Trust Region Policy Optimization

John Schulman   JOSCHU@EECS.BERKELEY.EDU
Sergey Levine   SLEVINE@EECS.BERKELEY.EDU
Philipp Moritz   PCMORITZ@EECS.BERKELEY.EDU
Michael Jordan   JORDAN@CS.BERKELEY.EDU
Pieter Abbeel   PABBEEL@CS.BERKELEY.EDU

University of California, Berkeley, Department of Electrical Engineering and Computer Sciences

Abstract

We describe an iterative procedure for optimizing policies, with guaranteed monotonic improvement. By making several approximations to the theoretically-justified procedure, we develop a practical algorithm, called Trust Region Policy Optimization (TRPO). This algorithm is similar to natural policy gradient methods and is effective for optimizing large nonlinear policies such as neural networks. Our experiments demonstrate its robust performance on a wide variety of tasks: learning simulated robotic swimming, hopping, and walking gaits; and playing Atari games using images of the screen as input. Despite its approximations that deviate from the theory, TRPO tends to give monotonic improvement, with little tuning of hyperparameters.

1 Introduction

Most algorithms for policy optimization can be classified into three broad categories: (1) policy iteration methods, which alternate between estimating the value function under the current policy and improving the policy (Bertsekas, 2005); (2) policy gradient methods, which use an estimator of the gradient of the expected return (total reward) obtained from sample trajectories (Peters & Schaal, 2008a) (and which, as we later discuss, have a close connection to policy iteration); and (3) derivative-free optimization methods, such as the cross-entropy method (CEM) and covariance matrix adaptation (CMA), which treat the return as a black box function to be optimized in terms of the policy parameters (Szita & Lörincz, 2006).

General derivative-free stochastic optimization methods such as CEM and CMA are preferred on many problems, because they achieve good results while being simple to understand and implement. For example, while

Tetris is a classic benchmark problem for approximate dynamic programming (ADP) methods, stochastic optimization methods are difficult to beat on this task (Gabillon et al., 2013). For continuous control problems, methods like CMA have been successful at learning control policies for challenging tasks like locomotion when provided with hand-engineered policy classes with low-dimensional parameterizations (Wampler & Popović, 2009). The inability of ADP and gradient-based methods to consistently beat gradient-free random search is unsatisfying, since gradient-based optimization algorithms enjoy much better sample complexity guarantees than gradient-free methods (Nemirovski, 2005). Continuous gradient-based optimization has been very successful at learning function approximators for supervised learning tasks with huge numbers of parameters, and extending their success to reinforcement learning would allow for efficient training of complex and powerful policies.

In this article, we first prove that minimizing a certain surrogate objective function guarantees policy improvement with non-trivial step sizes. Then we make a series of approximations to the theoretically-justified algorithm, yielding a practical algorithm, which we call trust region policy optimization (TRPO). We describe two variants of this algorithm: first, the *single-path* method, which can be applied in the model-free setting; second, the *vine* method, which requires the system to be restored to particular states, which is typically only possible in simulation. These algorithms are scalable and can optimize nonlinear policies with tens of thousands of parameters, which have previously posed a major challenge for model-free policy search (Deisenroth et al., 2013). In our experiments, we show that the same TRPO methods can learn complex policies for swimming, hopping, and walking, as well as playing Atari games directly from raw images.

2 Preliminaries

Consider an infinite-horizon discounted Markov decision process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma)$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the transition probability distri-

bution, $r : \mathcal{S} \to \mathbb{R}$ is the reward function, $\rho_0 : \mathcal{S} \to \mathbb{R}$ is the distribution of the initial state s_0, and $\gamma \in (0, 1)$ is the discount factor.

Let π denote a stochastic policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, and let $\eta(\pi)$ denote its expected discounted reward:

$$\eta(\pi) = \mathbb{E}_{s_0, a_0, \dots} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t) \right] , \text{ where}$$

$$s_0 \sim \rho_0(s_0), \ a_t \sim \pi(a_t | s_t), \ s_{t+1} \sim P(s_{t+1} | s_t, a_t).$$

We will use the following standard definitions of the state-action value function Q_π, the value function V_π, and the advantage function A_π:

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \dots} \left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l}) \right] ,$$

$$V_\pi(s_t) = \mathbb{E}_{a_t, s_{t+1}, \dots} \left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l}) \right] ,$$

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s), \text{ where}$$

$$a_t \sim \pi(a_t | s_t), s_{t+1} \sim P(s_{t+1} | s_t, a_t) \text{ for } t \geq 0.$$

The following useful identity expresses the expected return of another policy $\tilde{\pi}$ in terms of the advantage over π, accumulated over timesteps (see Kakade & Langford (2002) or Appendix A for proof):

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{s_0, a_0, \dots \sim \tilde{\pi}} \left[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t) \right] \quad (1)$$

where the notation $\mathbb{E}_{s_0, a_0, \dots \sim \tilde{\pi}} [\dots]$ indicates that actions are sampled $a_t \sim \tilde{\pi}(\cdot | s_t)$. Let ρ_π be the (unnormalized) discounted visitation frequencies

$$\rho_\pi(s) = P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \dots,$$

where $s_0 \sim \rho_0$ and the actions are chosen according to π. We can rewrite Equation (1) with a sum over states instead of timesteps:

$$\eta(\tilde{\pi}) = \eta(\pi) + \sum_{t=0}^{\infty} \sum_s P(s_t = s | \tilde{\pi}) \sum_a \tilde{\pi}(a|s) \gamma^t A_\pi(s, a)$$

$$= \eta(\pi) + \sum_s \sum_{t=0}^{\infty} \gamma^t P(s_t = s | \tilde{\pi}) \sum_a \tilde{\pi}(a|s) A_\pi(s, a)$$

$$= \eta(\pi) + \sum_s \rho_{\tilde{\pi}}(s) \sum_a \tilde{\pi}(a|s) A_\pi(s, a). \quad (2)$$

This equation implies that any policy update $\pi \to \tilde{\pi}$ that has a nonnegative expected advantage at *every* state s, i.e., $\sum_a \tilde{\pi}(a|s) A_\pi(s, a) \geq 0$, is guaranteed to increase the policy performance η, or leave it constant in the case that the expected advantage is zero everywhere. This implies the classic result that the update performed by exact policy iteration, which uses the deterministic policy

$\tilde{\pi}(s) = \arg\max_a A_\pi(s, a)$, improves the policy if there is at least one state-action pair with a positive advantage value and nonzero state visitation probability, otherwise the algorithm has converged to the optimal policy. However, in the approximate setting, it will typically be unavoidable, due to estimation and approximation error, that there will be some states s for which the expected advantage is negative, that is, $\sum_a \tilde{\pi}(a|s) A_\pi(s, a) < 0$. The complex dependency of $\rho_{\tilde{\pi}}(s)$ on $\tilde{\pi}$ makes Equation (2) difficult to optimize directly. Instead, we introduce the following local approximation to η:

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s, a). \quad (3)$$

Note that L_π uses the visitation frequency ρ_π rather than $\rho_{\tilde{\pi}}$, ignoring changes in state visitation density due to changes in the policy. However, if we have a parameterized policy π_θ, where $\pi_\theta(a|s)$ is a differentiable function of the parameter vector θ, then L_π matches η to first order (see Kakade & Langford (2002)). That is, for any parameter value θ_0,

$$L_{\pi_{\theta_0}}(\pi_{\theta_0}) = \eta(\pi_{\theta_0}),$$

$$\nabla_\theta L_{\pi_{\theta_0}}(\pi_\theta)\big|_{\theta = \theta_0} = \nabla_\theta \eta(\pi_\theta)\big|_{\theta = \theta_0}. \quad (4)$$

Equation (4) implies that a sufficiently small step $\pi_{\theta_0} \to \tilde{\pi}$ that improves $L_{\pi_{\theta_{\text{old}}}}$ will also improve η, but does not give us any guidance on how big of a step to take.

To address this issue, Kakade & Langford (2002) proposed a policy updating scheme called conservative policy iteration, for which they could provide explicit lower bounds on the improvement of η. To define the conservative policy iteration update, let π_{old} denote the current policy, and let $\pi' = \arg\max_{\pi'} L_{\pi_{\text{old}}}(\pi')$. The new policy π_{new} was defined to be the following mixture:

$$\pi_{\text{new}}(a|s) = (1 - \alpha)\pi_{\text{old}}(a|s) + \alpha\pi'(a|s). \quad (5)$$

Kakade and Langford derived the following lower bound:

$$\eta(\pi_{\text{new}}) \geq L_{\pi_{\text{old}}}(\pi_{\text{new}}) - \frac{2\epsilon\gamma}{(1-\gamma)^2}\alpha^2$$

$$\text{where } \epsilon = \max_s \big| \mathbb{E}_{a \sim \pi'(a|s)} [A_\pi(s, a)] \big|. \quad (6)$$

(We have modified it to make it slightly weaker but simpler.) Note, however, that so far this bound only applies to mixture policies generated by Equation (5). This policy class is unwieldy and restrictive in practice, and it is desirable for a practical policy update scheme to be applicable to all general stochastic policy classes.

