EFFICIENT WASSERSTEIN NATURAL GRADIENTS FOR REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

A novel optimization approach is proposed for application to policy gradient methods and evolution strategies for reinforcement learning (RL). The procedure uses a computationally efficient *Wasserstein natural gradient* (WNG) descent that takes advantage of the geometry induced by a Wasserstein penalty to speed optimization. This method follows the recent theme in RL of including a divergence penalty in the objective to establish a trust region. Experiments on challenging tasks demonstrate improvements in both computational cost and performance over advanced baselines.

1 INTRODUCTION

Defining efficient optimization algorithms for reinforcement learning (RL) that are able to leverage a meaningful measure of similarity between policies is a longstanding and challenging problem (Lee & Popović, 2010; Meyerson et al., 2016; Conti et al., 2018b). Many such works rely on similarity measures such as the Kullback-Leibler (KL) divergence (Kullback & Leibler, 1951) to define procedures for updating the policy of an agent as it interacts with the environment. These are generally motivated by the need to maintain a small variation of the KL between successive updates in an off-policy context, to control the variance of the importance weights used the estimation of the gradient. This includes work by Kakade (2002); Schulman et al. (2015), who propose to use the *Fisher Natural Gradient* (Amari, 1997) as a way to update policies, using *local* geometric information to allow larger steps in directions where policies vary less; and the work of Schulman et al. (2017), which relies on a *global* measure of proximity using a soft KL penalty to the objective. While those methods achieve impressive performance, and the choice of the KL is well-motivated, one can still ask if it is possible to include information about the behavior of policies when measuring similarity, and whether this could lead to more efficient algorithm. Pacchiano et al. (2019) provide a first insight into this question, representing policies using *behavioral distributions* which incorporate information about the outcome of the policies in the environment. The Wasserstein Distance (WD) (Villani, 2009) between those behavioral distributions is then used as a similarity measure between their corresponding policies. They further propose to use such behavioral similarity as a *global* soft penalty to the total objective. Hence, like the KL penalty, proximity between policies is measured globally, and does not necessarily exploit the local geometry defined by the behavioral embeddings, which could yield substantial improvements.

In this work, we introduce new, efficient optimization methods for RL that incorporate the local geometry defined by the behavioral distributions for both policy gradient (PG) and evolution strategies (ES) approaches.

1- We leverage recent work in (Li & Montufar, 2018a;b; Li, 2018; Li & Zhao, 2019a) which introduces the notion of Wasserstein Information Matrix to define a *local behavioral similarity* measure between policies. This allows us the identify the Wasserstein Natural Gradient (WNG) as a key ingredient for optimization methods that rely on the local behavior of policies. To enable efficient estimation of WNG, we build on the recent work of Arbel et al. (2020), and further extend it to cases where the re-parameterization trick is not applicable, but only the score function of the model is available.

2- This allows us to introduce two novel methods: *Wasserstein natural policy gradients* (WNPG) and *Wasserstein natural evolution strategies* (WNES) which use the local behavioral structure of policies through WNG and can be easily incorporated to standard RL optimization routines. When

combined in addition with a global behavioral similarity such as WD penalty, we show substantial improvement over using the penalty alone without access to local information. We find that such WNG-based methods are especially useful on tasks in which initial progress is difficult.

3- Finally, we demonstrate, to our knowledge, the first in-depth comparative analysis of the FNG and WNG, highlighting a clear interpretable advantage of using WNG over FNG on tasks where the optimal solution is deterministic. This scenario arises frequently in ES and in policy optimization for MDPs (Puterman, 2010). This suggests that WNG could be a powerful tool for this class of problems, especially when reaching accurate solutions quickly is crucial.

In Section 2, we present a brief of policy gradient approaches and the role of divergence measures as regularization penalties. In Section 3 we introduce the WNG and detail its relationship with the FNG and the use of Wasserstein penalties, and in Section 4 we derive practical algorithms for applying the WNG to PG and ES. Section 5 contains our empirical results.

2 Background

Policy Gradient (PG) methods directly parametrize a policy π_θ, optimizing the parameter θ using stochastic gradient ascent on the expected total discounted reward $\mathcal{R}(\theta)$. An estimate $\hat{g}_k$ of the gradient of $\mathcal{R}(\theta)$ at θ_k can be computed by differentiating a surrogate objective $\mathcal{L}_\theta$ which often comes in two flavors, depending on whether training is *on-policy* (left) or *off-policy* (right):

$$\mathcal{L}(\theta) = \hat{\mathbb{E}}\left[\log \pi_\theta(a_t|s_t)\hat{A}_t\right], \qquad \text{or} \qquad \mathcal{L}(\theta) = \hat{\mathbb{E}}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}\hat{A}_t\right]. \tag{1}$$

The expectation $\hat{\mathbb{E}}$ is an empirical average over N trajectories $\tau_i = (s_0^i, a_0^i, r_0^i, ..., s_T^i, a_T^i, r_T^i)$ of state-action-rewards obtained by simulating from the environment using π_{θ_k}. The scalar $\hat{A}_t$ is an estimator of the advantage function and can be computed, for instance, using

$$\hat{A}_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{2}$$

where $\gamma \in [0, 1)$ is a discount factor and V is the value function often learned as a parametric function via temporal difference learning (Sutton & Barto, 2018). Reusing trajectories can reduce the computational cost but drastically increases the variance of the gradient estimator. This can lead to a drastic degradation of performance if done naïvely (Schulman et al., 2017). KL-based policy optimization (PO) aims at addressing these limitations.

KL-based PO methods ensure that the policy does not change substantially between successive updates, where change is measured by the KL divergence between the resulting action distributions. The general idea is to add either a hard KL constraint, as in TRPO Schulman et al. (2015), or a soft constraint, as in PPO Schulman et al. (2015), to encourage proximity between policies. In the first case, TRPO recovers the FNG with a step-size further adjusted using line-search to enforce the hard constraint. The FNG allows taking larger steps in directions where policy changes the least, thus reducing the number of updates required for optimization. In the second case, the soft constraint leads to an objective of the form:

$$\text{maximize}_\theta \; \mathcal{L}(\theta) - \beta\hat{\mathbb{E}}\left[\text{KL}(\pi_{\theta_k}(.|s_t), \pi_\theta(.|s_t))\right]. \tag{3}$$

The KL penalty prevents the updates from deviating too far from the current policy π_{θ_k}, thereby controlling the variance of the gradient estimator. This allows making multiple steps with the same simulated trajectories without degradation of performance. While both methods take into account the proximity between policies as measured using the KL, they do not take into account the *behavior* of such policies in the environment. Exploiting such information can greatly improve performance.

