Trust Region Policy Optimization

John D. Schulman **Philipp C. Moritz** **Sergey Levine**

Michael I. Jordan **Pieter Abbeel**

Abstract

A bold dream of the field of reinforcement learning is to provide general algorithms that can solve a variety of control problems automatically. For most interesting tasks, the output of the optimal policy is a complicated nonlinear function of the observations, so we can only hope to realize this dream if we are willing to use a rich class of function approximators. Neural networks have proven to be effective for supervised learning because they have high representational power and yet can be efficiently optimized using gradient-based algorithms. This paper is motivated by the problem of developing reinforcement learning algorithms that are effective for optimizing control policies represented by large neural networks.

More concretely, we study *trust region policy optimization* (TRPO) for optimizing stochastic control policies. Two well-known methods—natural policy gradient and approximate policy iteration—are limiting cases of this algorithm. We start out with a theoretical analysis of the algorithm and prove a bound that guarantees policy improvement when certain conditions are met. In the subsequent sections we describe a practical approximate version of the algorithm by providing schemes for sampling and numerical optimization.

We apply trust region policy optimization to two challenging sets of tasks. First, we consider learning control policies for robots in a physics simulation. Our method outperforms baselines on several tasks involving 2D robots. Our method is also able to learn stable bipedal running gaits for simulated 3D robots with 19 degrees of freedom and 51 state dimensions. Second, we learn to play Atari games using raw image pixels as input to a convolutional neural network policy.

1 Introduction

Most policy optimization methods can be classified into three broad categories: policy iteration methods, which alternate between estimating the value function under the current policy and updating the policy to choose better actions [1]; policy gradient methods, which use an estimator of the gradient of the expected cost obtained from sample trajectories [2] (and which, as we later discuss, have a close connection to policy iteration); and derivative-free optimization methods, such as the cross-entropy method (CEM), which treat the cost as a black box function to be optimized in terms of the policy parameters [3, 4].

General derivative-free stochastic optimization methods such as CEM, which do not make use of the temporal aspect of the problem or compute any gradient of the objective, achieve the best results on many problems. For example, while Tetris is a classic benchmark problem for approximate dynamic programming (ADP) methods, stochastic optimization methods like the cross entropy method and covariance matrix adaptation (CMA) have achieved much better results. (See [5] for a review of this domain and the strongest counterattack so far from the ADP side.) For continuous control problems, ADP methods have had limited success, while methods like CMA have been successful at learning control policies for challenging tasks like locomotion, given hand-engineered policy classes with low-dimensional parameterizations [6]. The inability of ADP methods to beat gradient-free random search methods is unsatisfying because gradient-based optimization methods enjoy much better sample-complexity guarantees than gradient-free methods.[1] Empirically, CMA and cross-entropy seem to break down for dimensionality greater than about 50. In supervised learning, continuous optimization methods have been very successful in learning function approximators for classification and regression problems with huge numbers of parameters. The relative difficulty in learning high-dimensional policies may stem in part from the lack of a well-understood continuous optimization formulation of the policy search problem with monotonic improvement guarantees.

In this paper, we analyze a trust region method for optimizing stochastic policies. This method is amenable to theoretical analysis and effective for learning policies with large numbers of parameters. In our experiments, we learn control policies for bipedal locomotion and for playing Atari games directly from raw pixels.

2 Policy Improvement Theory for Stochastic Policies

Consider an infinite-horizon discounted Markov decision process (MDP), described by tuple $(\mathcal{S}, \mathcal{A}, P, c, \rho_0, \gamma)$, 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 distribution, $c : \mathcal{S} \to \mathbb{R}$ is the cost function, $\rho_0 : \mathcal{S} \to \mathbb{R}$ is a distribution that the initial state s_0 is drawn from, 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 the expected discounted cost of the policy.

$$\eta(\pi) = \mathbb{E}_{s_0,a_0,\ldots}\left[\sum_{t=0}^{\infty} \gamma^t c(s_t)\right],$$
$$\text{where } s_0 \sim \rho_0(s_0),\ a_t \sim \pi(\cdot|s_t),\ s_{t+1} \sim P(s_{t+1}|s_t, a_t). \quad (1)$$

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:

$$Q_\pi(s_0, a_0) = \mathbb{E}_{s_1,a_1,\ldots}\left[\sum_{t=0}^{\infty} \gamma^t c(s_t)\right],$$
$$\text{where } a_t \sim \pi(\cdot|s_t) \text{ for } t \geq 1, s_{t+1} \sim P(s_{t+1}|s_t, a_t) \text{ for } t \geq 0. \quad (2)$$
$$V_\pi(s_0) = \mathbb{E}_{a_0,s_0,\ldots}\left[\sum_{t=0}^{\infty} \gamma^t c(s_t)\right],$$
$$\text{where } a_t \sim \pi(\cdot|s_t), s_{t+1} \sim P(s_{t+1}|s_t, a_t) \text{ for } t \geq 0. \quad (3)$$
$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s). \quad (4)$$

The following useful identity expresses the expected cost of a policy $\tilde{\pi}$ in terms of the advantage over π, accumulated over timesteps (see [10] for proof):

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{s_0,a_0,s_1,a_1,\ldots}\left[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t)\right]$$
$$\text{where } s_0 \sim \rho_0(s_0),\ a_t \sim \tilde{\pi}(\cdot|s_t),\ s_{t+1} \sim P(s_{t+1}|s_t, a_t) \quad (5)$$

Let ρ_π be the (unnormalized) discounted visitation frequencies

$$\rho_\pi(s) = (P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \ldots). \quad (6)$$

[1] The oracle complexity model of optimization assumes that we have an oracle that we can query to obtain a noisy measurement of the function value or subgradient, and it asks how many queries are required to achieve a given optimization error. With an oracle that gives us subgradients, there are upper bounds on complexity that do not depend on dimensionality d of the underlying space [7]. However, the complexity scales as d^2 if we are given single noisy function evaluations [8] and d if we are given pairs of evaluations with the same noise sample [9].

assuming that $s_0 \sim \rho_0$ and actions are chosen according to π. Rearranging Equation (5) to sum over states instead of timesteps, we obtain

$$\eta(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_{\tilde{\pi}}(s) \sum_a \tilde{\pi}(a|s) A_\pi(s,a). \tag{7}$$

This equation implies that any policy update $\pi \to \tilde{\pi}$ that has a non-positive expected advantage at *every* state s, i.e., $\sum_a \tilde{\pi}(a|s)A_\pi(s,a) \leq 0$, is guaranteed to reduce 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 $\tilde{\pi}(s) = \arg\min_a A_\pi(s,a)$) improves the policy if there is at least one state-action pair with a negative advantage value (otherwise it has converged). However, in the approximate setting, it will typically be unavoidable, due to estimation and approximation error[2], that there will be some states s for which the expected advantage is positive (i.e., bad), that is, $\sum_a \tilde{\pi}(a|s)A_\pi(s,a) > 0$. Slightly modifying the definitions from Kakade and Langford [10], we introduce the following local approximation to η:

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_s \rho_\pi(s) \sum_a \tilde{\pi}(a|s) A_\pi(s,a). \tag{8}$$

Note that L_π uses the visitation frequency ρ_π rather than $\rho_{\tilde{\pi}}$, i.e., it ignores the change in state-space visitation density due to changing the policy. However, L_π matches η to first order, in the following sense. Suppose we have a parameterized policy π_θ, where $\pi_\theta(a|s)$ is a differentiable function of the parameter θ. Then for any parameter value θ_0,

$$L_{\pi_{\theta_0}}(\pi_{\theta_0}) = \eta(\pi_{\theta_0}), \tag{9}$$

$$\nabla_\theta L_{\pi_{\theta_0}}(\pi_\theta)\big|_{\theta=\theta_0} = \nabla_\theta \eta(\pi_\theta)\big|_{\theta=\theta_0}. \tag{10}$$

Equation (10) implies that a sufficiently small step $\pi_{\theta_0} \to \tilde{\pi}$ that reduces $L_{\pi_{\theta_{\text{old}}}}$ will also reduce η. However, this equation does not give us any guidance on how big of a step to take. Kakade and Langford [10] provided explicit lower bounds on the improvement of η, under a policy updating scheme called conservative policy iteration.

Conservative policy iteration uses the following scheme to update the policy at each iteration. Let π_{old} denote the current policy. First, solve for $\pi' = \arg\min_{\pi'} L_{\pi_{\text{old}}}(\pi')$. The new policy π_{new} is taken to be the following mixture between the policies π_{old} and π':

$$\pi_{\text{new}}(a|s) = (1-\alpha)\pi_{\text{old}}(a|s) + \alpha\pi'(a|s) \tag{11}$$

Kakade and Langford proved the following result:

$$\eta(\pi_{\text{new}}) \leq L_{\pi_{\text{old}}}(\pi_{\text{new}}) + \frac{2\epsilon\gamma}{(1-\gamma(1-\alpha))(1-\gamma)}\alpha^2 \tag{12}$$

where ϵ is the maximum advantage (positive or negative) of π' relative to π:

$$\epsilon = \max_s |\mathbb{E}_{a\sim\pi'(a|s)}\left[A_\pi(s,a)\right]| \tag{13}$$

Equation (12) implies the slightly weaker but simpler bound

$$\eta(\pi_{\text{new}}) \leq L_{\pi_{\text{old}}}(\pi_{\text{new}}) + \frac{2\epsilon\gamma}{(1-\gamma)^2}\alpha^2, \tag{14}$$

which drops the factor $(1-\alpha)$, since the relevant regime has $\alpha \ll 1$.

The result in Equation (12) implies that if we make a policy update that improves the right-hand side, we can guarantee that it will improve the true expected cost objective η. (See Equation (19) below.) Our principal theoretical result is that the policy improvement bound can be extended to a more general policy update scheme, where α is replaced by a distance measure between π and $\tilde{\pi}$. The total variation divergence between two discrete probability distributions p, q is defined as $D_{TV}(p\|q) = \frac{1}{2}\sum_i |p_i - q_i|$, and we define $D_{\text{TV}}^{\max}(\pi, \tilde{\pi})$ as

$$D_{\text{TV}}^{\max}(\pi,\tilde{\pi}) = \max_s D_{TV}(\pi(\cdot|s)\|\tilde{\pi}(\cdot|s)). \tag{15}$$

[2] By estimation error, we are referring to the fact certain quantities (like Q-values) are estimated by sampling, and there some random error when a finite number of samples is used. By approximation error, we are referring to the fact that approximate policies and value functions (rather than lookup tables) which can't exactly represent the true value functions or optimal policies.

Theorem 1. *Let* $\alpha = D_{\rm TV}^{\rm max}(\pi_{\rm old}, \pi_{\rm new})$, *and let* $\epsilon = \max_s \max_a |A_\pi(s,a)|$. *Then Equation* (14) *holds.*

We provide two proofs in the appendix. The first proof adapts Kakade and Langford's proof and uses the fact that if two probability distributions have total variation divergence less than α, we can couple the corresponding random variables so that they are equal with probability $1-\alpha$. The second proof uses perturbation theory to prove a slightly stronger version of Equation (14), which has a more favorable definition of ϵ that depends on $\tilde{\pi}$.

Next, we note the following relationship between the total variation divergence and the KL divergence ([11], Ch. 3): $D_{TV}(p\|q)^2 \leq D_{\rm KL}(p\|q)$. If we define $D_{\rm KL}^{\rm max}$ analogously to $D_{\rm TV}^{\rm max}$ as $D_{\rm KL}^{\rm max}(\pi, \tilde{\pi}) = \max_s D_{\rm KL}(\pi(\cdot|s) \| \tilde{\pi}(\cdot|s))$, then it follows from Equation (14) that

$$\eta(\tilde{\pi}) \leq L_\pi(\tilde{\pi}) + C D_{\rm KL}^{\rm max}(\pi, \tilde{\pi}), \quad \text{where } C = \frac{2\epsilon\gamma}{(1-\gamma)^2}. \tag{16}$$

Algorithm 1 describes an approximate policy iteration scheme based on the preceding policy improvement bounds. Here we assume that there is no estimation error, i.e., the advantage values A_π can be estimated exactly.

Algorithm 1 Approximate policy iteration algorithm guaranteeing non-increasing expected cost η

```
Initialize $\pi_0$.
for $i = 0, 1, 2, \ldots$ until convergence do
    Compute the advantage values $A_{\pi_i}(s,a), \forall a \in \mathcal{A}, s \in \mathcal{S}$.
    Solve the constrained optimization problem
        $\pi_{i+1} = \arg\min_\pi \left[ L_{\pi_i}(\pi) + \left( \frac{2\epsilon\gamma}{(1-\gamma)^2} \right) D_{\rm KL}^{\rm max}(\pi_i, \pi) \right]$
            where $\epsilon = \max_s \max_a |A_\pi(s,a)|$    (17)
            and $L_{\pi_i}(\pi) = \eta(\pi_i) + \sum_s \rho_{\pi_i}(s) \sum_a \pi(a|s) A_{\pi_i}(s,a)$    (18)
end for
```

It follows from Equation 16 that Algorithm 1 is guaranteed to generate a sequence of monotonically improving policies $\eta(\pi_0) \geq \eta(\pi_1) \geq \eta(\pi_2) \geq \ldots$. To see this, let $M_i(\pi) = L_{\pi_i}(\pi) + C D_{\rm KL}^{\rm max}(\pi_i, \pi)$. Then

$$\begin{aligned} \eta(\pi_{i+1}) &\leq M_i(\pi_{i+1}) \text{ by Equation (16)} \\ \eta(\pi_i) &= M_i(\pi_i) \\ \text{therefore,} \quad \eta(\pi_{i+1}) - \eta(\pi_i) &\leq M_i(\pi_{i+1}) - M(\pi_i). \end{aligned} \tag{19}$$

Thus, by minimizing M_i at each iteration, we guarantee that the true objective η is non-increasing. This algorithm is a type of majorization-minimization algorithm [12], which is a class of methods that also includes expectation maximization. In the terminology of MM algorithms, M_i is the surrogate function that majorizes η with equality at π_i. This algorithm is also reminiscent of proximal gradient methods and mirror descent.[3]

Trust region policy optimization, which we propose in the following section, is an approximation to Algorithm 1, which uses an approximation of the constraint and uses a hard constraint rather than a penalty.