3 Monotonic Improvement Guarantee for General Stochastic Policies

Equation (6), which applies to conservative policy iteration, implies that a policy update that improves the right-hand

side is guaranteed to improve the true performance η. Our principal theoretical result is that the policy improvement bound in Equation (6) can be extended to general stochastic policies, rather than just mixture polices, by replacing α with a distance measure between π and $\tilde{\pi}$, and changing the constant ϵ appropriately. Since mixture policies are rarely used in practice, this result is crucial for extending the improvement guarantee to practical problems. The particular distance measure we use is the total variation divergence, which is defined by $D_{TV}(p \parallel q) = \frac{1}{2}\sum_i |p_i - q_i|$ for discrete probability distributions p, q.[1] Define $D_{\mathrm{TV}}^{\max}(\pi, \tilde{\pi})$ as

$$D_{\mathrm{TV}}^{\max}(\pi, \tilde{\pi}) = \max_s D_{TV}(\pi(\cdot|s) \parallel \tilde{\pi}(\cdot|s)). \tag{7}$$

Theorem 1. *Let* $\alpha = D_{\mathrm{TV}}^{\max}(\pi_{\mathrm{old}}, \pi_{\mathrm{new}})$. *Then the following bound holds:*

$$\eta(\pi_{\mathrm{new}}) \geq L_{\pi_{\mathrm{old}}}(\pi_{\mathrm{new}}) - \frac{4\epsilon\gamma}{(1-\gamma)^2}\alpha^2$$
$$where\ \epsilon = \max_{s,a}|A_\pi(s,a)| \tag{8}$$

We provide two proofs in the appendix. The first proof extends Kakade and Langford's result using the fact that the random variables from two distributions with total variation divergence less than α can be coupled, so that they are equal with probability $1 - \alpha$. The second proof uses perturbation theory.

Next, we note the following relationship between the total variation divergence and the KL divergence (Pollard (2000), Ch. 3): $D_{TV}(p \parallel q)^2 \leq D_{\mathrm{KL}}(p \parallel q)$. Let $D_{\mathrm{KL}}^{\max}(\pi, \tilde{\pi}) = \max_s D_{\mathrm{KL}}(\pi(\cdot|s) \parallel \tilde{\pi}(\cdot|s))$. The following bound then follows directly from Theorem 1:

$$\eta(\tilde{\pi}) \geq L_\pi(\tilde{\pi}) - CD_{\mathrm{KL}}^{\max}(\pi, \tilde{\pi}),$$
$$where\ C = \frac{4\epsilon\gamma}{(1-\gamma)^2}. \tag{9}$$

Algorithm 1 describes an approximate policy iteration scheme based on the policy improvement bound in Equation (9). Note that for now, we assume exact evaluation of the advantage values A_π.

It follows from Equation (9) that Algorithm 1 is guaranteed to generate a monotonically improving sequence of policies $\eta(\pi_0) \leq \eta(\pi_1) \leq \eta(\pi_2) \leq \ldots$. To see this, let $M_i(\pi) = L_{\pi_i}(\pi) - CD_{\mathrm{KL}}^{\max}(\pi_i, \pi)$. Then

$$\eta(\pi_{i+1}) \geq M_i(\pi_{i+1}) \text{ by Equation (9)}$$
$$\eta(\pi_i) = M_i(\pi_i), \text{ therefore,}$$
$$\eta(\pi_{i+1}) - \eta(\pi_i) \geq M_i(\pi_{i+1}) - M(\pi_i). \tag{10}$$

Thus, by maximizing M_i at each iteration, we guarantee that the true objective η is non-decreasing. This algorithm

[1]Our result is straightforward to extend to continuous states and actions by replacing the sums with integrals.

Algorithm 1 Policy iteration algorithm guaranteeing non-decreasing expected return η

Initialize π_0.
for $i = 0, 1, 2, \ldots$ until convergence **do**
    Compute all advantage values $A_{\pi_i}(s, a)$.
    Solve the constrained optimization problem

$$\pi_{i+1} = \arg\max_\pi \left[L_{\pi_i}(\pi) - CD_{\mathrm{KL}}^{\max}(\pi_i, \pi)\right]$$
$$\text{where } C = 4\epsilon\gamma/(1-\gamma)^2$$
$$\text{and } L_{\pi_i}(\pi) = \eta(\pi_i) + \sum_s \rho_{\pi_i}(s) \sum_a \pi(a|s)A_{\pi_i}(s, a)$$

end for

---

is a type of minorization-maximization (MM) algorithm (Hunter & Lange, 2004), which is a class of methods that also includes expectation maximization. In the terminology of MM algorithms, M_i is the surrogate function that minorizes η with equality at π_i. This algorithm is also reminiscent of proximal gradient methods and mirror descent.

Trust region policy optimization, which we propose in the following section, is an approximation to Algorithm 1, which uses a constraint on the KL divergence rather than a penalty to robustly allow large updates.

4 Optimization of Parameterized Policies

In the previous section, we considered the policy optimization problem independently of the parameterization of π and under the assumption that the policy can be evaluated at all states. We now describe how to derive a practical algorithm from these theoretical foundations, under finite sample counts and arbitrary parameterizations.

Since we consider parameterized policies $\pi_\theta(a|s)$ with parameter vector θ, we will overload our previous notation to use functions of θ rather than π, e.g. $\eta(\theta) := \eta(\pi_\theta)$, $L_\theta(\tilde{\theta}) := L_{\pi_\theta}(\pi_{\tilde{\theta}})$, and $D_{\mathrm{KL}}(\theta \parallel \tilde{\theta}) := D_{\mathrm{KL}}(\pi_\theta \parallel \pi_{\tilde{\theta}})$. We will use θ_{old} to denote the previous policy parameters that we want to improve upon.

The preceding section showed that $\eta(\theta) \geq L_{\theta_{\mathrm{old}}}(\theta) - CD_{\mathrm{KL}}^{\max}(\theta_{\mathrm{old}}, \theta)$, with equality at $\theta = \theta_{\mathrm{old}}$. Thus, by performing the following maximization, we are guaranteed to improve the true objective η:

$$\underset{\theta}{\text{maximize}} \left[L_{\theta_{\mathrm{old}}}(\theta) - CD_{\mathrm{KL}}^{\max}(\theta_{\mathrm{old}}, \theta)\right].$$

In practice, if we used the penalty coefficient C recommended by the theory above, the step sizes would be very small. One way to take larger steps in a robust way is to use a constraint on the KL divergence between the new policy and the old policy, i.e., a trust region constraint:

$$\underset{\theta}{\text{maximize}}\ L_{\theta_{\mathrm{old}}}(\theta) \tag{11}$$
$$\text{subject to } D_{\mathrm{KL}}^{\max}(\theta_{\mathrm{old}}, \theta) \leq \delta.$$

This problem imposes a constraint that the KL divergence is bounded at every point in the state space. While it is motivated by the theory, this problem is impractical to solve due to the large number of constraints. Instead, we can use a heuristic approximation which considers the average KL divergence:

$$\overline{D}_{KL}^{\rho}(\theta_1, \theta_2) := \mathbb{E}_{s \sim \rho}\left[D_{KL}(\pi_{\theta_1}(\cdot|s) \| \pi_{\theta_2}(\cdot|s))\right].$$

We therefore propose solving the following optimization problem to generate a policy update:

$$\underset{\theta}{\text{maximize}}\, L_{\theta_{old}}(\theta) \quad (12)$$

$$\text{subject to } \overline{D}_{KL}^{\rho_{\theta_{old}}}(\theta_{old}, \theta) \leq \delta.$$

Similar policy updates have been proposed in prior work (Bagnell & Schneider, 2003; Peters & Schaal, 2008b; Peters et al., 2010), and we compare our approach to prior methods in Section 7 and in the experiments in Section 8. Our experiments also show that this type of constrained update has similar empirical performance to the maximum KL divergence constraint in Equation (11).

5 Sample-Based Estimation of the Objective and Constraint

The previous section proposed a constrained optimization problem on the policy parameters (Equation (12)), which optimizes an estimate of the expected total reward η subject to a constraint on the change in the policy at each update. This section describes how the objective and constraint functions can be approximated using Monte Carlo simulation.

We seek to solve the following optimization problem, obtained by expanding $L_{\theta_{old}}$ in Equation (12):

$$\underset{\theta}{\text{maximize}} \sum_s \rho_{\theta_{old}}(s) \sum_a \pi_\theta(a|s) A_{\theta_{old}}(s, a)$$

$$\text{subject to } \overline{D}_{KL}^{\rho_{\theta_{old}}}(\theta_{old}, \theta) \leq \delta. \quad (13)$$

We first replace $\sum_s \rho_{\theta_{old}}(s)\,[\ldots]$ in the objective by the expectation $\frac{1}{1-\gamma}\mathbb{E}_{s \sim \rho_{\theta_{old}}}[\ldots]$. Next, we replace the advantage values $A_{\theta_{old}}$ by the Q-values $Q_{\theta_{old}}$ in Equation (13), which only changes the objective by a constant. Last, we replace the sum over the actions by an importance sampling estimator. Using q to denote the sampling distribution, the contribution of a single s_n to the loss function is

$$\sum_a \pi_\theta(a|s_n) A_{\theta_{old}}(s_n, a) = \mathbb{E}_{a \sim q}\left[\frac{\pi_\theta(a|s_n)}{q(a|s_n)} A_{\theta_{old}}(s_n, a)\right].$$

Our optimization problem in Equation (13) is exactly equivalent to the following one, written in terms of expectations:

$$\underset{\theta}{\text{maximize}}\, \mathbb{E}_{s \sim \rho_{\theta_{old}}, a \sim q}\left[\frac{\pi_\theta(a|s)}{q(a|s)} Q_{\theta_{old}}(s, a)\right] \quad (14)$$

$$\text{subject to } \mathbb{E}_{s \sim \rho_{\theta_{old}}}\left[D_{KL}(\pi_{\theta_{old}}(\cdot|s) \| \pi_\theta(\cdot|s))\right] \leq \delta.$$
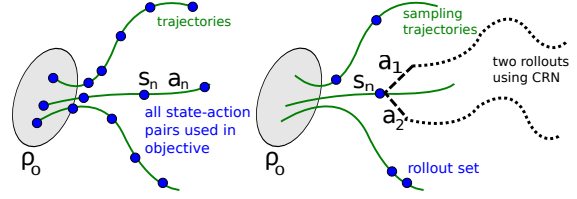

Figure 1. Left: illustration of single path procedure. Here, we generate a set of trajectories via simulation of the policy and incorporate all state-action pairs (s_n, a_n) into the objective. Right: illustration of vine procedure. We generate a set of "trunk" trajectories, and then generate "branch" rollouts from a subset of the reached states. For each of these states s_n, we perform multiple actions (a_1 and a_2 here) and perform a rollout after each action, using common random numbers (CRN) to reduce the variance.