Behavior-Guided Policy Optimization. Motivated by the idea that policies can differ substantially as measured by their KL divergence but still *behave* similarly in the environment, Pacchiano et al. (2019) recently proposed to use a notion of proximity in *behavior* between policies for PO. Exploiting similarity in behavior during optimization allows to take larger steps in directions where policies behave similarly despite having a large KL divergence. To capture a sense of global behavior, they define a *behavioral embedding map* (BEM) Φ that maps every trajectory τ to a behavior variable $X = \Phi(\tau)$ belonging to some embedding space $\mathcal{E}$. The behavior variable X provides a simplified,

yet meaningful representation of each the trajectory τ. As a random variable, X is distributed according to a distribution q_θ, called the *behavior distribution*. Examples of Φ include simply returning the final state of a trajectory ($\Phi(\tau) = s_T$) or its concatenated actions ($\Phi(\tau) = [a_0, \ldots, a_T]$). Proximity between two policies π_θ and $\pi_{\theta'}$ is then measured using the Wasserstein distance between their *behavior distributions* q_θ and $q_{\theta'}$. Although, the KL could also be used in some cases, the Wasserstein distance has the advantage of being well-defined even for distributions with non-overlapping support, therefore allowing more freedom in choosing the embedding Φ (see Section 3.1). This leads to a penalized objective that regulates behavioral proximity:

$$\text{maximize}_\theta \; \mathcal{L}(\theta) - \frac{\beta}{2} \mathrm{W}_2(q_{\theta_k}, q_\theta), \tag{4}$$

where $\beta \in \mathbb{R}$ is a hyperparameter controlling the strength of the regularization. To compute the penalty, Pacchiano et al. (2019) use an iterative procedure from Genevay et al. (2016). This procedure remains accurate when the Wasserstein distance changes slowly between successive updates. This proximity can be achieved using large values for β, however, this also means that the policy is updated using small steps, which can slow down convergence especially when using gradient descent. Moreover, the penalty only accounts for *global* proximity in behavior and doesn't necessarily exploit the local geometry induced by the BEM, which can substantially improve convergence. We introduce an efficient method that exploits the local geometry induced by the BEM through the Wasserstein Natural gradient (WNG) leading to gains in performance at a reduced computational cost. When global proximity is important to the task, we show that using the Wasserstein penalty as in Equation (4) and optimizing it using the WNG yields more efficient updates, thus converging faster than simply optimizing Equation (4) using standard gradients.

3 Wasserstein Natural Gradient

The Wasserstein natural gradient (WNG) (Li & Montufar, 2018a;b) corresponds to the steepest-ascent direction of an objective within a trust region defined by the local behavior of the Wasserstein-2 distance (W_2). The W_2 between two nearby densities q_θ and $q_{\theta+u}$ can be approximated by computing the average cost of moving every sample X from q_θ to a new sample X' **approximately** distributed according to $q_{\theta+u}$ using an **optimal** vector field of the form $\nabla_x f_u(x)$ so that $X' = X + \nabla_x f_u(X)$ (see Figure 6 in appendix). Optimality of $\nabla_x f_u$ is defined as a trade-off between accurately moving mass from q_θ to $q_{\theta+u}$ and reducing the transport cost measured by the average squared norm of $\nabla_x f_u$

$$\sup_f \; \nabla_\theta \mathbb{E}_{q_\theta} \left[f(X) \right]^\top u - \frac{1}{2} \mathbb{E}_{q_\theta} \left[\|\nabla_x f(X)\|^2 \right], \tag{5}$$

where the optimization is over a suitable set of smooth real valued functions on $\mathcal{E}$. Proposition 1 makes this intuition more precise and defines the Wasserstein Information Matrix.

Proposition 1 (Adapted from Defintion 3 Li & Zhao (2019b)) *The second order Taylor expansion of W_2 between two nearby parametric probability distributions q_θ and $q_{\theta+u}$ is given by*

$$W_2^2(q_\theta, q_{\theta+u}) = u^\top G(\theta) u + o(\|u\|^2) \tag{6}$$

where $G(\theta)$ is the Wasserstein Information Matrix (WIM), with components in a basis $(e_1, ..., e_p)$

$$G_{j,j'}(\theta) = \mathbb{E}_{q_\theta} \left[\nabla_x f_j(X)^\top \nabla_x f_{j'}(X) \right]. \tag{7}$$

The functions f_j solve Equation (5) with u chosen as e_j. Moreover, for any given u, the solution f_u to Equation (5) satisfies $\mathbb{E}_\theta[\|\nabla_x f_u(X)\|^2] = u^\top G(\theta) u$.

When q_θ and $q_{\theta+u}$ are the behavioral embedding distributions of two policies π_θ and $\pi_{\theta+u}$, the function f_u allows to transport behavior from a policy π_θ to a behavior as close as possible to $\pi_{\theta+u}$ with the least cost. We thus refer to f_u as the *behavioral transport function*. The function f_u determines how hard it is to change behavior **locally** from policy π_θ in a direction u, thus providing a tool to find update directions u with either *maximal* or *minimal* change to the behavior.

Probing all directions in a basis $(e_1, ..., e_p)$ of parameters allows to construct the WIM $G(\theta)$ in Equation (7) which summarizes proximity in behavior along all possible directions u using

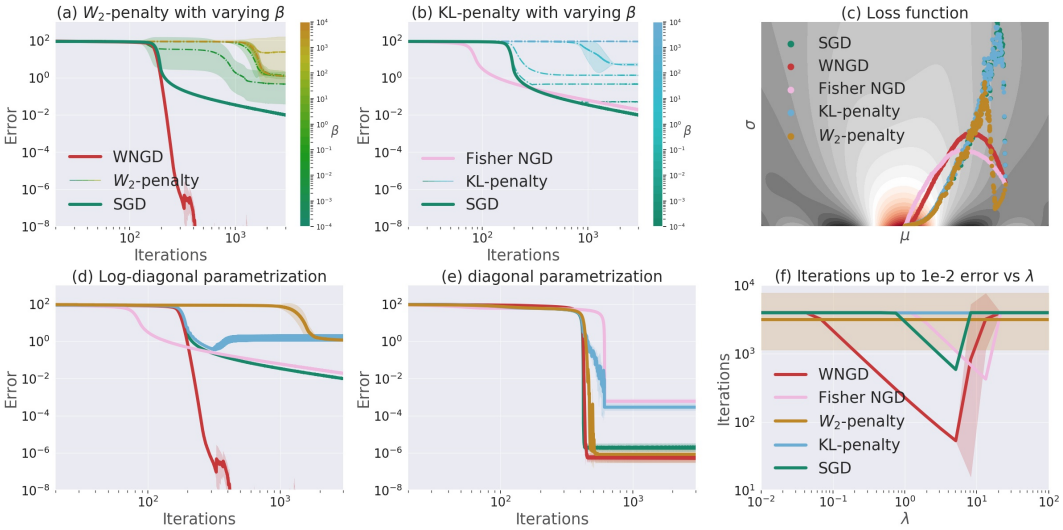

Figure 1: Different optimization methods using an objective $\mathcal{L}(\theta) = \mathbb{E}_{q_\theta}[\psi(x)]$ with gaussian q_θ of 100 dimension with $\theta = (mu, v)$ where mu is the mean, and the diagonal covariance matrix is parameterized either as $\Sigma_{ii} = e^{v_i}$ (log-diagonal) or v_i (diagonal). $\psi(x)$ is the sum of $sinc$ functions over all dimensions. Training up to 4000 iterations, with $\lambda = .9$ and $\beta = .1$ unless they are varied.