[3] The policy update algorithm we have described has precisely the form of a proximal gradient algorithm [13] (with a non-quadratic penalty function). That is, we minimize an affine approximation to the objective (note that L is affine in terms of the probabilities) plus a regularization term (KL divergence) that reduces the step length. Relatedly, the policy update is the same update that is performed in online mirror descent [14], using entropy as the regularizer. The KL divergence arises as the Bregman divergence from the entropy regularizer; this form of online mirror descent is commonly used for optimizing over probability distributions [15].

3 Optimization of Parameterized Policies

Section 2 considered the abstract setting of optimizing a stochastic policy π, without concern for its parameterization or how to estimate the objective L. Sections 3 to 5 describe how to turn the abstract policy-updating procedure from Section 2 into a practical algorithm.

In this section, we consider optimizing 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 $D_{\mathrm{KL}}(\theta \| \tilde{\theta}) := D_{\mathrm{KL}}(\pi_\theta \| \pi_{\tilde{\theta}})$. We will use θ_{old} to denote the previous policy parameters that we want to improve upon.

The preceding section shows that $\eta(\theta) \leq L_{\theta_{\mathrm{old}}}(\theta) + CD_{\mathrm{KL}}^{\max}(\theta_{\mathrm{old}}, \theta)$, with equality at $\theta = \theta_{\mathrm{old}}$. Thus, by solving the following minimization, we are guaranteed to improve the true objective η:

$$\underset{\theta}{\operatorname{minimize}} \left[CD_{\mathrm{KL}}^{\max}(\theta_{\mathrm{old}}, \theta) + L_{\theta_{\mathrm{old}}}(\theta) \right]. \tag{20}$$

In practice, if we used the penalty coefficient C recommended by the theory above, the stepsizes would be very small. One way to take largers in a robust way is to use a hard constraint on the KL divergence between the new policy and the old policy, i.e., a trust region constraint[4]:

$$\underset{\theta}{\operatorname{minimize}}\, L_{\theta_{\mathrm{old}}}(\theta) \ \text{ subject to } D_{\mathrm{KL}}^{\max}(\theta_{\mathrm{old}}, \theta) \leq \delta. \tag{21}$$

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 constraint may be computationally too expensive. Instead, we can use a heuristic approximation which considers the average KL divergence:

$$\overline{D}_{\mathrm{KL}}^{\rho}(\theta_1, \theta_2) := \mathbb{E}_{s\sim\rho}\left[D_{\mathrm{KL}}(\pi_{\theta_1}(\cdot|s) \| \pi_{\theta_2}(\cdot|s)) \right]. \tag{22}$$

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

$$\underset{\theta}{\operatorname{minimize}}\, L_{\theta_{\mathrm{old}}}(\theta) \ \text{ subject to } \overline{D}_{\mathrm{KL}}^{\rho_{\theta_{\mathrm{old}}}}(\theta_{\mathrm{old}}, \theta) \leq \delta. \tag{23}$$

The natural policy gradient ([17]) can be obtained as a special case of this update by using a linear approximation to L and a quadratic approximation to the $\overline{D}_{\mathrm{KL}}$ constraint in Equation (23), resulting in the following problem:

$$\underset{\theta}{\operatorname{minimize}} \left[\nabla_\theta L(\theta)\big|_{\theta=\theta_{\mathrm{old}}} \cdot (\theta - \theta_{\mathrm{old}}) \right] \ \text{ subject to } \frac{1}{2}(\theta_{\mathrm{old}} - \theta)^T A(\theta_{\mathrm{old}} - \theta) \leq \delta, \tag{24}$$

which has the solution $\theta_{\mathrm{new}} = \theta_{\mathrm{old}} - A(\theta_{\mathrm{old}})^{-1} \nabla_\theta L(\theta)\big|_{\theta=\theta_{\mathrm{old}}}$. Here,

$$A(\theta_{\mathrm{old}})_{ij} = \frac{\partial}{\partial\theta_i}\frac{\partial}{\partial\theta_j} \mathbb{E}_{s\sim\rho_\pi}\left[D_{\mathrm{KL}}(\pi(\cdot|s, \theta_{\mathrm{old}}) \| \pi(\cdot|s, \theta)) \right] \big|_{\theta=\theta_{\mathrm{old}}}. \tag{25}$$

Note that A is the averaged Fisher information matrix, i.e., Fisher information matrix of $\pi_\theta(\cdot|s)$ averaged under $s \sim \rho_\pi$. The averaged KL-divergence and averaged Fisher information matrix has been used in several other policy search methods [2, 18]. Bagnell pointed out that the averaged Fisher information measures the Fisher information on the distribution over trajectories induced by π_θ [19].

Finally, note that the classic policy gradient update can be written as

$$\underset{\theta}{\operatorname{minimize}} \left[\nabla_\theta L(\theta)\big|_{\theta=\theta_{\mathrm{old}}} \cdot (\theta - \theta_{\mathrm{old}}) \right] \ \text{ subject to } \frac{1}{2}\|\theta - \theta_{\mathrm{old}}\|^2 \leq \delta. \tag{26}$$

[4]For background on trust region methods in nonlinear optimization, see [16], chapter 4. A variety of nonlinear optimization algorithms work by solving a series of trust region subproblems, where a local approximation to the objective is optimized subject to a trust region constraint, which restricts the solution to the region where the approximation is valid.

4 Sample-Based Estimation of the Objective and Constraint

The previous section proposed a constrained optimization problem on the policy parameters (Equation (23)), which optimizes an estimate of the expected cost η subject to a constraint 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 $L_{\theta_{\text{old}}}$ in Equation (23):

$$\begin{aligned}\underset{\theta}{\text{minimize}} \sum_s \rho_{\theta_{\text{old}}}(s) \sum_a \pi_\theta(a|s) A_{\theta_{\text{old}}}(s,a) \\ \text{subject to } \overline{D}_{\text{KL}}^{\rho_{\theta_{\text{old}}}}(\theta_{\text{old}}, \theta) \leq \delta.\end{aligned} \tag{27}$$

We first replace $\sum_s \rho_{\theta_{\text{old}}}(s)\,[\dots]$ in the objective by the expectation $\frac{1}{1-\gamma}\mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}}[\dots]$.