All that remains is to replace the expectations by sample averages and replace the Q value by an empirical estimate. The following sections describe two different schemes for performing this estimation.

The first sampling scheme, which we call *single path*, is the one that is typically used for policy gradient estimation (Bartlett & Baxter, 2011), and is based on sampling individual trajectories. The second scheme, which we call *vine*, involves constructing a rollout set and then performing multiple actions from each state in the rollout set. This method has mostly been explored in the context of policy iteration methods (Lagoudakis & Parr, 2003; Gabillon et al., 2013).

5.1 Single Path

In this estimation procedure, we collect a sequence of states by sampling $s_0 \sim \rho_0$ and then simulating the policy $\pi_{\theta_{old}}$ for some number of timesteps to generate a trajectory $s_0, a_0, s_1, a_1, \ldots, s_{T-1}, a_{T-1}, s_T$. Hence, $q(a|s) = \pi_{\theta_{old}}(a|s)$. $Q_{\theta_{old}}(s, a)$ is computed at each state-action pair (s_t, a_t) by taking the discounted sum of future rewards along the trajectory.

5.2 Vine

In this estimation procedure, we first sample $s_0 \sim \rho_0$ and simulate the policy π_{θ_i} to generate a number of trajectories. We then choose a subset of N states along these trajectories, denoted $s_1, s_2, \ldots, s_N$, which we call the "rollout set". For each state s_n in the rollout set, we sample K actions according to $a_{n,k} \sim q(\cdot|s_n)$. Any choice of $q(\cdot|s_n)$ with a support that includes the support of $\pi_{\theta_i}(\cdot|s_n)$ will produce a consistent estimator. In practice, we found that $q(\cdot|s_n) = \pi_{\theta_i}(\cdot|s_n)$ works well on continuous problems, such as robotic locomotion, while the uniform distribution works well on discrete tasks, such as the Atari games, where it can sometimes achieve better exploration.

For each action $a_{n,k}$ sampled at each state s_n, we esti-

mate $\hat{Q}_{\theta_i}(s_n, a_{n,k})$ by performing a rollout (i.e., a short trajectory) starting with state s_n and action $a_{n,k}$. We can greatly reduce the variance of the Q-value differences between rollouts by using the same random number sequence for the noise in each of the K rollouts, i.e., *common random numbers*. See (Bertsekas, 2005) for additional discussion on Monte Carlo estimation of Q-values and (Ng & Jordan, 2000) for a discussion of common random numbers in reinforcement learning.

In small, finite action spaces, we can generate a rollout for every possible action from a given state. The contribution to $L_{\theta_{\text{old}}}$ from a single state s_n is as follows:

$$L_n(\theta) = \sum_{k=1}^{K} \pi_\theta(a_k|s_n)\hat{Q}(s_n, a_k), \qquad (15)$$

where the action space is $\mathcal{A} = \{a_1, a_2, \ldots, a_K\}$. In large or continuous state spaces, we can construct an estimator of the surrogate objective using importance sampling. The self-normalized estimator (Owen (2013), Chapter 9) of $L_{\theta_{\text{old}}}$ obtained at a single state s_n is

$$L_n(\theta) = \frac{\sum_{k=1}^{K} \frac{\pi_\theta(a_{n,k}|s_n)}{\pi_{\theta_{\text{old}}}(a_{n,k}|s_n)}\hat{Q}(s_n, a_{n,k})}{\sum_{k=1}^{K} \frac{\pi_\theta(a_{n,k}|s_n)}{\pi_{\theta_{\text{old}}}(a_{n,k}|s_n)}}, \qquad (16)$$

assuming that we performed K actions $a_{n,1}, a_{n,2}, \ldots, a_{n,K}$ from state s_n. This self-normalized estimator removes the need to use a baseline for the Q-values (note that the gradient is unchanged by adding a constant to the Q-values). Averaging over $s_n \sim \rho(\pi)$, we obtain an estimator for $L_{\theta_{\text{old}}}$, as well as its gradient.

The *vine* and *single path* methods are illustrated in Figure 1. We use the term *vine*, since the trajectories used for sampling can be likened to the stems of vines, which branch at various points (the rollout set) into several short offshoots (the rollout trajectories).

The benefit of the *vine* method over the *single path* method that is our local estimate of the objective has much lower variance given the same number of Q-value samples in the surrogate objective. That is, the *vine* method gives much better estimates of the advantage values. The downside of the *vine* method is that we must perform far more calls to the simulator for each of these advantage estimates. Furthermore, the *vine* method requires us to generate multiple trajectories from each state in the rollout set, which limits this algorithm to settings where the system can be reset to an arbitrary state. In contrast, the single path algorithm requires no state resets and can be directly implemented on a physical system (Peters & Schaal, 2008b).

6 Practical Algorithm

Here we present two practical policy optimization algorithm based on the ideas above, which use either the *single path* or *vine* sampling scheme from the preceding section. The algorithms repeatedly perform the following steps:

1. Use the *single path* or *vine* procedures to collect a set of state-action pairs along with Monte Carlo estimates of their Q-values.

2. By averaging over samples, construct the estimated objective and constraint in Equation (14).

3. Approximately solve this constrained optimization problem to update the policy's parameter vector θ. We use the conjugate gradient algorithm followed by a line search, which is altogether only slightly more expensive than computing the gradient itself. See Appendix C for details.

With regard to (3), we construct the Fisher information matrix (FIM) by analytically computing the Hessian of the KL divergence, rather than using the covariance matrix of the gradients. That is, we estimate A_{ij} as $\frac{1}{N}\sum_{n=1}^{N} \frac{\partial^2}{\partial\theta_i \partial\theta_j} D_{\text{KL}}(\pi_{\theta_{\text{old}}}(\cdot|s_n) \,\|\, \pi_\theta(\cdot|s_n))$, rather than $\frac{1}{N}\sum_{n=1}^{N} \frac{\partial}{\partial\theta_i}\log\pi_\theta(a_n|s_n)\frac{\partial}{\partial\theta_j}\log\pi_\theta(a_n|s_n)$. The analytic estimator integrates over the action at each state s_n, and does not depend on the action a_n that was sampled. As described in Appendix C, this analytic estimator has computational benefits in the large-scale setting, since it removes the need to store a dense Hessian or all policy gradients from a batch of trajectories. The rate of improvement in the policy is similar to the empirical FIM, as shown in the experiments.

Let us briefly summarize the relationship between the theory from Section 3 and the practical algorithm we have described:

- The theory justifies optimizing a surrogate objective with a penalty on KL divergence. However, the large penalty coefficient C leads to prohibitively small steps, so we would like to decrease this coefficient. Empirically, it is hard to robustly choose the penalty coefficient, so we use a hard constraint instead of a penalty, with parameter δ (the bound on KL divergence).

- The constraint on $D_{\text{KL}}^{\max}(\theta_{\text{old}}, \theta)$ is hard for numerical optimization and estimation, so instead we constrain $\overline{D}_{\text{KL}}(\theta_{\text{old}}, \theta)$.

- Our theory ignores estimation error for the advantage function. Kakade & Langford (2002) consider this error in their derivation, and the same arguments would hold in the setting of this paper, but we omit them for simplicity.

7 Connections with Prior Work

As mentioned in Section 4, our derivation results in a policy update that is related to several prior methods, providing a unifying perspective on a number of policy update

schemes. The natural policy gradient (Kakade, 2002) can be obtained as a special case of the update in Equation (12) by using a linear approximation to L and a quadratic approximation to the $\bar{D}_{\mathrm{KL}}$ constraint, resulting in the following problem:

$$\underset{\theta}{\text{maximize}} \left[\nabla_\theta L_{\theta_{\text{old}}}(\theta) \big|_{\theta=\theta_{\text{old}}} \cdot (\theta - \theta_{\text{old}}) \right] \quad (17)$$

$$\text{subject to } \frac{1}{2}(\theta_{\text{old}} - \theta)^T A(\theta_{\text{old}})(\theta_{\text{old}} - \theta) \leq \delta,$$

$$\text{where } A(\theta_{\text{old}})_{ij} =$$

$$\frac{\partial}{\partial \theta_i} \frac{\partial}{\partial \theta_j} \mathbb{E}_{s \sim \rho_\pi} \left[D_{\mathrm{KL}}(\pi(\cdot|s, \theta_{\text{old}}) \| \pi(\cdot|s, \theta)) \right] \big|_{\theta=\theta_{\text{old}}}.$$

The update is $\theta_{\text{new}} = \theta_{\text{old}} + \frac{1}{\lambda} A(\theta_{\text{old}})^{-1} \nabla_\theta L(\theta) \big|_{\theta=\theta_{\text{old}}}$, where the stepsize $\frac{1}{\lambda}$ is typically treated as an algorithm parameter. This differs from our approach, which enforces the constraint at each update. Though this difference might seem subtle, our experiments demonstrate that it significantly improves the algorithm's performance on larger problems.