$u^\top G(\theta)u = \mathbb{E}_{q_\theta}[\|\nabla_x f_u(X)\|^2]$. For an objective $\mathcal{L}(\theta)$, such as the expected total reward of a policy, the Wasserstein natural gradient (WNG) is then defined as the direction u that locally maximizes $\mathcal{L}(\theta + u)$ the most with the least change in behavior as measured by f_u. Formally, the WNG is related to the usual Euclidean gradient $g = \nabla_\theta \mathcal{L}(\theta)$ by

$$g^W = \arg\max 2g^\top u - u^\top G(\theta)u. \tag{8}$$

From Equation (8), the WNG can be expressed in closed-form in terms of $G(\theta)$ and g as $g^W = G^{-1}(\theta)g$. Hence, WNG ascent is simply performed using the update equation $\theta_{k+1} = \theta_k + \lambda g_k^W$. We'll see in Section 4 how to estimate WNG efficiently without storing or explicitly inverting the matrix G. Next, we discuss the advantages of using WNG compared to other methods.

3.1 Why use the Wasserstein Natural Gradient?

To illustrate the advantages the WNG, we consider a simple setting where the objective is of the form $\mathcal{L}(\theta) = \mathbb{E}_{q_\theta}[\psi(x)]$, with q_θ being a gaussian distribution. The optimal solution in this example is a deterministic point mass located at the global optimum $x^\star$ of the function $\psi(x)$. This situation arises systematically in the context of ES when using a gaussian noise distribution with learnable mean and variance. Moreover, the optimal policy of a Markov Decision Processes (MDP) is necessarily deterministic (Puterman, 2010). Thus, despite its simplicity this example allows us to obtain closed form expressions for all methods while capturing a crucial property in many RL problems (deterministic optimal policies) which, as we will see, results in differing performance.

Wasserstein natural gradient vs Fisher natural gradient As shown in Figure 1 (d) and (e) for two different parameterizations of q_θ, FNG is faster at first then slows down to reach a final error of 10^{-4}. On the other hand, WNG is slower at first then transitions suddenly to an error of 10^{-8}. The optimal solution being deterministic, the variance of the gaussian q_θ needs to shrink to 0 as shown in (Figure 1 (c)). In this case, the KL blows-up, while the W_2 distance remains finite. As both natural gradients are derived from those two penalties (Theorem 2 of Appendix B), they inherit this behavior. This explains why, unlike the WNG, the FNG remains far from the optimum. Beyond this example, when the policy π_θ is defined only implicitly using a generative network as in cite, the FNG and KL penalty are ill-defined since π_{θ_k} and $\pi_{\theta_{k+1}}$ might have non-overlapping supports. However, the WNG remains well-defined (see Arbel et al. (2020)) and allows for more flexibility in representing policies such as behavioral embeddings.

Wasserstein penalty vs Wasserstein natural gradient The Wasserstein penalty Equation (4) encourages *global* proximity between updates q_{θ_k}. For small values of the penalty parameter β, the method behaves like standard gradient descent Figure 1 (a). As β increases, the penalty encourages more local updates and thus incorporates more information about the local geometry defined by q_θ. In fact it recovers the WNG direction (Theorem 2 of Appendix B) albeit with an infinitely small step-size which is detrimental to convergence of the algorithm. To avoid slowing-down, there is an intricate balance between the step-size and penalty β that needs to be maintained Schulman et al. (2017). All those issues are avoided when directly using the WNG as shown in Figure 1 (a) which performs the best and is tolerates the widest range of step-sizes Figure 1 (f).

Combining WNG and a Wasserstein penalty. The global proximity encouraged by a W_2 penalty can be useful on its own, for instance, to explicitly guarantee policy improvement as in (Pacchiano et al., 2019, Theorem 5.1). However, this requires estimating the W_2 at every iteration, which can be costly. Using WNG instead of the usual gradient can yield more efficient updates, thus reducing the number of time W_2 needs to be estimated. The speed-up can be understood as performing second-order optimization on the W_2 penalty since WNG arises precisely from a second order expansion of the W_2 distance as shown in Section 3 (See also Example 2 in Arbel et al. (2020)).

4 Policy Optimization using Behavioral Geometry

We now present practical algorithms to exploit the behavioral geometry induced by the embeddings Φ. We begin by describing how to efficiently estimate the WNG.

Efficient estimation of the WNG can be performed using kernel methods as shown in Arbel et al. (2020) in the case where the re-parametrization trick is applicable. This is the case, if for instance, the behavioral variable is the concatenation of actions $X = [a_0, ..., a_T]$ and if actions are sampled from a gaussian with mean and variance parameterized by a neural network, as is often done in practice for continuous state actions. Then X can be expressed as $X = B_\theta(Z)$ where B_θ is a known function and Z is an input sample consisting in the concatenation of states $[s_0, ..., s_T]$ and the gaussian noise used to generate the actions. However, the proposed algorithm is not readily applicable if for instance the behavioral variable X is a function of the reward.

We introduce a general procedure that extends the previous method to more general cases, including those where only the score $\nabla_\theta \log q_\theta$ is available without an explicit re-parametrization trick. The core idea is to approximate the functions f_{e_j} defining $G(\theta_k)$ in Equation (7) using a linear combinations of user-specified basis functions $(h_1(x), ..., h_M(x))$

$$\hat{f}_{e_j}(x) = \sum_{m=1}^{M} \alpha_m^j h_m(x), \tag{9}$$

The number M controls the computational cost of the estimation and is typically chosen in the order of $M = 10$. The basis can be chosen to be *data-dependent* using kernel methods. More precisely, we use the same approach as in Arbel et al. (2020), where we first subsample M data-points Y_m from a batch of N variables X_n and M indices i_m from $\{1, ..., d\}$ where d is the dimension of X_n. Then, each basis can of the form $h_m(x) = \partial_{i_m} K(Y_m, x)$ where K is a positive semi-definite kernel, such as the gaussian kernel $K(x, y) = \exp(-\frac{\|x-y\|^2}{\sigma})$. This choice of basis allows us to provide guarantees for functions f_j in terms of the batch size N and the number of basis points M (Arbel et al., 2020, Theorem 7). Plugging-in each $\hat{f}_j$ in the transport cost problem Equation (5) yields a quadratic problem of dimension M in the coefficients α^j:

$$\text{maximize}_{\alpha^j} \; 2J_{.,j}\alpha^j - (\alpha^j)^\top L \alpha^j$$

where L is a square matrix of size $M \times M$ independent of the index j and J is a Jacobian matrix of shape $M \times p$ with rows given by $J_{m,.} = \nabla_\theta \mathbb{E}_{p_{\theta_k}}[h_m(X)]$. There are two expressions for J, depending on the applicability of the re-parametrization trick or if the availability of the score

$$J_{m,.} = \hat{\mathbb{E}}_{q_\theta}[\nabla_x h_m(X) \nabla_\theta B_\theta(Z)] \qquad \text{or} \qquad J_{m,.} = \hat{\mathbb{E}}_{q_\theta}[\nabla_\theta \log q_\theta(X) h_m(X)] \tag{10}$$