Next, we replace the advantage values $A_{\theta_{\text{old}}}$ by the Q-values $Q_{\theta_{\text{old}}}$ in Equation (27), and the objective changes by an irrelevant constant independent of θ.

Last, we replace the sum over the action space by an importance sampling estimator. In the following equations, q denotes the distribution used for sampling. Consider the contribution of a single state s_n to the loss function:

$$\sum_a \pi_\theta(a|s_n) A_{\theta_{\text{old}}}(s_n, a) = \mathbb{E}_{a\sim q}\left[\frac{\pi_\theta(a|s_n)}{q(a|s_n)} A_{\theta_{\text{old}}}(s_n, a)\right] \tag{28}$$

Our optimization problem (Equation (27)) is exactly equivalent to the following one, written in terms of expectations:

$$\begin{aligned}\underset{\theta}{\text{minimize}}\ \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}, a\sim q}\left[\frac{\pi_\theta(a|s)}{q(a|s)} Q_{\theta_{\text{old}}}(s,a)\right] \\ \text{subject to } \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}}\left[D_{\text{KL}}(\pi_{\theta_{\text{old}}}(\cdot|s) \| \pi_\theta(\cdot|s))\right] \leq \delta.\end{aligned} \tag{29}$$

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 doing this estimation, each with their own advantages and disadvantages.

The first sampling scheme, which we call *single path*, is the one that is typically used for policy gradient estimation (e.g., [20]), and is based on sample paths. The second sampling 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 (e.g., [21, 5].)

4.1 Single Path

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

4.2 Vines

In this estimation procedure, we first sample $s_0 \sim \rho_0$ and simulate the policy π_{θ_i} to generate a trajectory of states and controls. Then we choose a subset of N of the points in these trajectories, $s_1, s_2, \dots, s_N$, called the "rollout set". For each state s_n in the rollout set, sample K actions according to $a_{n,k} \sim q(\cdot|s_n)$. We can take $q(\cdot|s_n) = \pi_{\theta_i}(\cdot|s_n)$, or some other distribution.

For each of these actions, we estimate $\hat{Q}_{\theta_i}(s_n, a_{n,k})$ by performing a rollout (i.e., a short trajectory) whose first action is $a_{n,k}$. 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. See

[22], Sec. 6.4.2 for additional discussion on Monte-Carlo estimation of Q-values and [23] for further discussion of common random numbers in reinforcement learning.

Lastly, one can reduce the variance of the estimated objective (and more importantly, its gradient) by using self-normalized importance sampling or subtracting out the mean Q-value at each state.

The benefit of the *vine* scheme over the *single path* scheme that is our local estimate of the objective has much lower variance given the same number of samples. The disadvantage is that we must collect more data for each of these samples. Furthermore, the vines procedure 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. (E.g., see [24].)

5 Practical Algorithm

Here we describe two variants of a practical policy optimization algorithm based on the ideas above. The algorithms differ in their sampling scheme, using the *single path* or *vine* procedure from Section 4. The algorithms repeatedly perform the following steps:

1. Use the *single path* or *vine* procedures to collect a set of states along with estimated Q-values.
2. By averaging over samples, construct the estimated objective and constraint in Equation (29).
3. Approximately solve this constrained optimization problem to update the policy's parameter vector θ. In our implementation, we use the conjugate gradient method to perform a truncated Newton step, based on a quadratic approximation to the KL divergence, as described in Appendix D. We use a line search that ensures that we are improving the objective and satisfying the exact (non-quadratic) KL-divergence constraint.

Since we are making a quadratic approximation to the KL divergence constraint, the algorithm we have implemented is quite similar to the natural policy gradient algorithm, proposed by Kakade [17]. However, there are several important differences between our implementation and typical implementations of the natural policy gradient algorithm: (1) we construct the Fisher information matrix by taking the analytic derivative of the KL divergence, (2) we put a hard constraint on the KL divergence, and we use a line search to ensure improvement of the surrogate objective and satisfaction of the KL divergence constraint.

Several variants of the algorithm we described are conceivable and would depart further from the natural policy gradient algorithm; for example, one could use a stochastic optimization method to approximately solve the trust region optimization problem; in other domains, stochastic methods have been more successful than batch methods for optimizing neural network parameters.

The next section will refer to the two variants of the algorithm above as the *vine* and *single path* algorithm and evaluate them on a variety of tasks.

6 Experiments

A video showing the policies generated by trust region policy optimization (TRPO) for these experiments is available at `http://rll.berkeley.edu/policyopt`.

6.1 Robotic Locomotion Domain

We evaluated the vine and single-path variants of TRPO on the task of learning locomotion controllers for simulated robots. Most prior work that has learned locomotion controllers with reinforcement learning has used hand-engineered policy representations with a small number of parameters. On the other hand, our policies map directly from kinematics to joint torques and have tens of thousands of parameters. (See Table 2 for the exact number of parameters.)

The simulated robots consist of linked rigid bodies, and the 2D models are shown in Figure 2. They were simulated using MuJoCo [25]. The swimmer (left) has 10 state dimensions and propels itself forward by making an undulating motion. The other two robots hop and walk, respectively, and have 12 and 18 state dimensions. Underactuation, high dimensionality, and non-smooth dynamics due to contacts makes the latter two tasks challenging for reinforcement learning methods. The states of the robots are their generalized positions and velocities, and the controls are their joint torques. The cost rates were defined to be a linear reward for forward velocity and a quadratic penalty on joint torque:

$$\text{cost}(x, u) = -v_x + 10^{-5}\|u\|^2 - 10 \tag{30}$$

For the hopper and biped, the rollouts were terminated when the robot fell. For the biped, we also added a penalty for strong impacts of the feet against the ground to encourage a smooth walk rather than a hopping gait.

We used neural networks as the parameterization of the stochastic policy. Appendix A provides details of precisely how π_θ is parameterized. We compared the single path and vine algorithms to two other approaches: reward-weighted regression (RWR) [26][5] and the cross entropy method. All of the methods were used to optimize the same neural-network parameterization of the policy.[6] A detailed listing of parameters used in the experiment is provided in Appendix E. We include the classic cart-pole balancing task in additional to the more challenging locomotion domains, based on the formulation from Barto et al. [27].

Learning curves of the policy optimization methods are shown in Figure 2. The vine algorithm was able to solve all of the tasks, learning a stable and naturalistic gait for the 2D hopper and walker. The single path algorithm exhibits the best looking learning curves (the most reliably monotonic improvement of total expected cost), but it did not yield a locomotion controller for the 2D walker; instead, it yielded a controller that stood up but did not bother to move.