We can also obtain the standard policy gradient update by using an ℓ_2 constraint or penalty:

$$\underset{\theta}{\text{maximize}} \left[\nabla_\theta L_{\theta_{\text{old}}}(\theta) \big|_{\theta=\theta_{\text{old}}} \cdot (\theta - \theta_{\text{old}}) \right] \quad (18)$$

$$\text{subject to } \frac{1}{2} \|\theta - \theta_{\text{old}}\|^2 \leq \delta.$$

The policy iteration update can also be obtained by solving the unconstrained problem $\text{maximize}_\pi L_{\pi_{\text{old}}}(\pi)$, using L as defined in Equation (3).

Several other methods employ an update similar to Equation (12). *Relative entropy policy search* (REPS) (Peters et al., 2010) constrains the state-action marginals $p(s, a)$, while TRPO constrains the conditionals $p(a|s)$. Unlike REPS, our approach does not require a costly nonlinear optimization in the inner loop. Levine and Abbeel (2014) also use a KL divergence constraint, but its purpose is to encourage the policy not to stray from regions where the estimated dynamics model is valid, while we do not attempt to estimate the system dynamics explicitly. Pirotta et al. (2013) also build on and generalize Kakade and Langford's results, and they derive different algorithms from the ones here.

8 Experiments

We designed our experiments to investigate the following questions:

1. What are the performance characteristics of the *single path* and *vine* sampling procedures?

2. TRPO is related to prior methods (e.g. natural policy gradient) but makes several changes, most notably by using a fixed KL divergence rather than a fixed penalty coefficient. How does this affect the performance of the algorithm?
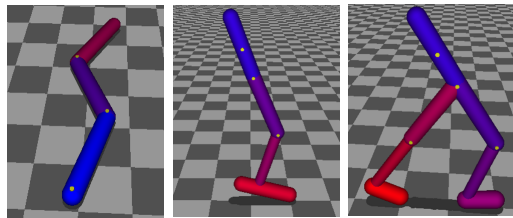

Figure 2. 2D robot models used for locomotion experiments. From left to right: swimmer, hopper, walker. The hopper and walker present a particular challenge, due to underactuation and contact discontinuities.
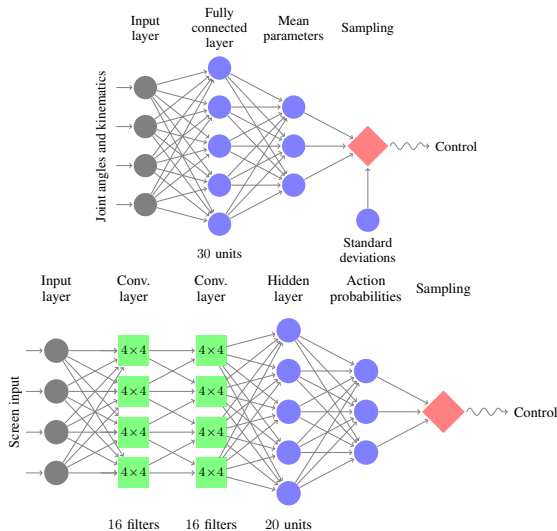

Figure 3. Neural networks used for the locomotion task (top) and for playing Atari games (bottom).

3. Can TRPO be used to solve challenging large-scale problems? How does TRPO compare with other methods when applied to large-scale problems, with regard to final performance, computation time, and sample complexity?

To answer (1) and (2), we compare the performance of the *single path* and *vine* variants of TRPO, several ablated variants, and a number of prior policy optimization algorithms. With regard to (3), we show that both the *single path* and *vine* algorithm can obtain high-quality locomotion controllers from scratch, which is considered to be a hard problem. We also show that these algorithms produce competitive results when learning policies for playing Atari games from images using convolutional neural networks with tens of thousands of parameters.

8.1 Simulated Robotic Locomotion

We conducted the robotic locomotion experiments using the MuJoCo simulator (Todorov et al., 2012). The three simulated robots are shown in Figure 2. The states of the robots are their generalized positions and velocities, and the controls are joint torques. Underactuation, high dimensionality, and non-smooth dynamics due to contacts make these

tasks very challenging. The following models are included in our evaluation:

1. *Swimmer*. 10-dimensional state space, linear reward for forward progress and a quadratic penalty on joint effort to produce the reward $r(x, u) = v_x - 10^{-5}\|u\|^2$. The swimmer can propel itself forward by making an undulating motion.

2. *Hopper*. 12-dimensional state space, same reward as the swimmer, with a bonus of $+1$ for being in a non-terminal state. We ended the episodes when the hopper fell over, which was defined by thresholds on the torso height and angle.

3. *Walker*. 18-dimensional state space. For the walker, we added a penalty for strong impacts of the feet against the ground to encourage a smooth walk rather than a hopping gait.

We used $\delta = 0.01$ for all experiments. See Table 2 in the Appendix for more details on the experimental setup and parameters used. We used neural networks to represent the policy, with the architecture shown in Figure 3, and further details provided in Appendix D. To establish a standard baseline, we also included the classic cart-pole balancing problem, based on the formulation from Barto et al. (1983), using a linear policy with six parameters that is easy to optimize with derivative-free black-box optimization methods.

The following algorithms were considered in the comparison: *single path TRPO*; *vine TRPO*; *cross-entropy method* (CEM), a gradient-free method (Szita & Lörincz, 2006); *covariance matrix adaption* (CMA), another gradient-free method (Hansen & Ostermeier, 1996); *natural gradient*, the classic natural policy gradient algorithm (Kakade, 2002), which differs from *single path* by the use of a fixed penalty coefficient (Lagrange multiplier) instead of the KL divergence constraint; *empirical FIM*, identical to *single path*, except that the FIM is estimated using the covariance matrix of the gradients rather than the analytic estimate; *max KL*, which was only tractable on the cart-pole problem, and uses the maximum KL divergence in Equation (11), rather than the average divergence, allowing us to evaluate the quality of this approximation. The parameters used in the experiments are provided in Appendix E. For the *natural gradient* method, we swept through the possible values of the stepsize in factors of three, and took the best value according to the final performance.

Learning curves showing the total reward averaged across five runs of each algorithm are shown in Figure 4. *Single path* and *vine* TRPO solved all of the problems, yielding the best solutions. *Natural gradient* performed well on the two easier problems, but was unable to generate hopping and walking gaits that made forward progress. These results provide empirical evidence that constraining the KL divergence is a more robust way to choose step sizes and make fast, consistent progress, compared to using a fixed
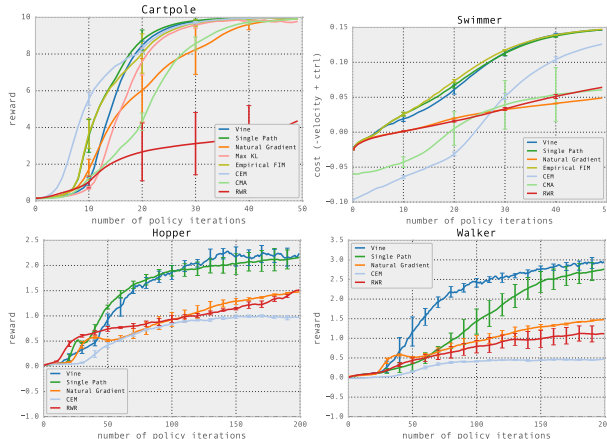

Figure 4. Learning curves for locomotion tasks, averaged across five runs of each algorithm with random initializations. Note that for the hopper and walker, a score of -1 is achievable without any forward velocity, indicating a policy that simply learned balanced standing, but not walking.

penalty. CEM and CMA are derivative-free algorithms, hence their sample complexity scales unfavorably with the number of parameters, and they performed poorly on the larger problems. The *max KL* method learned somewhat more slowly than our final method, due to the more restrictive form of the constraint, but overall the result suggests that the average KL divergence constraint has a similar effect as the theorecally justified maximum KL divergence. Videos of the policies learned by TRPO may be viewed on the project website: `http://sites.google.com/site/trpopaper/`.

Note that TRPO learned all of the gaits with general-purpose policies and simple reward functions, using minimal prior knowledge. This is in contrast with most prior methods for learning locomotion, which typically rely on hand-architected policy classes that explicitly encode notions of balance and stepping (Tedrake et al., 2004; Geng et al., 2006; Wampler & Popović, 2009).

8.2 Playing Games from Images

To evaluate TRPO on a partially observed task with complex observations, we trained policies for playing Atari games, using raw images as input. The games require learning a variety of behaviors, such as dodging bullets and hitting balls with paddles. Aside from the high dimensionality, challenging elements of these games include delayed rewards (no immediate penalty is incurred when a life is lost in Breakout or Space Invaders); complex sequences of behavior (Q*bert requires a character to hop on 21 different platforms); and non-stationary image statistics (Enduro involves a changing and flickering background).