Computing J can be done efficiently for moderate size M by first computing a surrogate vector of V of size M whose Jacobian recovers J using automatic differentiation software:

$$V_m = \hat{\mathbb{E}}_{q_\theta}\left[h_m(X_n)\right], \qquad \text{or} \qquad V_m = \hat{\mathbb{E}}_{q_\theta}\left[\log q_\theta(X_n) h_m(X_n)\right]. \tag{11}$$

The optimal coefficients α^j are then simply expressed as $\alpha = L^\dagger J$, pluggin-in the optimal functions in the expression of the Wasserstein Matrix information Equation (7), yields a low rank approximation of G of the form $\hat{G} = J^\top L^\dagger J$. By adding a small diagonal perturbation matrix ϵI, it is possible efficiently compute $(\hat{G} + \epsilon I)^{-1}\hat{g}$ using a generalized Woodbury matrix identity which yields an estimator for the Wasserstein Natural gradient

$$\hat{g}_k^W = \frac{1}{\epsilon}\left(\hat{g} - J^\top \left(JJ^\top + \epsilon L\right)^\dagger J\hat{g}\right). \tag{12}$$

The pseudo-inverse is only computed for a matrix of size M. Using the Jacobian-vector product, Equation (12) can be computed without storing large matrices G as shown in Algorithm 3.

Wasserstein Natural Policy Gradient (WNPG). It is possible to incorporate local information about the behavior of a policy in standard algorithms for policy gradient as summarized in Algorithm 1. In its simplest form, one first needs to compute the gradient $\hat{g}_k$ of the objective $\mathcal{L}(\theta_k)$ using the REINFORCE estimator computed using N trajectories τ_n. The trajectories are then used to compute the BEMs which are fed as input, along with the gradient $\hat{g}_k$ to get an estimate of the WNG g_k^W. Finally, the policy can be updated in the direction of g_k^W. Algorithm 1 can also be used in combination with an explicit W_2 penalty to control non-local changes in behavior of the policy thus ensuring a policy improvement property as in (Pacchiano et al., 2019, Theorem 5.1). In that case, WNG enhances convergences by acting as a second-order optimizer as discussed in Section 3.1. The standard gradient $\hat{g}_k$ in Algorithm 1 is then simply replaced by the one computed in (Pacchiano et al., 2019, Algorithm 3). We show in Section 5, that this combination, which we call behavior-guided WNPG (BG-WNPG), leads to the best performance.

Algorithm 1: Wasserstein Natural Policy Gradient

1: **Input** Initial policy π_{θ_0}
2: **for** iteration $k = 1, 2, ...$ do **do**
3: Obtain N rollouts $\{\tau\}_{n=1}^N$ of length T using policy π_{θ_k}
4: Compute loss $\mathcal{L}(\theta_k)$ in a forward pass
5: Compute gradient $\hat{g}_k$ in the backward pass on $\mathcal{L}(\theta_k)$
6: Compute Behavioral embeddings $\{X_n = \Phi(\tau^n)\}_{n=1}^N$
7: Compute WNG $\hat{g}_k^W$ using Algorithm 3 with samples $\{X_n\}_{n=1}^N$ and gradient estimate $\hat{g}_k$.
8: Update policy using: $\theta_{k+1} = \theta_k + \lambda \hat{g}_k^W$.
9: **end for**

Wasserstein Natural Evolution Strategies (WNES). ES treats the total reward observed on a trajectory under policy π_θ as a black-box function $\mathcal{L}(\theta)$ (Salimans et al., 2017; Mania et al., 2018; Choromanski et al., 2020). Evaluating it under N policies whose parameters $\widetilde{\theta}^n$ are gaussian perturbations centered around θ_k and with variance σ can give an estimate of the gradient of $\mathcal{L}(\theta_k)$:

$$\hat{g}_k = \frac{1}{N\sigma}\sum_{n=1}^{N}\left(\mathcal{L}(\widetilde{\theta}^n) - \mathcal{L}(\theta_k)\right)(\widetilde{\theta}^n - \theta_k). \tag{13}$$

Instead of directly updating the policy using Equation (13), it is possible to encourage either proximity or diversity in behavior using the embeddings $X_n = \Phi(\tau_n)$ of the trajectories τ_n generated for each perturbed policy $\pi_{\widetilde{\theta}_n}$. Those embeddings can be used as input to Algorithm 3 (see appendix), along with Equation (13) to estimate the $\hat{g}_k^W$, which captures similarity in behavior. The algorithm remains unchanged except for the estimation of the Jacobian J of Equation (10) which becomes

$$J_{m,\cdot} = \frac{1}{N\sigma}\sum_{n=1}^{N} h_m(X_n)(\widetilde{\theta}^n - \theta_k). \tag{14}$$

The policy parameter can then be updated using an interpolation between $\hat{g}_k$ and the WNG $\hat{g}_k^W$, i.e.,

$$\Delta\theta_k \propto (1-\delta)\hat{g}_k + \delta\hat{g}_k^W \tag{15}$$

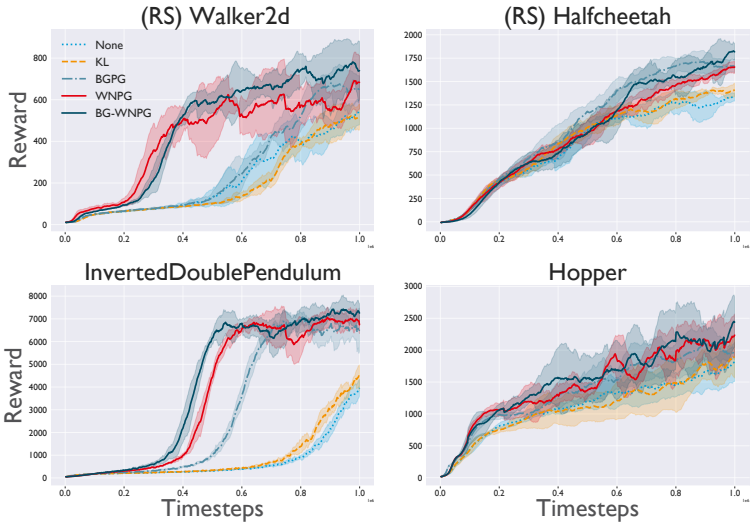

Figure 2: **WNG-based algorithms provide large gains on tasks where initial progress is difficult.** The performance mean $\pm$ standard deviation is plotted versus time steps for 5 random seeds on each task.

with $\delta \leq 1$ that can also be negative. Positive values for δ encourage proximity in behavior, the limit case being $\delta = 1$ where a full WNG step is taken. Negative values encourage repulsion and therefore need to compensated by $\hat{g}_k$ to ensure overall policy improvement. Algorithm 2 summarizes the whole procedure which can be easily adapted from existing ES implementations by calling a variant of Algorithm 3. In particular, it can also be used along with an explicit W_2 penalty, in which case the proposed algorithm in Pacchiano et al. (2019) is used to estimate the standard gradient $\hat{g}_k$ of the penalized loss. Then the policy is updated using Equation (15) instead of $\hat{g}_k$ as done in Pacchiano et al. (2019). We refer to this approach as behavior-guided WNES (BG-WNES).

