymbol{\theta}) \right\|^2. \tag{24}$$

Then, applying Assumption 2 to Equation (21):

$$\mathbb{E}[J(\boldsymbol{\theta'}) - J(\boldsymbol{\theta})] \geq \eta \left\| \nabla J(\boldsymbol{\theta}) \right\|^2 - \eta^2 \frac{L}{2} \left(\left\| \nabla J(\boldsymbol{\theta}) \right\|^2 + \frac{V}{N} \right) \tag{25}$$

$$= \frac{\left\| \nabla J(\boldsymbol{\theta}) \right\|^4}{2L \left(\left\| \nabla J(\boldsymbol{\theta}) \right\|^2 + \frac{V}{N} \right)} \tag{26}$$

$$= \frac{\left\| \nabla J(\boldsymbol{\theta}) \right\|^2}{2L \left(1 + \frac{V}{\left\| \nabla J(\boldsymbol{\theta}) \right\|^2 N} \right)} \tag{27}$$

$$\geq \frac{C^2}{2L \left(1 + \frac{V}{NC^2} \right)}, \tag{28}$$

where the η that maximizes (25) is computed as

$$\frac{\partial}{\partial \eta} \left(\eta \left\| \nabla J(\boldsymbol{\theta}) \right\|^2 - \eta^2 \frac{L}{2} \left(\left\| \nabla J(\boldsymbol{\theta}) \right\|^2 + \frac{V}{N} \right) \right) \tag{29}$$

$$= \left\| \nabla J(\boldsymbol{\theta}) \right\|^2 - \eta L \left(\left\| \nabla J(\boldsymbol{\theta}) \right\|^2 + \frac{V}{N} \right) = 0 \tag{30}$$

$$\implies \eta = \frac{\left\| \nabla J(\boldsymbol{\theta}) \right\|^2}{L \left(\left\| \nabla J(\boldsymbol{\theta}) \right\|^2 + \frac{V}{N} \right)}. \tag{31}$$

□

B Statistics on Angles

In this section we report the tools from directional statistics [11] that we used to detect valid splitting points in our algorithm. We refer the reader to Chapter 7 of [11] for further details.

Construction of angle dataset. Given a batch of n trajectories, we can compute n estimates of the left gradient, say $\{g_1^l, \ldots, g_n^l\}$, and n estimates of the right gradient, $\{g_1^r, \ldots, g_n^r\}$. We denote with $\angle(v, w)$ the smallest angle between vectors v and w, measured (in the plane spanned by the two vectors) between 0 and π radians. We construct a dataset of n angles $\{\vartheta_1, \ldots, \vartheta_n\}$ where $\vartheta_i = \angle(g_i^l, g_i^r)$ for $i = 1, \ldots, n$. We will work with this dataset of angles in the following.

Preliminary definitions. From our dataset of angles we can compute the following statistics:

$$R = \sqrt{C^2 + S^2}, \quad C = \sum_{j=1}^{n} \cos(\vartheta_j), \quad S = \sum_{j=1}^{n} \sin(\vartheta_j), \tag{32}$$

and their normalized versions:

$$\bar{R} = \frac{R}{n}, \quad \bar{C} = \frac{C}{n}, \quad \bar{S} = \frac{S}{n}. \tag{33}$$

Uniformity test. First we need to make sure that our angles concentrate around some value. We perform a uniformity test known as Rayleigh test, where the null hypothesis is that the angles follow a uniform distribution. The null hypothesis is rejected when:

$$\frac{2R^2}{n} > \chi^2_{1,\alpha}, \tag{34}$$

where $\chi^2_{1,\alpha}$ is the *upper* α quantile of the Chi-square distribution with one degree of freedom.

Maximum likelihood angle. After rejecting the uniformity hypothesis, we assume that the angles follow a Von Mises distribution $\mathcal{M}(\mu, \kappa)$ with unknown mean μ and variance κ [11], and compute the maximum likelihood estimate of its mean μ as:

$$\bar{\vartheta} = \begin{cases} \tan^{-1}(\bar{S}/\bar{C}) & \text{if } \bar{C} \geq 0, \\ \pi + \tan^{-1}(\bar{S}/C) & \text{if } \bar{C} < 0, \end{cases} \tag{35}$$

where the inverse tangent is assumed to return values in $[-\pi/2, \pi/2]$, which yields $\bar{\vartheta} \in [-\pi/2, 3\pi/2]$. The angle is opportunely converted into the $[0, \pi]$ range to represent the *minimum* angle between two vectors: if $\bar{\vartheta} < 0$, $\bar{\vartheta} \leftarrow -\bar{\vartheta}$, and if $\bar{\vartheta} > \pi$, $\bar{\vartheta} \leftarrow \bar{\vartheta} - \pi$.