We tested our algorithms on the same seven games reported on in (Mnih et al., 2013) and (Guo et al., 2014), which are

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Human (Mnih et al., 2013)	7456	31.0	368	−3.0	18900	28010	3690
Deep Q Learning (Mnih et al., 2013)	4092	168.0	470	20.0	1952	1705	581
UCC-I (Guo et al., 2014)	5702	380	741	21	20025	2995	692
TRPO - single path	1425.2	10.8	534.6	20.9	1973.5	1908.6	568.4
TRPO - vine	859.5	34.2	430.8	20.9	7732.5	788.4	450.2

Table 1. Performance comparison for vision-based RL algorithms on the Atari domain. Our algorithms (bottom rows) were run once on each task, with the same architecture and parameters. Performance varies substantially from run to run (with different random initializations of the policy), but we could not obtain error statistics due to time constraints.

made available through the Arcade Learning Environment (Bellemare et al., 2013) The images were preprocessed following the protocol in Mnih et al (2013), and the policy was represented by the convolutional neural network shown in Figure 3, with two convolutional layers with 16 channels and stride 2, followed by one fully-connected layer with 20 units, yielding 33,500 parameters.

The results of the *vine* and *single path* algorithms are summarized in Table 1, which also includes an expert human performance and two recent methods: deep Q-learning (Mnih et al., 2013), and a combination of Monte-Carlo Tree Search with supervised training (Guo et al., 2014), called UCC-I. The 500 iterations of our algorithm took about 30 hours (with slight variation between games) on a 16-core computer. While our method only outperformed the prior methods on some of the games, it consistently achieved reasonable scores. Unlike the prior methods, our approach was not designed specifically for this task. The ability to apply the same policy search method to methods as diverse as robotic locomotion and image-based game playing demonstrates the generality of TRPO.

9 Discussion

We proposed and analyzed trust region methods for optimizing stochastic control policies. We proved monotonic improvement for an algorithm that repeatedly optimizes a local approximation to the expected return of the policy with a KL divergence penalty, and we showed that an approximation to this method that incorporates a KL divergence constraint achieves good empirical results on a range of challenging policy learning tasks, outperforming prior methods. Our analysis also provides a perspective that unifies policy gradient and policy iteration methods, and shows them to be special limiting cases of an algorithm that optimizes a certain objective subject to a trust region constraint.

In the domain of robotic locomotion, we successfully learned controllers for swimming, walking and hopping in a physics simulator, using general purpose neural networks and minimally informative rewards. To our knowledge, no prior work has learned controllers from scratch for all of these tasks, using a generic policy search method and non-engineered, general-purpose policy representations. In

the game-playing domain, we learned convolutional neural network policies that used raw images as inputs. This requires optimizing extremely high-dimensional policies, and only two prior methods report successful results on this task.

Since the method we proposed is scalable and has strong theoretical foundations, we hope that it will serve as a jumping-off point for future work on training large, rich function approximators for a range of challenging problems. At the intersection of the two experimental domains we explored, there is the possibility of learning robotic control policies that use vision and raw sensory data as input, providing a unified scheme for training robotic controllers that perform both perception and control. The use of more sophisticated policies, including recurrent policies with hidden state, could further make it possible to roll state estimation and control into the same policy in the partially-observed setting. By combining our method with model learning, it would also be possible to substantially reduce its sample complexity, making it applicable to real-world settings where samples are expensive.

Acknowledgements

References

Bagnell, J. A. and Schneider, J. Covariant policy search. IJCAI, 2003.

Bartlett, P. L. and Baxter, J. Infinite-horizon policy-gradient estimation. *arXiv preprint arXiv:1106.0665*, 2011.

Barto, A., Sutton, R., and Anderson, C. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics*, (5):834–846, 1983.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

Bertsekas, D. *Dynamic programming and optimal control*, volume 1. 2005.

Deisenroth, M., Neumann, G., and Peters, J. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142, 2013.

Gabillon, Victor, Ghavamzadeh, Mohammad, and Scherrer, Bruno. Approximate dynamic programming finally performs well in the game of Tetris. In *Advances in Neural Information Processing Systems*, 2013.

Geng, T., Porr, B., and Wörgötter, F. Fast biped walking with a reflexive controller and realtime policy searching. In *Advances in Neural Information Processing Systems (NIPS)*, 2006.

Guo, X., Singh, S., Lee, H., Lewis, R. L., and Wang, X. Deep learning for real-time atari game play using offline Monte-Carlo tree search planning. In *Advances in Neural Information Processing Systems*, pp. 3338–3346, 2014.

Hansen, Nikolaus and Ostermeier, Andreas. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pp. 312–317. IEEE, 1996.

Hunter, David R and Lange, Kenneth. A tutorial on MM algorithms. *The American Statistician*, 58(1):30–37, 2004.

Kakade, Sham. A natural policy gradient. In *Advances in Neural Information Processing Systems*, pp. 1057–1063. MIT Press, 2002.

Kakade, Sham and Langford, John. Approximately optimal approximate reinforcement learning. In *ICML*, volume 2, pp. 267–274, 2002.

Lagoudakis, Michail G and Parr, Ronald. Reinforcement learning as classification: Leveraging modern classifiers. In *ICML*, volume 3, pp. 424–431, 2003.

Levin, D. A., Peres, Y., and Wilmer, E. L. *Markov chains and mixing times*. American Mathematical Society, 2009.

Levine, Sergey and Abbeel, Pieter. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*, pp. 1071–1079, 2014.

Martens, J. and Sutskever, I. Training deep and recurrent networks with hessian-free optimization. In *Neural Networks: Tricks of the Trade*, pp. 479–535. Springer, 2012.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Nemirovski, Arkadi. Efficient methods in convex programming. 2005.

Ng, A. Y. and Jordan, M. PEGASUS: A policy search method for large mdps and pomdps. In *Uncertainty in artificial intelligence (UAI)*, 2000.

Owen, Art B. *Monte Carlo theory, methods and examples*. 2013.

Pascanu, Razvan and Bengio, Yoshua. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584*, 2013.

Peters, J. and Schaal, S. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697, 2008a.

Peters, J., Mülling, K., and Altün, Y. Relative entropy policy search. In *AAAI Conference on Artificial Intelligence*, 2010.

Peters, Jan and Schaal, Stefan. Natural actor-critic. *Neurocomputing*, 71(7):1180–1190, 2008b.

Pirotta, Matteo, Restelli, Marcello, Pecorino, Alessio, and Calandriello, Daniele. Safe policy iteration. In *Proceedings of The 30th International Conference on Machine Learning*, pp. 307–315, 2013.

Pollard, David. *Asymptopia: an exposition of statistical asymptotic theory*. 2000. URL http://www.stat.yale.edu/~pollard/Books/Asymptopia.

Szita, István and Lörincz, András. Learning tetris using the noisy cross-entropy method. *Neural computation*, 18(12):2936–2941, 2006.

Tedrake, R., Zhang, T., and Seung, H. Stochastic policy gradient reinforcement learning on a simple 3d biped. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.

Todorov, Emanuel, Erez, Tom, and Tassa, Yuval. MuJoCo: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 5026–5033. IEEE, 2012.

Wampler, Kevin and Popović, Zoran. Optimal gait and form for animal locomotion. In *ACM Transactions on Graphics (TOG)*, volume 28, pp. 60. ACM, 2009.

Wright, Stephen J and Nocedal, Jorge. *Numerical optimization*, volume 2. Springer New York, 1999.

A Proof of Policy Improvement Bound

This proof (of Theorem 1) uses techniques from the proof of Theorem 4.1 in (Kakade & Langford, 2002), adapting them to the more general setting considered in this paper. An informal overview is as follows. Our proof relies on the notion of coupling, where we jointly define the policies π and π' so that they choose the same action with high probability $= (1 - \alpha)$. Surrogate loss $L_\pi(\tilde{\pi})$ accounts for the the advantage of $\tilde{\pi}$ the first time that it disagrees with π, but not subsequent disagreements. Hence, the error in L_π is due to two or more disagreements between π and $\tilde{\pi}$, hence, we get an $O(\alpha^2)$ correction term, where α is the probability of disagreement.

We start out with a lemma from Kakade & Langford (2002) that shows that the difference in policy performance $\eta(\tilde{\pi}) - \eta(\pi)$ can be decomposed as a sum of per-timestep advantages.

Lemma 1. *Given two policies $\pi, \tilde{\pi}$,*

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{\tau \sim \tilde{\pi}} \left[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t) \right]$$

(19)

This expectation is taken over trajectories $\tau := (s_0, a_0, s_1, a_0, \dots)$, and the notation $\mathbb{E}_{\tau \sim \tilde{\pi}}[\dots]$ indicates that actions are sampled from $\tilde{\pi}$ to generate τ.

Proof. First note that $A_\pi(s, a) = \mathbb{E}_{s' \sim P(s'|s,a)}[r(s) + \gamma V_\pi(s') - V_\pi(s)]$. Therefore,

$$\mathbb{E}_{\tau|\tilde{\pi}} \left[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t) \right]$$

(20)

$$= \mathbb{E}_{\tau|\tilde{\pi}} \left[\sum_{t=0}^{\infty} \gamma^t (r(s_t) + \gamma V_\pi(s_{t+1}) - V_\pi(s_t)) \right]$$

(21)

$$= \mathbb{E}_{\tau|\tilde{\pi}} \left[-V_\pi(s_0) + \sum_{t=0}^{\infty} \gamma^t r(s_t) \right]$$

(22)

$$= -\mathbb{E}_{s_0} [V_\pi(s_0)] + \mathbb{E}_{\tau|\tilde{\pi}} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t) \right]$$

(23)

$$= -\eta(\pi) + \eta(\tilde{\pi})$$

(24)

Rearranging, the result follows. $\square$

Define $\bar{A}(s)$ to be the expected advantage of $\tilde{\pi}$ over π at state s:

$$\bar{A}(s) = \mathbb{E}_{a \sim \tilde{\pi}(\cdot|s)}[A_\pi(s, a)].$$

(25)

Now Lemma 1 can be written as follows:

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{\tau \sim \tilde{\pi}} \left[\sum_{t=0}^{\infty} \gamma^t \bar{A}(s_t) \right]$$

(26)

Note that L_π can be written as

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t \bar{A}(s_t) \right]$$

(27)

The difference in these equations is whether the states are sampled using π or $\tilde{\pi}$. To bound the difference between $\eta(\tilde{\pi})$ and $L_\pi(\tilde{\pi})$, we will bound the difference arising from each timestep. To do this, we first need to introduce a measure of how much π and $\tilde{\pi}$ agree. Specifically, we'll *couple* the policies, so that they define a joint distribution over pairs of actions.