Algorithm 2: Wasserstein Natural Evolution Strategies

1: **Input** Initial policy π_{θ_0}, $\alpha > 0$, $\delta \leq 1$
2: **for** iteration $k = 1, 2, \ldots$ do **do**
3: Sample $\epsilon_1, \ldots, \epsilon_n \sim \mathcal{N}(0, I)$.
4: Perform rollouts $\{\tau_n\}_{n-1}^N$ of length T using the perturbed parameters $\{\widetilde{\theta}^n = \theta_k + \sigma\epsilon_n\}_{n=1}^N$ and compute behavioral embeddings $\{X_n = \Phi(\tau^n)\}_{n=1}^N$
5: Compute gradient estimate of $\mathcal{L}(\widetilde{\theta}^n)$ using Equation (13) and trajectories $\{\tau^n\}_{n=1}^N$.
6: Compute Jacobian matrix J appearing in Algorithm 3 using Equation (14).
7: Compute WNG $\hat{g}_k^W$ using Algorithm 3, with samples $\{X_n\}_{i=1}^N$ and computed $\hat{g}_k$ and J.
8: Update policy using Equation (15).
9: **end for**

5 Experiments

We now test the performance of our estimators for both policy gradients (PG) and evolution strategies (ES) against their associated baseline methods. We seek to show that in addition to lowering improving efficiency, our approach can efffectively utilize the geometry induced by a Wasserstein penalty to improve performance, particularly when the optimization problem is ill-conditioned. Further experimental details can be found in the appendix. **Policy Gradients.** We first apply WNPG and BG-WNPG to challenging tasks from OpenAI Gym (Brockman et al., 2016) and Roboschool (RS). From Figure 2, we can see that WNPG matches or exceeds final performance compared to baseline methods on all tasks, with BG-WNPG demonstrating the furthest gains on all environments. It's also important to note that WNPG appears to offer the biggest advantage on tasks where initial progress is difficult. Arbel et al. (2020) showed that WNG performs most favorably compared to other methods when the optimization problem is ill-conditioned. We therefore computed the hessian matrix at

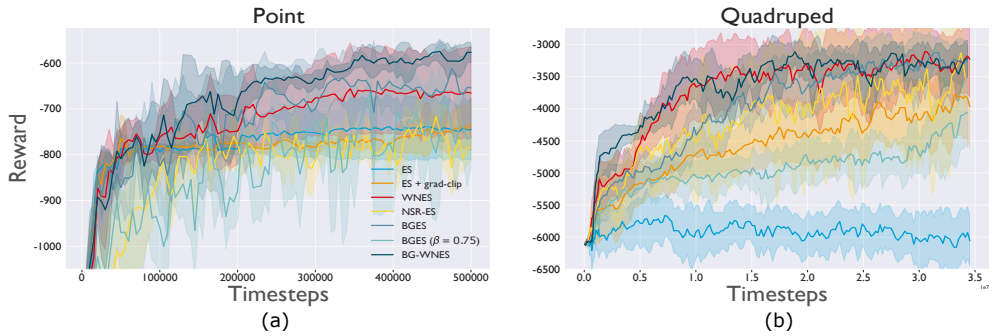

Figure 4: **WNES methods more reliably overcome local maxima.** Results obtained on the point (a) and quadruped (b) tasks. The mean $\pm$ standard deviation is plotted across 5 random seeds.

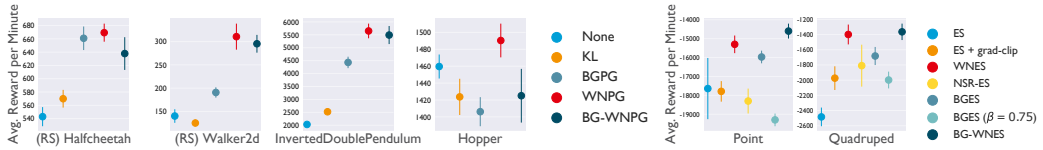

Figure 5: **WNG methods improve computational efficiency.** Average reward per minute is plotted for both gradient tasks (left) and ES tasks (right) for the runs depicted above.

the end of training for each task and measured the ratios of its largest eigenvalue to each successive eigenvalue (Figure 3). Larger ratios indicate ill-conditioning, and it is significant that WNG methods produce the greatest improvement on the environments with the poorest conditioning.

Evolution Strategies To test our estimator for WNES, as well as BG-WNES, we applied our approach to the environment introduced by Pacchiano et al. (2019), designed to test the ability of behavior-guided learning to succeed despite deceptive rewards. In the task, the agent receives a penalty proportional to its distance from a goal, but a wall is placed directly in the agent's path (Figure 7). This barrier induces a local maximum in progress—a naïve agent will simply walk directly towards the goal and get stuck at the barrier. The idea is that the behavioral repulsion fostered by applying a positive coefficient to the Wasserstein penalty ($\beta > 0$) will encourage the agent to seek novel policies, helping it to eventually circumvent the wall. As in Pacchiano et al. (2019), we test two agents, a simple point and a quadruped. As baselines, we tested vanilla ES as described by Salimans et al. (2017), ES with gradient norm clipping, BGES (Pacchiano et al., 2019), and NSR-ES (Conti et al., 2018a). In Figure 4, we can see that WNES and BG-WNES improve over the baselines for both agents. To test that the improvement shown by BG-WNES wasn't simply a case of additional "repulsion" supplied by the WNG to BGES, we also tested BGES with an increased $\beta = 0.75$, compared to the default of 0.5. This resulted in a decrease in performance, attesting to the unique benefit provided by the WNES estimator.

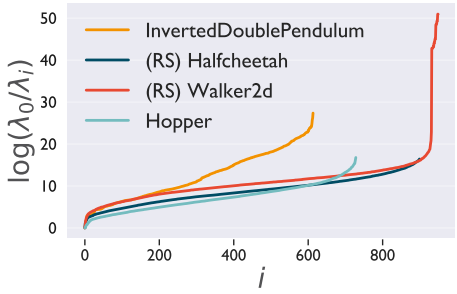

Figure 3: Condition numbers for different tasks.

Computational Efficiency We define the *computational efficiency* of an algorithm as the rate with which it accumulates reward relative to its runtime. To test the computational efficiency of our approach, we plotted the total reward divided by wall clock time obtained by each agent for each task (Fig. 5). Methods using a WNG estimator were the most efficient on each task for both PG and ES agents. On several environments used for the policy gradients tasks, the added cost of BG-WNPG reduced its efficiency, despite having the highest absolute performance.

REFERENCES

Shun-ichi Amari. Neural learning in structured parameter spaces - natural riemannian gradient. In M. C. Mozer, M. I. Jordan, and T. Petsche (eds.), *Advances in Neural Information Processing Systems 9*, pp. 127–133. MIT Press, 1997. URL http://papers.nips.cc/paper/1248-neural-learning-in-structured-parameter-spaces-natural-riemannian-gradient.pdf.

M Arbel, A Gretton, W Li, and G Montufar. Kernelized wasserstein natural gradient. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Hklz71rYvS.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Krzysztof Choromanski, Aldo Pacchiano, Jack Parker-Holder, Yunhao Tang, Deepali Jain, Yuxiang Yang, Atil Iscen, Jasmine Hsu, and Vikas Sindhwani. Provably robust blackbox optimization for reinforcement learning. In *Conference on Robot Learning*, pp. 683–696, 2020.

Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth Stanley, and Jeff Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. In *Advances in Neural Information Processing Systems 31*, pp. 5027–5038. Curran Associates, Inc., 2018a.

Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth Stanley, and Jeff Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. In *Advances in neural information processing systems*, pp. 5027–5038, 2018b.

Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. https://github.com/openai/baselines, 2017.

Aude Genevay, Marco Cuturi, Gabriel Peyré, and Francis Bach. Stochastic optimization for large-scale optimal transport. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 29*, pp. 3440–3448. Curran Associates, Inc., 2016. URL http://papers.nips.cc/paper/6566-stochastic-optimization-for-large-scale-optimal-transport.pdf.

Sham M Kakade. A natural policy gradient. In *Advances in neural information processing systems*, pp. 1531–1538, 2002.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

Seong Jae Lee and Zoran Popović. Learning behavior styles with inverse reinforcement learning. *ACM transactions on graphics (TOG)*, 29(4):1–7, 2010.

Wuchen Li. Geometry of probability simplex via optimal transport. *arXiv:1803.06360 [math]*, March 2018. URL http://arxiv.org/abs/1803.06360. arXiv: 1803.06360.

Wuchen Li and Guido Montufar. Natural gradient via optimal transport. *arXiv:1803.07033 [cs, math]*, March 2018a. URL http://arxiv.org/abs/1803.07033. arXiv: 1803.07033.

Wuchen Li and Guido Montufar. Ricci curvature for parametric statistics via optimal transport. *arXiv:1807.07095 [cs, math, stat]*, July 2018b. URL http://arxiv.org/abs/1807.07095. arXiv: 1807.07095.

Wuchen Li and Jiaxi Zhao. Wasserstein information matrix. *arXiv:1910.11248 [cs, math, stat]*, November 2019a. URL http://arxiv.org/abs/1910.11248. arXiv: 1910.11248.

Wuchen Li and Jiaxi Zhao. Wasserstein information matrix. *arXiv preprint arXiv:1910.11248*, 2019b.

Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search of static linear policies is competitive for reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 1800–1809, 2018.

Elliot Meyerson, Joel Lehman, and Risto Miikkulainen. Learning behavior characterizations for novelty search. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pp. 149–156, 2016.

Aldo Pacchiano, Jack Parker-Holder, Yunhao Tang, Anna Choromanska, Krzysztof Choromanski, and Michael I Jordan. Learning to score behaviors for guided policy optimization. *arXiv preprint arXiv:1906.04349*, 2019.

Martin L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. 2010.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *arXiv e-prints*, art. arXiv:1703.03864, March 2017.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015. URL http://arxiv.org/abs/1502.05477.

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

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Cedric Villani. Optimal transport: Old and new. Technical report, 2009.

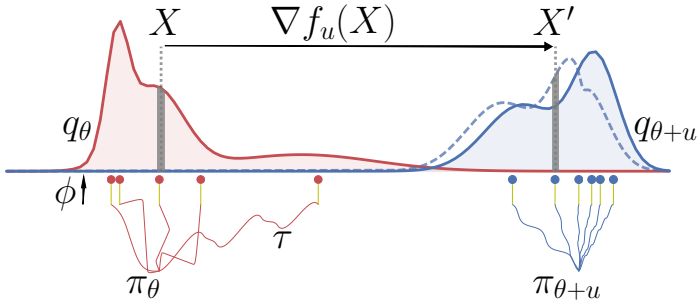

Figure 6: Wasserstein.

A Background

A.1 Policy Optimization

An agent interacting with an environment form a system that can be described by a *state* variable s belonging to a *state space* $\mathcal{S}$. In the Markov Decision Process (MDP) setting, the agent can interact with the environment by taking an action a from a set of possible actions $\mathcal{A}$ given the current state s of the system. As a consequence, the system moves to a new state s' according to a probability transition function $P(s'|a, s)$ which describes the probability of moving to state s' given the previous state s and action a. The agent also receives a partial reward r which can be expressed as a possibly randomized function of the new state s', $r = r(s')$. The agent has access to a set of possible *policies* $\pi_\theta(a|s)$ parametrized by $\theta \in \mathbb{R}^p$ and that generates an action a given a current state s. Thus, each policy can be seen as a probability distribution conditioned a state s. Using the same policy induces a whole trajectory of state-action-rewards $\tau = (s_t, a_t, r_t)_{t\geq 0}$ which can be viewed as a sample from a trajectory distribution $\mathbb{P}_\theta$ defined over the space of possible trajectories τ. Hence, for a given random trajectory τ induced by a policy π_θ, the agent receives a total discounted reward $R(\tau) := \sum_{t=1}^{\infty} \gamma^{t-1} r(s_t)$ with discount factor $0 < \gamma < 1$. This allows to define the *value* function as the expected total reward conditioned on a particular initial state s:

$$V_\theta(s_t) = \mathbb{E}_{\mathbb{P}_\theta|s_t}\left[\sum_{l=1}^{\infty} \gamma^{l-1} r(s_{l+t})\right]. \tag{16}$$

When the gradient of the *score function* $\nabla \log \pi_\theta(a|s)$ is available, the policy gradient theorem allows to express the gradient of $\mathcal{R}(\theta)$:

$$\nabla_\theta \mathcal{R}(\theta) = \mathbb{E}_{\mathbb{P}_\theta}\left[\sum_{t=0}^{\infty} \gamma^t \nabla \log \pi_\theta(a_t|s_t) A_\theta(s_t, a_t)\right] \tag{17}$$

where the expectation is taken over trajectories τ under $\mathbb{P}_\theta$ and $A_\theta(s, a)$ represents the *advantage* function which can be expressed in terms of the value function $V_\theta(s)$ in terms of

$$A_\theta(s_t, a_t) = \mathbb{E}_{s_{t+1}|s_t, a_t}\left[r(s_{t+1}) + \gamma V_\theta(s_{t+1})\right] - V_\theta(s_t).$$

The agent seeks an optimal policy $\pi_{\theta^\star}$ that maximizes the expected total reward under the trajectory distribution: $\mathcal{R}(\theta) = \mathbb{E}_{\mathbb{P}_\theta}[R(\tau)]$.

B Wasserstein Natural gradient

Connection to the Fisher natural gradient and proximal methods. Both WNG and FNG are obtained from a proximity measure between probability distributions:

Proposition 2 *Let $D(\theta, \theta')$ be either the KL-divergence KL$(\pi_\theta, \pi_{\theta'})$ or the Wasserstein-2 distance between the behavioral distributions $W_2(q_\theta, q_{\theta'})$ and let g^D be either the FNG g^F or WNG g^W, then*

$$g_k^D = \lim_{\beta \to +\infty} \arg\max_u \beta \left(\mathcal{L}(\theta_k + \beta^{-1}u) - \mathcal{L}(\theta_k) - \frac{\beta}{2} D\left(\theta_k, \theta_k + \beta^{-1}u\right)\right) \tag{18}$$

Equation (18) simply states that the both WNG and FNG arise as limit cases of penalized objectives provided the strength of the penalty β diverges to infinity and the step-size is shrank proportionally to β^{-1}. An additional global rescaling by β of the total objective prevents it from collapsing to 0. Intuitively, performing a Taylor expansion of Equation (18) recovers an equation similar to Equation (8). Equation (18) shows that using a penalty that encourages **global** proximity between successive policies, it is possible to recover the **local** geometry of policies (captured by the local) by increasing the strength of the penalty using appropriate re-scaling. This also informally shows why both natural gradients are said to be invariant to re-parametrization ((Arbel et al., 2020, Proposition 1)), since both KL and W_2 remains unchanged if q_θ is parameterized in a different way.