The cross-entropy method was not able to solve any of the tasks other than cart-pole, presumably because it does not perform well for optimization problems with more than a couple dozen parameters. Reward-weighted regression also performed reasonably well on the tasks, which is consistent with the fact that it is performing gradient-based optimization of the policy, thus it can be expected to scale with the number of parameters.[7]

We have similarly been able to learn stable walking controllers for a 3D humanoid model, which has 51 state dimensions and 19 actuators, using the *single path* algorithm. While the initial policy falls over immediately, after several hundred batches of policy iterations, we obtain a policy that make substantial forward progress and survives for more than ten seconds before falling over. A learning curve of one of the cost functions considered is shown in Figure 2. Some other gaits are shown in our demo video. To obtain the 3D walking results, the policy optimization algorithm described in this paper was supplemented by a variance reduction scheme based on value function estimation, which will be described in a later paper.

6.2 Atari Domain

Next, we applied trust region policy optimization to learn policies for playing Atari games, using raw images as input. These games require a variety of different behaviors, such as dodging bullets and hitting balls with paddles. Some games are made more difficult by the timing of reward delivery; for example, in most games, no immediate penalty is received after losing a life. The diversity of the games makes Atari an interesting testbed for reinforcement learning. We tested our policy optimization algorithms on the same seven games that Mnih et al. [28] reported results on, since theirs is the only prior work on this domain that uses raw images as input.

[5]We implemented the following variant of reward-weighted regression. First, sample a set of trajectories from the stochastic policy. Select the top 10% of trajectories that achieve the lowest total cost. Then use L-BFGS to maximize the log-likelihood of all of the controls performed along those trajectories.

[6] We reduced the number of hidden units for the cross-entropy method, which improved its performance.

[7] The learning curves slightly overstate the performance of RWR, though, because the experiment with RWR used a single initial state, whereas the other methods used a distribution over initial states. Using a single initial state was necessary for good performance of our implementation of RWR, but there may be some modifications that make RWR suitable for a distribution of initial states.

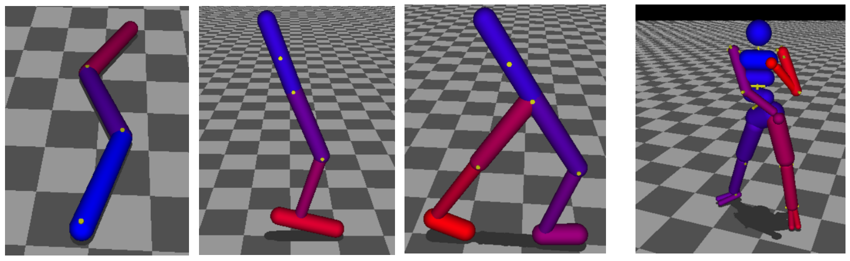

Figure 1: Robot models used for locomotion experiments, instantiated in MuJoCo physics simulator. The three models on the left are constrained to two dimensions and are called the swimmer, hopper, and walker—these models were used for the main experimental comparisons.

	B. Rider	*Breakout*	*Enduro*	*Pong*	*Q*bert*	*Seaquest*	*S. Invaders*
Random	354	1.2	0	−20.4	157	110	179
Human	7456	31.0	368	−3.0	18900	28010	3690
Deep Q Network	4092	168.0	470	20.0	1952	1705	581
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 reinforcement algorithms on the Atari domain according to [28]. Our algorithms (bottom two rows) were run once on each task, with the exact same architecture and same parameters. Note that performance varies substantially from run to run (with different random initializations of the policy), but it would be prohibitively expensive to perform enough repetitions of the training procedure to obtain error statistics.

Before feeding the images to the policy, we convert them to grayscale and downsample them by a factor of four. We stack the last four images as input to the policy, giving a $4 \times 52 \times 40$ array. The policy is represented by a convolutional neural network, whose structure is shown in Figure 5.

The results of the *vine* and *single path* algorithms are summarized in Table 1, which also includes the numbers reported my Mnih et al. [28] for their Deep Q network and an expert human's performance. The 500 iterations of our algorithm took about 30 hours (with slight variation between games) on a 16-core computer.

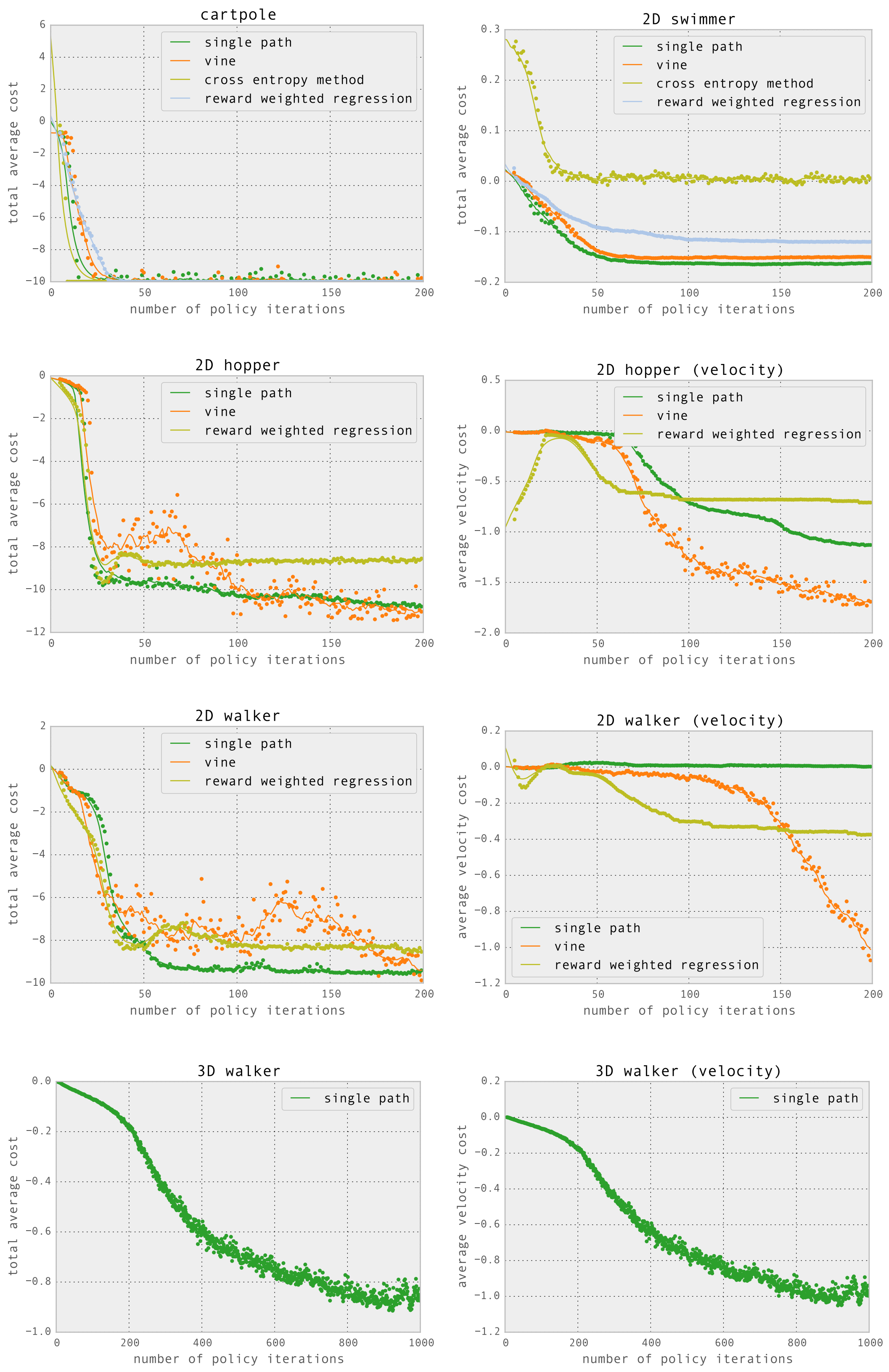

Figure 2: Learning curves for the locomotion domain. For the 2D hopper, 2D walker, and 3D walker, the total composite cost is shown on the left, and the velocity cost (negative velocity is plotted on the right.)