Definition 1. *$(\pi, \tilde{\pi})$ is an α-coupled policy pair if it defines a joint distribution $(a, \tilde{a})|s$, such that $P(a \neq \tilde{a}|s) \leq \alpha$ for all s. π and $\tilde{\pi}$ will denote the marginal distributions of a and $\tilde{a}$, respectively.*

Computationally, α-coupling means that if we randomly choose a seed for our random number generator, and then we sample from each of π and $\tilde{\pi}$ after setting that seed, the results will agree for at least fraction $1 - \alpha$ of seeds.

Lemma 2. *Given that $\pi, \tilde{\pi}$ are α-coupled policies, for all s,*

$$\left| \bar{A}(s) \right| \leq 2\alpha \max_{s,a} |A_\pi(s, a)| \tag{28}$$

Proof.

$$\bar{A}(s) = \mathbb{E}_{\tilde{a} \sim \tilde{\pi}} \left[A_\pi(s, \tilde{a}) \right] = \mathbb{E}_{(a,\tilde{a}) \sim (\pi, \tilde{\pi})} \left[A_\pi(s, \tilde{a}) - A_\pi(s, a) \right] \quad \text{since} \quad \mathbb{E}_{a \sim \pi} \left[A_\pi(s, a) \right] = 0 \tag{29}$$

$$= P(a \neq \tilde{a} | s) \mathbb{E}_{(a,\tilde{a}) \sim (\pi, \tilde{\pi}) | a \neq \tilde{a}} \left[A_\pi(s, \tilde{a}) - A_\pi(s, a) \right] \tag{30}$$

$$|\bar{A}(s)| \leq \alpha \cdot 2 \max_{s,a} |A_\pi(s, a)| \tag{31}$$

$\square$

Lemma 3. *Let $(\pi, \tilde{\pi})$ be an α-coupled policy pair. Then*

$$\left| \mathbb{E}_{s_t \sim \tilde{\pi}} \left[\bar{A}(s_t) \right] - \mathbb{E}_{s_t \sim \pi} \left[\bar{A}(s_t) \right] \right| \leq 2\alpha \max_s \bar{A}(s) \leq 4\alpha (1 - (1 - \alpha)^t) \max_s |A_\pi(s, a)| \tag{32}$$

Proof. Given the coupled policy pair $(\pi, \tilde{\pi})$, we can also obtain a coupling over the trajectory distributions produced by π and $\tilde{\pi}$, respectively. Namely, we have pairs of trajectories $\tau, \tilde{\tau}$, where τ is obtained by taking actions from π, and $\tilde{\tau}$ is obtained by taking actions from $\tilde{\pi}$, where the same random seed is used to generate both trajectories. We will consider the advantage of $\tilde{\pi}$ over π at timestep t, and decompose this expectation based on whether π agrees with $\tilde{\pi}$ at all timesteps $i < t$.

Let n_t denote the number of times that $a_i \neq \tilde{a}_i$ for $i < t$, i.e., the number of times that π and $\tilde{\pi}$ disagree before timestep t.

$$\mathbb{E}_{s_t \sim \tilde{\pi}} \left[\bar{A}(s_t) \right] = P(n_t = 0) \mathbb{E}_{s_t \sim \tilde{\pi} | n_t = 0} \left[\bar{A}(s_t) \right] + P(n_t > 0) \mathbb{E}_{s_t \sim \tilde{\pi} | n_t > 0} \left[\bar{A}(s_t) \right] \tag{33}$$

The expectation decomposes similarly for actions are sampled using π:

$$\mathbb{E}_{s_t \sim \pi} \left[\bar{A}(s_t) \right] = P(n_t = 0) \mathbb{E}_{s_t \sim \pi | n_t = 0} \left[\bar{A}(s_t) \right] + P(n_t > 0) \mathbb{E}_{s_t \sim \pi | n_t > 0} \left[\bar{A}(s_t) \right] \tag{34}$$

Note that the $n_t = 0$ terms are equal:

$$\mathbb{E}_{s_t \sim \tilde{\pi} | n_t = 0} \left[\bar{A}(s_t) \right] = \mathbb{E}_{s_t \sim \pi | n_t = 0} \left[\bar{A}(s_t) \right], \tag{35}$$

because $n_t = 0$ indicates that π and $\tilde{\pi}$ agreed on all timesteps less than t. Subtracting Equations (33) and (34), we get

$$\mathbb{E}_{s_t \sim \tilde{\pi}} \left[\bar{A}(s_t) \right] - \mathbb{E}_{s_t \sim \pi} \left[\bar{A}(s_t) \right] = P(n_t > 0) \left(\mathbb{E}_{s_t \sim \tilde{\pi} | n_t > 0} \left[\bar{A}(s_t) \right] - \mathbb{E}_{s_t \sim \pi | n_t > 0} \left[\bar{A}(s_t) \right] \right) \tag{36}$$

By definition of α, $P(\pi, \tilde{\pi}$ agree at timestep $i) \geq 1 - \alpha$, so $P(n_t = 0) \geq (1 - \alpha)^t$, and

$$P(n_t > 0) \leq 1 - (1 - \alpha)^t \tag{37}$$

Next, note that

$$\left| \mathbb{E}_{s_t \sim \tilde{\pi} | n_t > 0} \left[\bar{A}(s_t) \right] - \mathbb{E}_{s_t \sim \pi | n_t > 0} \left[\bar{A}(s_t) \right] \right| \leq \left| \mathbb{E}_{s_t \sim \tilde{\pi} | n_t > 0} \left[\bar{A}(s_t) \right] \right| + \left| \mathbb{E}_{s_t \sim \pi | n_t > 0} \left[\bar{A}(s_t) \right] \right| \tag{38}$$

$$\leq 4\alpha \max_{s,a} |A_\pi(s, a)| \tag{39}$$

Where the second inequality follows from Lemma 3.

Plugging Equation (37) and Equation (39) into Equation (36), we get

$$\left| \mathbb{E}_{s_t \sim \tilde{\pi}} \left[\bar{A}(s_t) \right] - \mathbb{E}_{s_t \sim \pi} \left[\bar{A}(s_t) \right] \right| \leq 4\alpha (1 - (1 - \alpha)^t) \max_{s,a} |A_\pi(s, a)| \tag{40}$$

$\square$

The preceding Lemma bounds the difference in expected advantage at each timestep t. We can sum over time to bound the difference between $\eta(\tilde{\pi})$ and $L_\pi(\tilde{\pi})$. Subtracting Equation (26) and Equation (27), and defining $\epsilon = \max_{s,a} |A_\pi(s,a)|$,

$$|\eta(\tilde{\pi}) - L_\pi(\tilde{\pi})| = \sum_{t=0}^{\infty} \gamma^t \left| \mathbb{E}_{\tau \sim \tilde{\pi}} \left[\bar{A}(s_t) \right] - \mathbb{E}_{\tau \sim \pi} \left[\bar{A}(s_t) \right] \right| \tag{41}$$

$$\leq \sum_{t=0}^{\infty} \gamma^t \cdot 4\epsilon\alpha(1 - (1-\alpha)^t) \tag{42}$$

$$= 4\epsilon\alpha \left(\frac{1}{1-\gamma} - \frac{1}{1-\gamma(1-\alpha)} \right) \tag{43}$$

$$= \frac{4\alpha^2 \gamma \epsilon}{(1-\gamma)(1-\gamma(1-\alpha))} \tag{44}$$

$$\leq \frac{4\alpha^2 \gamma \epsilon}{(1-\gamma)^2} \tag{45}$$

Last, to replace α by the total variation divergence, we need to use the correspondence between TV divergence and coupled random variables:

> Suppose p_X and p_Y are distributions with $D_{TV}(p_X \| p_Y) = \alpha$. Then there exists a joint distribution (X, Y) whose marginals are p_X, p_Y, for which $X = Y$ with probability $1 - \alpha$.

See (Levin et al., 2009), Proposition 4.7.

It follows that if we have two policies π and $\tilde{\pi}$ such that $\max_s D_{TV}(\pi(\cdot|s) \| \tilde{\pi}(\cdot|s)) \leq \alpha$, then we can define an α-coupled policy pair $(\pi, \tilde{\pi})$ with appropriate marginals. Taking $\alpha = \max_s D_{TV}(\pi(\cdot|s) \| \tilde{\pi}(\cdot|s)) \leq \alpha$ in Equation (45), Theorem 1 follows.

B Perturbation Theory Proof of Policy Improvement Bound

We also provide an alternative proof of Theorem 1 using perturbation theory.