Confidence interval. Finally, we compute an approximate $100(1-\alpha)\%$ confidence interval for the mean μ of the Von Mises distribution as:

$$\bar{\vartheta} \pm \cos^{-1}\left(\sqrt{\frac{2n\left(2R^2 - n\chi^2_{1;\alpha}\right)}{R^2\left(4n - \chi^2_{1;\alpha}\right)}}\right) \qquad \text{if } \bar{R} \leq \frac{2}{3}, \tag{36}$$

or

$$\bar{\vartheta} \pm \cos^{-1}\left(\frac{\sqrt{n^2 - (n^2 - R^2)\exp\left(\chi^2_{1;\alpha}/n\right)}}{R}\right) \qquad \text{if } \bar{R} > \frac{2}{3}, \tag{37}$$

making sure once again that it refers to the $[0, \pi]$ range.

If the confidence interval is entirely contained in $(\pi/2, \pi)$, we can say with the desired confidence that

$$\angle\left(\mathbb{E}[g_i^l], \mathbb{E}[g_i^r]\right) > \frac{\pi}{2}, \tag{38}$$

that is to say, the left and right gradients point in opposite directions.

C Experimental Setting

In this section, we present the implementation details for what concerns the experiment shown in Section 6 and in Appendix D.

Algorithm Settings To fatherly reduce the variance in the estimation of the gradients, we adopt a version of GPOMDP with the optimal baseline ([20]).

$$\begin{aligned} \hat{\nabla}_{\theta_t} J_{\theta_t} &= \mathbb{E}_{p_\theta(\tau)}\left[\sum_{j=0}^{T-1}\sum_{t=0}^{j} \nabla_\theta \log \pi_\theta\left(a_t \mid s_t\right)\left(r_j - b_j\right)\right] \\ b_j &= \frac{\mathbb{E}_{p_\theta}(\tau)\left[\left(\sum_{t=0}^{j} \nabla_{\theta_h} \log \pi_\theta\left(a_t \mid s_t\right)\right)^2 r_j\right]}{\mathbb{E}_{p_\theta}(\tau)\left[\left(\sum_{t=0}^{j} \nabla_{\theta_h} \log \pi_\theta\left(a_t \mid s_t\right)\right)^2\right]} \end{aligned} \tag{39}$$

We tested our algorithm on 5 environment of increasing complexity. *Linear Quadratic Regulator* (LQR) ([9]); *Minigolf* in which the agent tries to throw a ball as near as possible to a target, here the cost is proportional to the distance from the target; *Swimmer-v4*, *HalfCheetah-v4*, *Ant-v4* from the MuJoCo suite ([27]). Details on the environmental parameters are shown in Table 2. In order to facilitate the exploration, since it is not a concern in this work, we added an action clipping to the MuJoCo environment, i.e., before computing the reward a clipping, on the limits of the action space, is performed from the drawn action. We adopted different learning rates η depending on the task we were observing, employing in some cases Adam ([8]) to adaptively set the step size. The exact method adopted for each environment is shown in Table 3. Moreover, for `GAPS` we set the limit of 50 split point candidates per iteration in every environment.

Computational Resources All the experiments were run on a 2023 14-inches MacBook Pro. The machine was equipped as follows:

- CPU: Apple M2 Pro (10 cores, 3.4 GHz);
- RAM: 16 GB;
- GPU: 16-core GPU.

In particular, $N = 1000$ trajectories of the MuJoCo environments with $H = 100$ scored ≈ 4 iterations per second for GPOMDP, while ≈ 3 iterations per second for `GAPS`. $N = 1000$ trajectories of the LQR environment with $H = 10$ scored ≈ 5 iterations per second respectively for both `GAPS` and GPOMDP. All the performance are run over a single CPU core.

Environment	Epoch	N	H	$d_{\mathcal{S}}$	$d_{\mathcal{A}}$
LQR 1-dimensional	1000	100	10	1	1
LQR 2-dimensional	1000	100	10	2	2
Minigolf	1000	100	200	1	1
Swimmer	1000	100	200	8	2
Half-Cheetah	1000	100	100	17	6
Ant	1500	100	200	27	8

Table 2: Parameters of the environment

Environment	η	Type
LQR 1-dimensional	0.01	Adam
LQR 2-dimensional	0.01	Adam
Minigolf	0.1	Adam
Swimmer	0.01	Adam
Half-Cheetah	0.01	Adam
Ant	0.01	Adam

(a) Learning rates for GPOMDP

Environment	η	Type
LQR 1-dimensional	0.001	Constant
LQR 2-dimensional	0.001	Constant
Minigolf	0.01	Constant
Swimmer	0.001	Constant
Half-Cheetah	1	Adam
Ant	0.0001	Constant

(b) Learning rates for `GAPS`

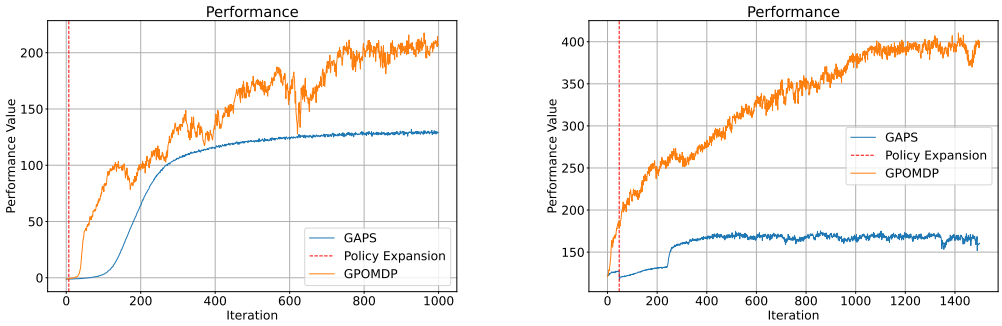
Table 3: Learning rates for the environment

D Additional Experimental Results

In this section we present some additional experimental results for what concerns the comparison between `GAPS` and GPOMDP, and the policy analysis.