C Algorithm for estimating WNG

Algorithm 3: Efficient Wasserstein Natural Gradient

Input mini-batch of samples $\{X_n\}_{n=1}^N$ distributed according to q_θ, gradient direction $\hat{g}$, basis functions $h_1, ..., h_M$, regularization parameter ϵ.
Output Wasserstein Natural gradient $\hat{g}^W$
Compute a matrix C of shape $M \times Nd$ using $C_{m,(n,i)} \leftarrow \partial_i h_m(X_n)$.
Compute similarity matrix $L \leftarrow \frac{1}{N} CC^T$.
Compute surrogate vector V using Equation (11).
for iteration$= 1, 2, ...M$ do **do**
    Use automatic differentiation on V_m to compute Jacobian matrix J in Equation (10).
end for
Compute a matrix D of shape $M \times M$ using $D \leftarrow JJ^\top + \epsilon L$.
Compute a vector b of size M using $b \leftarrow J\hat{g}$.
Solve linear system of size $M : b \leftarrow \texttt{solve}\ (D, b)$
Return $\hat{g}^W \leftarrow \frac{1}{\epsilon}(\hat{g} - J^\top b)$

Algorithm 4: Similarity

Input N, M, K, $\hat{g}$, ϵ
Ouput $\hat{g}^W$
Get a minibatch of samples $Z_1, ...Z_N$.
Compute embedded samples $X_i = B_\theta(Z_i) \in \mathbb{R}^d$
Randomly sample $Y_1, ...Y_M$ from $X_1, ...X_N$ without replacement
Randomly sample indices $i_1, ..., i_M$ from $\{1, ..., p\}$ without replacement
Compute a similarity matrix C of shape $M \times Nd$:

$$C_{m,(n,i)} \leftarrow \partial_{i_m} \partial_{i+d} K(Y_m, X_n) \tag{19}$$

Compute a summary vector V of size M

$$V_m \leftarrow \frac{1}{N} \sum_{i=1}^{N} \partial_{i_m} K(Y_m, X_n) \tag{20}$$

for iteration$= 1, 2, ...M$ do **do**
    Using automatic differentiation compute the gradient of V_m w.r.t $\theta \in \mathbb{R}^p$.
    Collect gradients in a Jacobian matrix J of shape $M \times p$.
end for
Compute a matrix G of shape $M \times M$ using $G \leftarrow JJ^\top + \frac{\epsilon}{N} CC^\top$.
Collect product $J\hat{g}$ in a vector $\widetilde{g}$ of shape M.
Solve linear system of size $M : b \leftarrow \texttt{solve}\ (G, \widetilde{g})$
Return $\frac{1}{\epsilon}(\hat{g} - J^\top b)$

D ADDITIONAL TECHNICAL DETAILS

Consider a policy π_θ and denote by $\mathcal{R}(\tau)$ the reward of a sample τ from $\mathbb{P}_{\pi_\theta}$. The agent tries to maximize the expected reward:

$$l(\theta) = \mathbb{E}_{\mathbb{P}_{\pi_\theta}}\left[\mathcal{R}(\tau)\right]. \tag{21}$$

We are interested in the setting where $l(\theta)$ is maximized using stochastic gradient methods while maintaining a small Wasserstein distance between successive Behavioral distributions. More precisely, denote by Φ the Behavioral Embedding Map as in Pacchiano et al. (2019) which maps a trajectory τ to a feature (*final state*, *actions* or *total reward*) and let $\mathbb{P}^{\phi}_{\pi_\theta}$ be the push-forward distribution of $\mathbb{P}_{\pi_\theta}$ by ϕ as in Pacchiano et al. (2019). We would like to ensure that $W_2(\mathbb{P}^{\phi}_{\pi_{\theta_{k+1}}}, \mathbb{P}^{\phi}_{\pi_{\theta_k}})$ remains small between two successive iterates θ_k and θ_{k+1}.

To this end, Pacchiano et al. (2019) consider the penalized objective:

$$f(\theta) := -l(\theta) + \frac{\beta}{2} WD_\lambda(\mathbb{P}^{\phi}_{\pi_\theta}, \mathbb{P}^{\phi}_{\pi_{\theta_k}}) \tag{22}$$

Where WD_λ is the entropy regularized Wasserstein distance, with regularization parameter $\lambda > 0$. Hence, minimizing f in θ gives the next iterate θ_{k+1} while maintaining a small distance $WD_\lambda(\mathbb{P}^{\phi}_{\pi_{\theta_{k+1}}}, \mathbb{P}^{\phi}_{\pi_{\theta_k}})$. However, this process is costly since each iteration requires minimizing Equation (22). Instead, we propose to use the Wasserstein Natural gradient estimator introduced in Arbel et al. (2020) which is formally obtained by letting β tends to ∞. In this case, Equation (22) is replaced by the following cost:

$$f(\theta) := -\nabla l(\theta_k)^\top(\theta - \theta_k) + \frac{\beta}{2}(\theta - \theta_k)^\top G(\theta_k)(\theta - \theta_k) \tag{23}$$

where $G(\theta_k)$ is the Wasserstein Information Matrix. Minimizing Equation (23) gives the following update:

$$\theta_{k+1} = \theta_k + \beta G(\theta_k)^{-1}\nabla l(\theta_k). \tag{24}$$

While computing and inverting $G(\theta_k)^{-1}$ in closed form is expensive, Arbel et al. (2020) shows that scalable approximation to Equation (24) are obtained using:

$$\theta_{k+1} = \theta_k + \frac{\beta}{\epsilon}\left(\nabla l(\theta_k) - M_{\theta_k}(f^*)\right) \tag{25}$$

where $\epsilon > 0$ is a regularization parameter and $M_\theta(f)$ is a vector whose components are given by:

$$M_\theta(f)_i = \lim_{\gamma} \gamma^{-1}\left(\mathbb{E}_{\mathbb{P}_{\pi_{\theta+\gamma e_i}}}[f(\phi(\tau))] - \mathbb{E}_{\mathbb{P}_{\pi_\theta}}[f(\phi(\tau))]\right) \tag{26}$$

and $f^\star$ is obtained by minimizing the objective $\mathcal{J}(f)$ over an RKHS space:

$$\mathcal{J}(f) := \mathbb{E}_{\mathbb{P}_{\pi_\theta}}[\|\nabla f(\phi(\tau))\|^2] + \frac{1}{\epsilon}\|\nabla l(\theta_k) - M_{\theta_k}(f)\|^2 + \lambda\|f\|^2_{\mathcal{H}} \tag{27}$$

To figure out exact expression of the updates, it remains to find explicit expressions for $M_\theta(f)$. We distinguish two cases **gradient-based optimization** and **gradient-free optimization**.