Proof. Let $G = (1+\gamma P_\pi + (\gamma P_\pi)^2 + \dots) = (1-\gamma P_\pi)^{-1}$, and similarly Let $\tilde{G} = (1+\gamma P_{\tilde{\pi}} + (\gamma P_{\tilde{\pi}})^2 + \dots) = (1-\gamma P_{\tilde{\pi}})^{-1}$. We will use the convention that ρ (a density on state space) is a vector and r (a reward function on state space) is a dual vector (i.e., linear functional on vectors), thus $r\rho$ is a scalar meaning the expected reward under density ρ. Note that $\eta(\pi) = rG\rho_0$, and $\eta(\tilde{\pi}) = c\tilde{G}\rho_0$. Let $\Delta = P_{\tilde{\pi}} - P_\pi$. We want to bound $\eta(\tilde{\pi}) - \eta(\pi) = r(\tilde{G} - G)\rho_0$. We start with some standard perturbation theory manipulations.

$$G^{-1} - \tilde{G}^{-1} = (1 - \gamma P_\pi) - (1 - \gamma P_{\tilde{\pi}})$$
$$= \gamma \Delta. \tag{46}$$

Left multiply by G and right multiply by $\tilde{G}$.

$$\tilde{G} - G = \gamma G \Delta \tilde{G}$$
$$\tilde{G} = G + \gamma G \Delta \tilde{G} \tag{47}$$

Substituting the right-hand side into $\tilde{G}$ gives

$$\tilde{G} = G + \gamma G \Delta G + \gamma^2 G \Delta G \Delta \tilde{G} \tag{48}$$

So we have

$$\eta(\tilde{\pi}) - \eta(\pi) = r(\tilde{G} - G)\rho = \gamma r G \Delta G \rho_0 + \gamma^2 r G \Delta G \Delta \tilde{G} \rho_0 \tag{49}$$

Let us first consider the leading term $\gamma r G \Delta G \rho_0$. Note that $rG = v$, i.e., the infinite-horizon state-value function. Also note that $G\rho_0 = \rho_\pi$. Thus we can write $\gamma c G \Delta G \rho_0 = \gamma v \Delta \rho_\pi$. We will show that this expression equals the expected

advantage $L_\pi(\tilde{\pi}) - L_\pi(\pi)$.

$$
\begin{aligned}
L_\pi(\tilde{\pi}) - L_\pi(\pi) &= \sum_s \rho_\pi(s) \sum_a (\tilde{\pi}(a|s) - \pi(a|s)) A_\pi(s, a) \\
&= \sum_s \rho_\pi(s) \sum_a \left(\pi_\theta(a|s) - \pi_{\tilde{\theta}}(a|s)\right) \left[r(s) + \sum_{s'} p(s'|s, a)\gamma v(s') - v(s) \right] \\
&= \sum_s \rho_\pi(s) \sum_{s'} \sum_a (\pi(a|s) - \tilde{\pi}(a|s)) \, p(s'|s, a)\gamma v(s') \\
&= \sum_s \rho_\pi(s) \sum_{s'} (p_\pi(s'|s) - p_{\tilde{\pi}}(s'|s))\gamma v(s') \\
&= \gamma v \Delta \rho_\pi
\end{aligned}
\tag{50}
$$

Next let us bound the $O(\Delta^2)$ term $\gamma^2 r G\Delta G\Delta \tilde{G}\rho$. First we consider the product $\gamma r G\Delta = \gamma v\Delta$. Consider the component s of this dual vector.

$$
\begin{aligned}
|(\gamma v\Delta)_s| &= \left| \sum_a (\tilde{\pi}(s, a) - \pi(s, a)) Q_\pi(s, a) \right| \\
&= \left| \sum_a (\tilde{\pi}(s, a) - \pi(s, a)) A_\pi(s, a) \right| \\
&\le \sum_a |\tilde{\pi}(s, a) - \pi(s, a)| \cdot \max_a |A_\pi(s, a)| \\
&\le 2\alpha\epsilon
\end{aligned}
\tag{51}
$$

where the last line used the definition of the total-variation divergence, and the definition of $\epsilon = \max_{s,a} |A_\pi(s, a)|$. We bound the other portion $G\Delta \tilde{G}\rho$ using the ℓ_1 operator norm

$$
\|A\|_1 = \sup_\rho \left\{ \frac{\|A\rho\|_1}{\|\rho\|_1} \right\}
\tag{52}
$$

where we have that $\|G\|_1 = \|\tilde{G}\|_1 = 1/(1-\gamma)$ and $\|\Delta\|_1 = 2\alpha$. That gives

$$
\begin{aligned}
\|G\Delta \tilde{G}\rho\|_1 &\le \|G\|_1 \|\Delta\|_1 \|\tilde{G}\|_1 \|\rho\|_1 \\
&= \frac{1}{1-\gamma} \cdot 2\alpha \cdot \frac{1}{1-\gamma} \cdot 1
\end{aligned}
\tag{53}
$$

So we have that

$$
\begin{aligned}
\gamma^2 \left| r G\Delta G\Delta \tilde{G}\rho \right| &\le \gamma \|\gamma r G\Delta\|_\infty \|G\Delta \tilde{G}\rho\|_1 \\
&\le \gamma \|v\Delta\|_\infty \|G\Delta \tilde{G}\rho\|_1 \\
&\le \gamma \cdot 2\alpha\epsilon \cdot \frac{2\alpha}{(1-\gamma)^2} \\
&= \frac{4\gamma\epsilon}{(1-\gamma)^2} \alpha^2
\end{aligned}
\tag{54}
$$

$\square$

C   Efficiently Solving the Trust-Region Constrained Optimization Problem

This section describes how to efficiently approximately solve the following constrained optimization problem, which we must solve at each iteration of TRPO:

$$
\text{maximize } L(\theta) \text{ subject to } \overline{D}_{\text{KL}}(\theta_{\text{old}}, \theta) \le \delta.
\tag{55}
$$

The method we will describe involves two steps: (1) compute a search direction, using a linear approximation to objective and quadratic approximation to the constraint; and (2) perform a line search in that direction, ensuring that we improve the nonlinear objective while satisfying the nonlinear constraint.

The search direction is computed by approximately solving the equation $Ax = g$, where A is the Fisher information matrix, i.e., the quadratic approximation to the KL divergence constraint: $\overline{D}_{\mathrm{KL}}(\theta_{\mathrm{old}}, \theta) \approx \frac{1}{2}(\theta - \theta_{\mathrm{old}})^T A (\theta - \theta_{\mathrm{old}})$, where $A_{ij} = \frac{\partial}{\partial \theta_i} \frac{\partial}{\partial \theta_j} \overline{D}_{\mathrm{KL}}(\theta_{\mathrm{old}}, \theta)$. In large-scale problems, it is prohibitively costly (with respect to computation and memory) to form the full matrix A (or A^{-1}). However, the conjugate gradient algorithm allows us to approximately solve the equation $Ax = b$ without forming this full matrix, when we merely have access to a function that computes matrix-vector products $y \to Ay$. Appendix C.1 describes the most efficient way to compute matrix-vector products with the Fisher information matrix. For additional exposition on the use of Hessian-vector products for optimizing neural network objectives, see (Martens & Sutskever, 2012) and (Pascanu & Bengio, 2013).

Having computed the search direction $s \approx A^{-1} g$, we next need to compute the maximal step length β such that $\theta + \beta s$ will satisfy the KL divergence constraint. To do this, let $\delta = \overline{D}_{\mathrm{KL}} \approx \frac{1}{2}(\beta s)^T A (\beta s) = \frac{1}{2}\beta^2 s^T A s$. From this, we obtain $\beta = \sqrt{2\delta/s^T A s}$, where δ is the desired KL divergence. The term $s^T A s$ can be computed through a single Hessian vector product, and it is also an intermediate result produced by the conjugate gradient algorithm.

Last, we use a line search to ensure improvement of the surrogate objective and satisfaction of the KL divergence constraint, both of which are nonlinear in the parameter vector θ (and thus depart from the linear and quadratic approximations used to compute the step). We perform the line search on the objective $L_{\theta_{\mathrm{old}}}(\theta) - \mathcal{X}[\overline{D}_{\mathrm{KL}}(\theta_{\mathrm{old}}, \theta) \leq \delta]$, where $\mathcal{X}[\ldots]$ equals zero when its argument is true and $+\infty$ when it is false. Starting with the maximal value of the step length β computed in the previous paragraph, we shrink β exponentially until the objective improves. Without this line search, the algorithm occasionally computes large steps that cause a catastrophic degradation of performance.

C.1 Computing the Fisher-Vector Product

Here we will describe how to compute the matrix-vector product between the averaged Fisher information matrix and arbitrary vectors. This matrix-vector product enables us to perform the conjugate gradient algorithm. Suppose that the parameterized policy maps from the input x to "distribution parameter" vector $\mu_\theta(x)$, which parameterizes the distribution $\pi(u|x)$. Now the KL divergence for a given input x can be written as follows:

$$D_{\mathrm{KL}}(\pi_{\theta_{\mathrm{old}}}(\cdot|x) \parallel \pi_\theta(\cdot|x)) = \mathrm{kl}(\mu_\theta(x), \mu_{\mathrm{old}}) \tag{56}$$

where kl is the KL divergence between the distributions corresponding to the two mean parameter vectors. Differentiating kl twice with respect to θ, we obtain

$$\frac{\partial \mu_a(x)}{\partial \theta_i} \frac{\partial \mu_b(x)}{\partial \theta_j} \mathrm{kl}''_{ab}(\mu_\theta(x), \mu_{\mathrm{old}}) + \frac{\partial^2 \mu_a(x)}{\partial \theta_i \partial \theta_j} \mathrm{kl}'_a(\mu_\theta(x), \mu_{\mathrm{old}}) \tag{57}$$

where the primes ($'$) indicate differentiation with respect to the first argument, and there is an implied summation over indices a, b. The second term vanishes, leaving just the first term. Let $J := \frac{\partial \mu_a(x)}{\partial \theta_i}$ (the Jacobian), then the Fisher information matrix can be written in matrix form as $J^T M J$, where $M = \mathrm{kl}''_{ab}(\mu_\theta(x), \mu_{\mathrm{old}})$ is the Fisher information matrix of the distribution in terms of the mean parameter μ (as opposed to the parameter θ). This has a simple form for most parameterized distributions of interest.