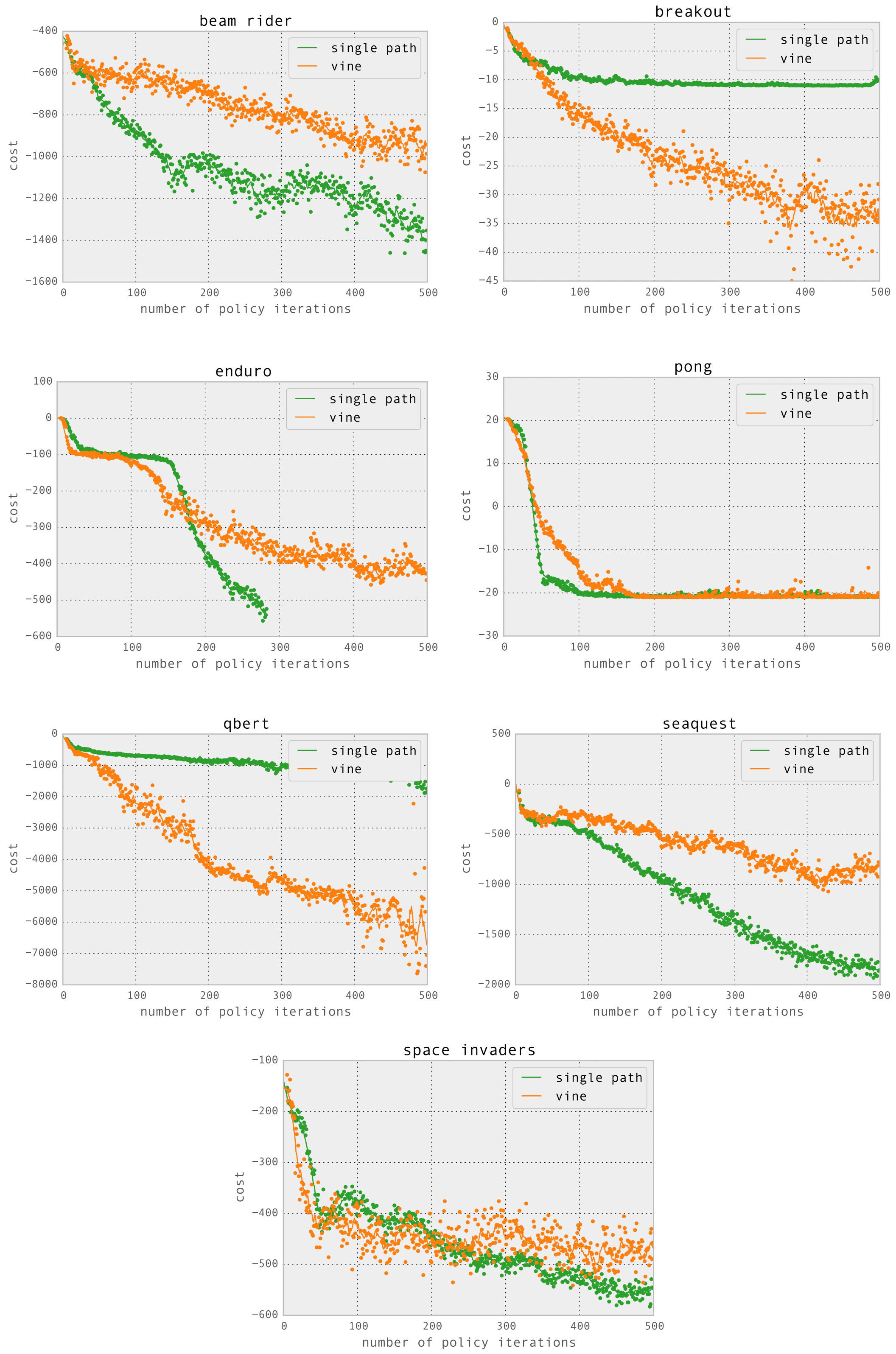

Figure 3: Learning curves for the Atari domain

7 Discussion

In this paper we theoretically and empirically studied trust region methods for optimizing stochastic control policies. We proved a monotonic improvement result for an algorithm that repeatedly optimizes a local approximation to the expected cost of the policy subject to a KL divergence constraint. Furthermore, our analysis helps provide 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. KL divergence constraints have been employed by a number of recent policy search methods [17, 2], however this result is the first to prove the soundness of such a scheme.

Subsequently, we discussed the questions of how to perform the estimation and optimization steps, culminating in the practical instantiation of the algorithm that was described in Section 5. We presented two different sampling schemes that can be used along with our algorithm—the *vine* and *single path* procedures.

We tested our algorithm (in both of its variants) in two different challenging domains. In the first domain, we considered robotic locomotion tasks with continuous state and action spaces. We successfully obtained locomotion controllers for walking and hopping in 2D and 3D physics simulators. To our knowledge, no prior work has learned controllers from scratch for these tasks, using a generic policy search method and non-engineered policy representation.

In the second domain, we learned policies for playing Atari games, using raw images as inputs to a convolutional neural network policies. Relative to a recently proposed method method based on Q-learning [28], our method performs better on some games but worse on others. However, our approach to policy optimization arguably has two fundamental advantages over Q-learning. First of all, it is generally easier to accurately approximate a policy than a Q function: to approximate the policy, we merely need the difference in quality between a pair of actions at a given state; whereas Q-learning depends on also accurately measuring the values of states, which unavoidably have a large dynamic range. Second, the performance of our algorithm is easier to analyze theoretically and to debug as a practitioner, since it is based on nonlinear optimization and generates a monotonically improving sequence of policies. On the other hand, in Q-learning (and other methods based on greedy policies), it is not clear how the accuracy of the Q-function approximation relates to the performance of the poolicy.

References

[1] Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 2. Athena Scientific Belmont, MA, 4th edition, 2012.

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

[3] Michael C Fu, Fred W Glover, and Jay April. Simulation optimization: a review, new developments, and applications. In *Proceedings of the 37th conference on Winter simulation*, pages 83–95. Winter Simulation Conference, 2005.