We show the results gainend by learning in two high-dimensional environment taken from the MuJoCo suite ([27]): *Ant-v4* and *Half-Cheetah-v4*. Details on the environment parameters are shown in Appendix C. Here we show the learning curve obtained by the two algorithm, for Half-Cheetah in Figure 4 and for Ant in Figure 4. Although GPOMDP's performance is higher w.r.t. `GAPS`, it is interesting to notice how with significantly smaller parameters space (Table 2), the policies devised from `GAPS` manage to keep the performance positive, i.e., managing to sufficiently fullfil the environment task.

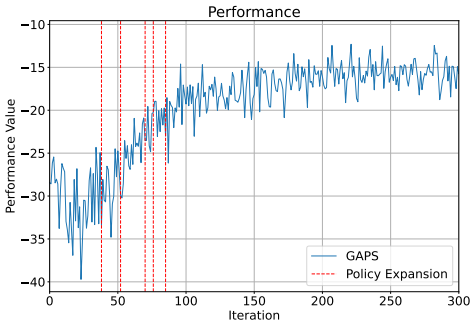
We also present a policy analysis on a one dimensional LQR case. Here for better comprehension GAPS was run for $P = 300$ iteration. From Figure 5, we show at first the learning curve with its various policy expasion step, to then visualize how without other division the policy remain stable without any kind of improvement. This result proves our claims of Section 5. We can in fact notice how without any further expansion the performance remains stable as the resulting gradient tend to be close to zero.



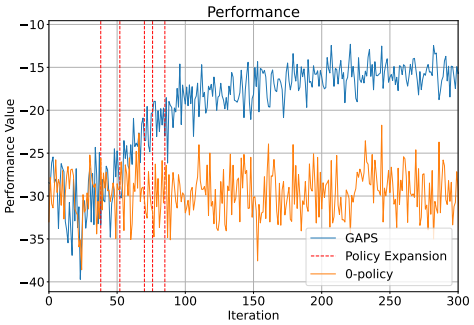
(a) Half-Cheetah

(b) Ant

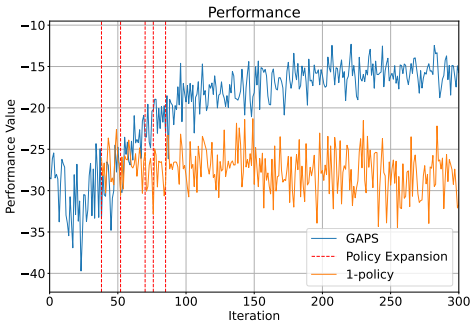
Figure 4: Learning curves on Half-Cheetah-v4 and And-v4 environments



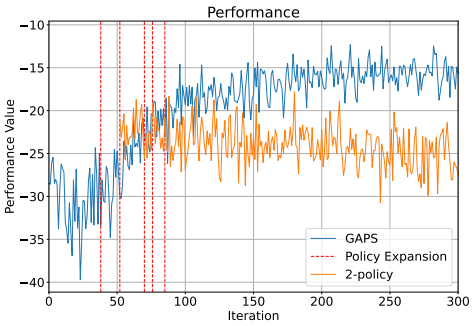
(a) Complete policy learning curve



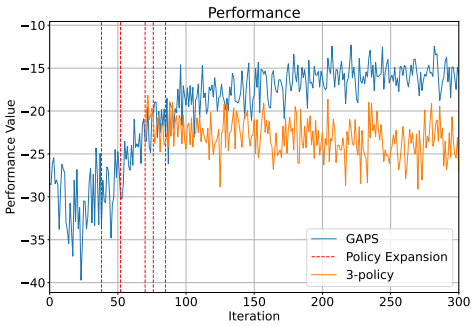
(b) First split learning curve



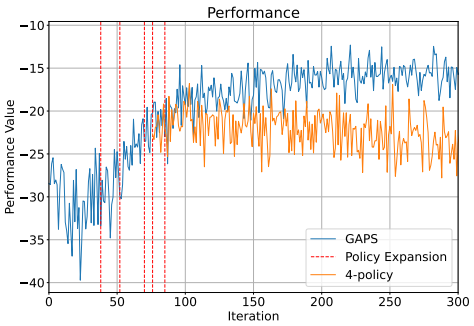
(c) Second split learning curve



(d) Third split learning curve



(e) Fourth split learning curve



(f) Fifth split learning curve

Figure 5: 1-dimension LQR policy sequence