D.1 GRADIENT-BASED OPTIMIZATION

When the trajectories τ are obtained through an mapping $h_\theta(z, s) := \tau$ where z is a latent noise and s is a state, then Arbel et al. (2020) shows that $M_\theta(f)$ is expressed in terms of the jacobian of h_θ:

$$M_\theta(f) = \mathbb{E}_{Z,S}\left[\nabla f(h_\theta(Z, S))\nabla_\theta h_\theta(Z, S)\right] \tag{28}$$

This leads to the algorithm implemented in Arbel et al. (2020).

Sometimes the mapping $h_\theta(z, s) := \tau$ is not differentiable neither is the cost θ. In this case, it advantageous to use Evolution Strategies. In our setting, this means that the policy π_θ is replaces by a perturbed policy π_ψ where ψ is a sample from a gaussian $p_{\theta,\sigma}$ with mean θ and variance σ. This

allows to obtain gradient wrt θ after smoothing by averaging over several draws ψ. More precisely, the loss l is replaced by the averaged loss $\bar{l}(\theta)$:

$$\bar{l}(\theta) = \mathbb{E}_{\psi \sim p_{\theta,\sigma}}[l(\psi)]. \tag{29}$$

with gradient simply given by:

$$\nabla \bar{l}(\theta) := \frac{1}{\sigma} \mathbb{E}_{\psi \sim p_{\theta,\sigma}}[l(\psi)(\psi - \theta)] \tag{30}$$

Moreover, the vector $M_\theta(f)$ is replaced by an averaged one:

$$\bar{M}_\theta(f)_i = \lim_\gamma \gamma^{-1} \left(\mathbb{E}_{\psi \sim p_{\theta+\gamma e_i,\sigma}}[\mathbb{E}_{\mathbb{P}_{\pi_\psi}}[f(\phi(\tau))]] - \mathbb{E}_{\psi \sim p_{\theta,\sigma}}[\mathbb{E}_{\mathbb{P}_{\pi_\psi}}[f(\phi(\tau))]] \right) \tag{31}$$

In this case it is easy to see that $\bar{M}_\theta(f)$ admits a simple expression:

$$\bar{M}_\theta(f) = \frac{1}{\sigma} \mathbb{E}_{\psi \sim p_{\theta,\sigma}}[\mathbb{E}_{\mathbb{P}_{\pi_\psi}}[f(\phi(\tau))](\psi - \theta)] \tag{32}$$

Finally, the objective in Equation (27) is replaced by an averaged one $\bar{\mathcal{J}}(f)$:

$$\bar{\mathcal{J}}(f) := \mathbb{E}_{\psi \sim p_{\theta,\sigma}} \mathbb{E}_{\mathbb{P}_{\pi_\psi}}[\|\nabla f(\phi(\tau))\|^2] + \frac{1}{\epsilon} \|\nabla \bar{l}(\theta_k) - \bar{M}_{\theta_k}(f^\star)\|^2 + \lambda \|f\|^2_{\mathcal{H}} \tag{33}$$

This avoids computing the jacobian wrt to $h_\theta(Z, S)$ which might be unknown. Finally, minimizing $\bar{\mathcal{J}}(f)$ can be done using Nystrom methods as in Arbel et al. (2020).

Expression of the estimator Let ψ^n be noisy parameters with mean θ and X_n the corresponding behavioral embeddings. Let $(Y_m)_{1 \leq m \leq M}$ be M uniform subsamples from $(X_n)_{1 \leq n \leq N}$. The estimator of the natural gradient is given by:

$$\widehat{\nabla^W \mathcal{L}(\theta)} = \frac{1}{\epsilon} \left(D(\theta)^{-1} - D(\theta)^{-1} T^\top \left(T D(\theta)^{-1} T^\top + \lambda \epsilon K + \frac{\epsilon}{N} C C^\top \right)^\dagger T D(\theta)^{-1} \right) \widehat{\nabla \mathcal{L}(\theta)}, \tag{34}$$

where C and K are matrices in $\mathbb{R}^{M \times Nd}$ and $\mathbb{R}^{M \times M}$ given by

$$C_{m,(n,i)} = \partial_{i_m} \partial_{i+d} k(Y_m, X_n), \qquad K_{m,m'} = \partial_{i_m} \partial_{i_{m'}+d} k(Y_m, Y_{m'}), \tag{35}$$

while T is a matrix in $\mathbb{R}^{M \times q}$ given by:

$$T = \frac{1}{N\sigma} \sum_{i=1}^{N} \partial_{i_m} k(Y_m, X_n)(\psi^n - \theta) \tag{36}$$

The matrix $D(\theta)$ is diagonal and can be computed in a similar way as in Arbel et al. (2020).

E Additional Experimental Details

E.1 Policy Gradient Tasks

We conserve all baseline and shared hyperparameters used by Pacchiano et al. (2019). More precisely, for each task we ran a hyperparameter sweep over learning rates in the set $\{$1e-5, 5e-5, 1e-4, 3e-4$\}$, and used the concatenation-of-actions behavioral embedding $\Phi(\tau) = [a_0, a_1, \ldots, a_T]$ with the base network implementation the same as Dhariwal et al. (2017).

The WNG hyperparameters were also left the same as in Arbel et al. (2020).

E.2 Evolution Strategies Tasks

As with the policy gradient tasks, we conserved all baseline and shared hyperparameters used by Pacchiano et al. (2019). Specifically, for the point task, we set the learning rate to be $\eta = 0.1$, the standard deviation of the noise to be $\sigma = 0.01$, the rollout length H was 50 time steps, and the behavioral embedding function to be the last state $\Phi(\tau) = s_H$. For the quadruped task we set $\eta = 0.02$, $\sigma = 0.02$, $H = 400$, and $\Phi(\tau) = \sum_{t=0}^{H} r_t \left(\sum_{i=0}^{t} e_i \right)$ (reward-to-go encoding; see Pacchiano et al. (2019) for more details). Both tasks used 1000-dimensional random features and embeddings from the $n = 2$ previous policies to compute the WD.

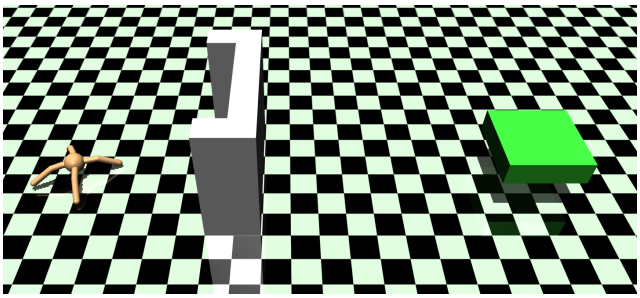

Figure 7: A visualization of the quadruped task. The agent receives receives more reward the closer it is to the goal (green). A naïve agent will get stuck in the local maximum at the wall if it attempts to move directly to the goal.