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

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

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

[7] Arkadi Nemirovski. Efficient methods in convex programming. 2005.

[8] Ohad Shamir. On the complexity of bandit and derivative-free stochastic convex optimization. *arXiv preprint arXiv:1209.2388*, 2012.

[9] John C Duchi, Michael I Jordan, Martin J Wainwright, and Andre Wibisono. Optimal rates for zero-order optimization: the power of two function evaluations. *arXiv preprint arXiv:1312.2139*, 2013.

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

[11] David Pollard. *Asymptopia: an exposition of statistical asymptotic theory*. 2000.

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

[13] Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3):123–231, 2013.

[14] Shai Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends in Machine Learning*, 4(2):107–194, 2011.

[15] Jyrki Kivinen and Manfred K Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132(1):1–63, 1997.

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

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

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

[19] J Andrew Bagnell and Jeff Schneider. Covariant policy search. IJCAI, 2003.

[20] Peter L Bartlett and Jonathan Baxter. Infinite-horizon policy-gradient estimation. *arXiv preprint arXiv:1106.0665*, 2011.

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

[22] Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA, 3rd edition, 2005.

[23] Andrew Y Ng and Michael Jordan. Pegasus: A policy search method for large mdps and pomdps. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 406–415. Morgan Kaufmann Publishers Inc., 2000.

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

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

[26] Jan Peters and Stefan Schaal. Reinforcement learning by reward-weighted regression for operational space control. In *Proceedings of the 24th international conference on Machine learning*, pages 745–750. ACM, 2007.

[27] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *Systems, Man and Cybernetics, IEEE Transactions on*, (5):834–846, 1983.

[28] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[29] David Asher Levin, Yuval Peres, and Elizabeth Lee Wilmer. *Markov chains and mixing times*. American Mathematical Society, 2009.

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

[31] James Bergstra et al. Theano: a CPU and GPU math expression compiler.

A Approximating policies with neural networks

To represent the stochastic policy π_θ, we use a neural network with weights θ. The network maps the observations to a set of parameters indexing the probability distribution that is then used to sample the controls for the rollouts. We now describe the architecture used in the respective domains in more detail.

Locomotion domain. The input x_0 of the network are joint angles and velocities as well as cartesian coordinates of body parts. As shown in Fig. 4, these are passed through a fully connected layer with a "soft rectifier" nonlinearity $\sigma(x) = \log(1 + e^x)$; the final layer uses a softmax nonlinearity to compute the means $\mu = \mathrm{softmax}(W_{12} \cdot \sigma(W_{01} x_0 + b_1) + b_2)$ of a normal distribution; the diagonal covariance matrix Σ of this distribution is also learned, but does not depend on x_0. Finally, the controls are sampled from $\mathcal{N}(\mu, \Sigma)$ and clipped to the torque limits.

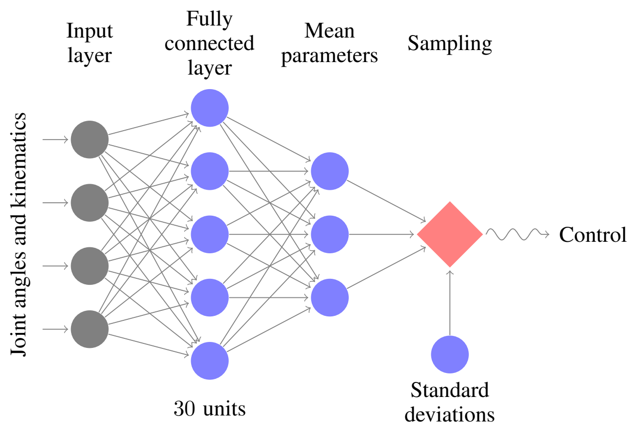

Figure 4: Neural network architecture for the locomotion domain: Two fully connected hidden layers transform the input to the mean μ of a Normal distribution from which the controls are sampled.

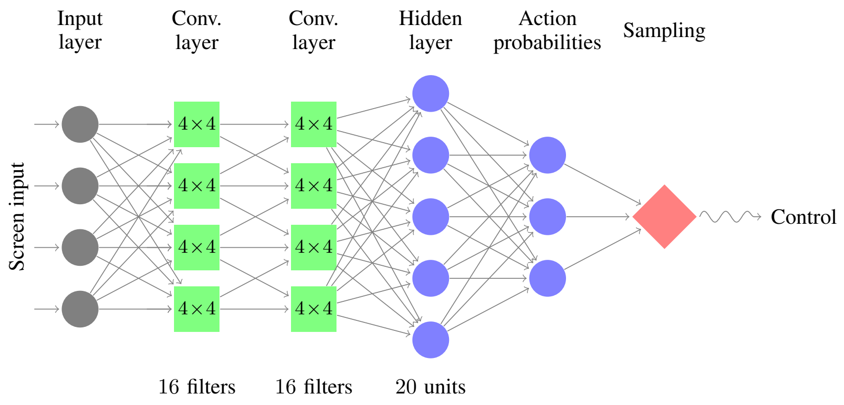

Figure 5: Neural network architecture for the Atari domain: The screen input is downsampled from $4 \times 210 \times 160$ to $4 \times 52 \times 40$, passed through two layers of 16 filters of 4×4 convolutions, flattened and passed through a layer of 20 hidden units. Their output are parameters for a probability distribution from which the actions are sampled.

Atari domain. In the Atari domain, the observations consist of the screen pixels from the last 4 timesteps; each of these 210×160 images is downsampled by a factor of four. The architecture of the network is shown in Fig. 5. We first apply two convolutional layers with 16 filters each, using tanh nonlinearities and no max pooling. The resulting image is passed through a fully connected layer with 20 units. The final layer uses a softmax nonlinearity to get the probabilities of a multinomial distribution, from which the actions are sampled.

B Proof of Policy Improvement Bound

We will adapt Kakade and Langford's proof to the more general setting considered in this paper. First let us review their proof for self-containedness (and since our notation differs from theirs by constant factors.) Recall the useful identity introduced in Section 2, which expresses the policy improvement as an accumulation of expected advantages over time:

$$\eta(\pi_{\text{new}}) = \eta(\pi_{\text{old}}) + \mathbb{E}_{s_0, a_0, s_1, a_1, \ldots}\left[\sum_{t=0}^{\infty} \gamma^t A_{\pi_{\text{old}}}(s_t, a_t)\right]$$
$$\text{where } s_0 \sim \rho_0(s_0),\ a_t \sim \pi_{\text{new}}(\cdot|s_t),\ s_{t+1} \sim P(s_{t+1}|s_t, a_t). \qquad (31)$$