The Fisher-vector product can now be written as a function $y \to J^T M J y$. Multiplication by J^T and J can be performed by most automatic differentiation and neural network packages (multiplication by J^T is the well-known backprop operation), and the operation for multiplication by M can be derived for the distribution of interest. Note that this Fisher-vector product is straightforward to average over a set of datapoints, i.e., inputs x to μ.

One could alternatively use a generic method for calculating Hessian-vector products using reverse mode automatic differentiation ((Wright & Nocedal, 1999), chapter 8), computing the Hessian of $\overline{D}_{\mathrm{KL}}$ with respect to θ. This method would be slightly less efficient as it does not exploit the fact that the second derivatives of $\mu(x)$ (i.e., the second term in Equation (57)) can be ignored, but may be substantially easier to implement.

We have described a procedure for computing the Fisher-vector product $y \to Ay$, where the Fisher information matrix is averaged over a set of inputs to the function μ. Computing the Fisher-vector product is typically about as expensive as computing the gradient of an objective that depends on $\mu(x)$ (Wright & Nocedal, 1999). Furthermore, we need to compute

k of these Fisher-vector products per gradient, where k is the number of iterations of the conjugate gradient algorithm we perform. We found $k = 10$ to be quite effective, and using higher k did not result in faster policy improvement. Hence, a naïve implementation would spend more than 90% of the computational effort on these Fisher-vector products. However, we can greatly reduce this burden by subsampling the data for the computation of Fisher-vector product. Since the Fisher information matrix merely acts as a metric, it can be computed on a subset of the data without severely degrading the quality of the final step. Hence, we can compute it on 10% of the data, and the total cost of Hessian-vector products will be about the same as computing the gradient. With this optimization, the computation of a natural gradient step $A^{-1}g$ does not incur a significant extra computational cost beyond computing the gradient g.

D   Approximating Factored Policies with Neural Networks

The policy, which is a conditional probability distribution $\pi_\theta(a|s)$, can be parameterized with a neural network. This neural network maps (deterministically) from the state vector s to a vector μ, which specifies a distribution over action space. Then we can compute the likelihood $p(a|\mu)$ and sample $a \sim p(a|\mu)$.

For our experiments with continuous state and action spaces, we used a Gaussian distribution, where the covariance matrix was diagonal and independent of the state. A neural network with several fully-connected (dense) layers maps from the input features to the mean of a Gaussian distribution. A separate set of parameters specifies the log standard deviation of each element. More concretely, the parameters include a set of weights and biases for the neural network computing the mean, $\{W_i, b_i\}_{i=1}^L$, and a vector r (log standard deviation) with the same dimension as a. Then, the policy is defined by the normal distribution $\mathcal{N}\left(\text{mean} = \text{NeuralNet}\left(s; \{W_i, b_i\}_{i=1}^L\right), \text{stdev} = \exp(r)\right)$. Here, $\mu = [\text{mean}, \text{stdev}]$.

For the experiments with discrete actions (Atari), we use a factored discrete action space, where each factor is parameterized as a categorical distribution. That is, the action consists of a tuple $(a_1, a_2, \ldots, a_K)$ of integers $a_k \in \{1, 2, \ldots, N_k\}$, and each of these components is assumed to have a categorical distribution, which is specified by a vector $\mu_k = [p_1, p_2, \ldots, p_{N_k}]$. Hence, μ is defined to be the concatenation of the factors' parameters: $\mu = [\mu_1, \mu_2, \ldots, \mu_K]$ and has dimension $\dim \mu = \sum_{k=1}^K N_k$. The components of μ are computed by taking applying a neural network to the input s and then applying the softmax operator to each slice, yielding normalized probabilities for each factor.

E   Experiment Parameters

	Swimmer	Hopper	Walker
State space dim.	10	12	20
Control space dim.	2	3	6
Total num. policy params	364	4806	8206
Sim. steps per iter.	50K	1M	1M
Policy iter.	200	200	200
Stepsize ($\overline{D}_{\text{KL}}$)	0.01	0.01	0.01
Hidden layer size	30	50	50
Discount (γ)	0.99	0.99	0.99
Vine: rollout length	50	100	100
Vine: rollouts per state	4	4	4
Vine: Q-values per batch	500	2500	2500
Vine: num. rollouts for sampling	16	16	16
Vine: len. rollouts for sampling	1000	1000	1000
Vine: computation time (minutes)	2	14	40
SP: num. path	50	1000	10000
SP: path len.	1000	1000	1000
SP: computation time	5	35	100

Table 2. Parameters for continuous control tasks, vine and single path (SP) algorithms.

	All games
Total num. policy params	33500
Vine: Sim. steps per iter.	400K
SP: Sim. steps per iter.	100K
Policy iter.	500
Stepsize ($\overline{D}_{\mathrm{KL}}$)	0.01
Discount (γ)	0.99
Vine: rollouts per state	≈ 4
Vine: computation time	≈ 30 hrs
SP: computation time	≈ 30 hrs

Table 3. Parameters used for Atari domain.

F   Learning Curves for the Atari Domain

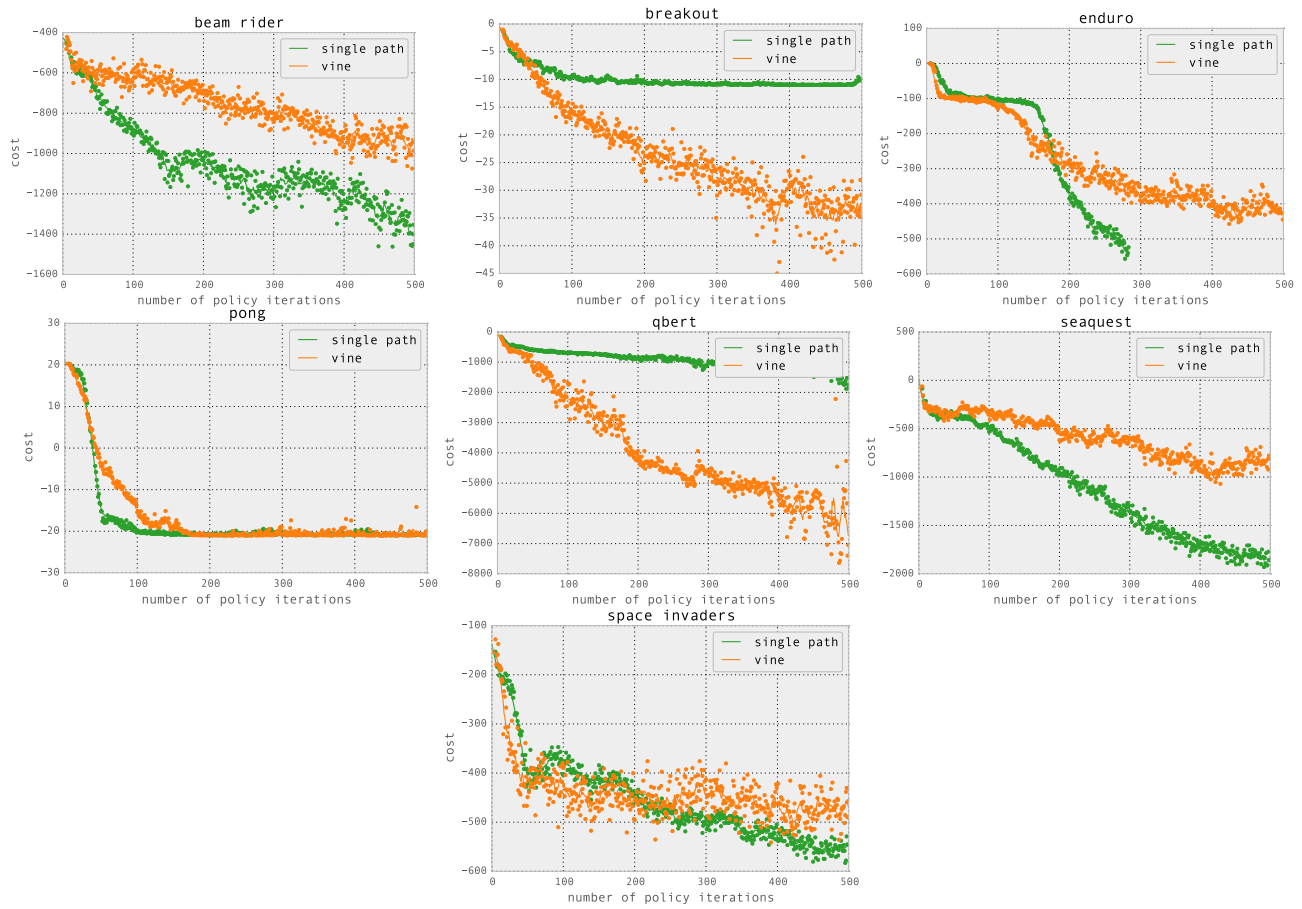*Figure 5.* Learning curves for the Atari domain. For historical reasons, the plots show cost = negative reward.