Define $\bar{A}(s)$ as the expected advantage of $\tilde{\pi}$ at state s, averaged over the actions it might take:

$$\bar{A}(s) = \sum_a A_{\pi_{\text{old}}}(s,a)\pi_{\text{new}}(a|s) \tag{32}$$

Then Equation (31) can be reorganized as follows

$$\eta(\pi_{\text{new}}) = \eta(\pi_{\text{old}}) + \sum_t \gamma^t \mathbb{E}_{s\sim P(s_t|\pi_{\text{new}})}\left[\bar{A}(s)\right]. \tag{33}$$

Recall that in conservative policy iteration, the new policy π_{new} is taken to be a mixture of the old policy π_{old} and an increment π', i.e., $\pi_{\text{new}}(a|s) = \pi_{\text{old}}(a|s) + \pi'(a|s)$. In other words, to sample from π_{new}, we first draw a Bernoulli random variable, which tells us to choose π_{old} with probability $(1-\alpha)$ and choose π' with probability α. Let c_t be the random variable that indicates the number of times π' was chosen for $t' < t$. We can condition on the value of c_t to break the probability distribution $P(s_t = s)$ into two pieces. Let us consider the expected advantage at timestep t. All expectations over s_t are taken assuming that the policy π_{new} was executed for $t = 0, 1, \dots, t-1$.

$$\begin{aligned}
\mathbb{E}_{s_t}\left[\bar{A}(s_t)\right] &= P(c_t = 0)\mathbb{E}_{s_t}\left[\bar{A}(s) \mid c_t = 0\right] + P(c_t > 0)\mathbb{E}_{s_t}\left[\bar{A}(s) \mid c_t > 0\right] \qquad (34)\\
&= P(c_t = 0)\mathbb{E}_{s_t}\left[\bar{A}(s_t) \mid c_t > 0\right] + P(c_t > 0)\mathbb{E}_{s_t}\left[\bar{A}(s_t) \mid c_t > 0\right]\\
&= (1 - P(c_t \geq 1))\mathbb{E}_{s_t}\left[\bar{A}(s_t)\right] + P(s_t|c_t \geq 1)\mathbb{E}_{s_t}\left[\bar{A}(s_t) \mid c_t < 0\right]\\
&= \mathbb{E}_{s_t \mid c_t < 0}\left[\bar{A}(s_t)\right] + P(c_t \geq 1)\left(\mathbb{E}_{s_t}\left[\bar{A}(s_t) \mid c_t = 0\right] - \mathbb{E}_{s_t}\left[\bar{A}(s_t) \mid c_t > 0\right]\right)
\end{aligned}$$

Next, note that

$$\begin{aligned}
\mathbb{E}_{s_t}\left[\bar{A}(s) \mid c_t = 0\right] &\leq \epsilon,\\
-\mathbb{E}_{s_t}\left[\bar{A}(s) \mid c_t > 0\right] &\leq \epsilon,\\
P(c_t \geq 1) &= 1 - (1-\alpha)^t.
\end{aligned} \tag{35}$$

Hence, we can bound the second term in Equation (34):

$$P(c_t \geq 1)\left(\mathbb{E}_{s\sim P(s_t|c_t\geq 1)}\left[\bar{A}(s)\right] - \mathbb{E}_{s\sim P(c_t=0)}\left[\bar{A}(s)\right]\right) \leq 2\epsilon(1-(1-\alpha)^t). \tag{36}$$

Returning to Equation (33), we get

$$\begin{aligned}
\eta(\pi_{\text{new}}) &= \eta(\pi_{\text{old}}) + \sum_t \gamma^t \left[P(c_t = 0)\mathbb{E}_{s_t}\left[\bar{A}(s) \mid c_t = 0\right] + P(c_t > 0)\mathbb{E}_{s_t}\left[\bar{A}(s) \mid c_t > 0\right]\right]\\
&\leq \eta(\pi_{\text{old}}) + \sum_t \gamma^t \left(\mathbb{E}_{s\sim P(s_t|\pi_{\text{new}}, c_t=0)}\left[\bar{A}(s)\right] + 2\epsilon(1-(1-\alpha)^t)\right)\\
&= \eta(\pi_{\text{old}}) + L_{\pi_{\text{old}}}(\pi_{\text{new}}) + \sum_t \gamma^t \cdot 2\epsilon(1-(1-\alpha)^t)\\
&= \eta(\pi_{\text{old}}) + L_{\pi_{\text{old}}}(\pi_{\text{new}}) + 2\epsilon \cdot \left(\frac{1}{1-\gamma} - \frac{1}{1-\gamma(1-\alpha)}\right)\\
&= \eta(\pi_{\text{old}}) + L_{\pi_{\text{old}}}(\pi_{\text{new}}) + \frac{2\epsilon\gamma}{(1-\gamma)(1-(1-\alpha)\gamma)}
\end{aligned} \tag{37}$$

That concludes Kakade and Langford's proof.

Now we will adapt their proof to the case that π_{new} is not necessarily a mixture involving π_{old}, however, the total variation divergence $D_{\text{TV}}^{\max}(\pi_{\text{old}}, \pi_{\text{new}})$ is bounded. The basic idea is to couple the old and new policies so that they choose the same action with probability $1-\alpha$. Then the same line of reasoning will be applied.

See [29] for an exposition on couplings. We will use the following fundamental result:

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 [29], Proposition 4.7. This joint distribution is constructed as follows:

(i) With probability $1-\alpha$, we sample $X = Y$ from the distribution $\min(p_X, p_Y)/(1-\alpha)$.

(ii) With probability α, we sample X from $\max(p_X - p_Y, 0)/\alpha$ and sample Y from $\max(p_Y - p_X, 0)/\alpha$.

Proof of Theorem 1. Let $a_{\pi_{\text{old}},t}, a_{\pi_{\text{new}},t}$ be random variables which represent the actions chosen by policies π_{old} and π_{new} at time t. By the preceding results, we can define $a_{\pi_{\text{old}},t}, a_{\pi_{\text{new}},t}$ on a common probability space so that $P(a_{\pi_{\text{old}},t} \neq a_{\pi_{\text{new}},t}) = D_{TV}(\pi_{\text{old}}(\cdot|s_t) \| \pi_{\text{new}}(\cdot|s_t)) \leq \alpha$.

Now consider sampling a trajectory from the MDP as follows. At each timestep, draw the actions $a_{\pi_{\text{old}},t}, a_{\pi_{\text{new}},t}$ from the coupled distribution described above. Perform the action $a_{\pi_{\text{new}},t}$. Define c_t as the number of times $t' < t$ where case (ii) is chosen, i.e., $a_{\pi_{\text{old}},t}, a_{\pi_{\text{new}},t}$. Next we redefine $\epsilon := \max_s \max_a |A_\pi(s,a)|$ in a slightly weaker way. (See Appendix C for an stronger definition.) Then Equations (35), (36), (37) imply the result.

□

C Perturbation Theory Proof of Policy Improvement Bound

We also provide a different proof of Theorem 1 using perturbation theory. This method makes it possible to provide slightly stronger bounds.

Theorem 1a. *Let α denote the maximum total variation divergence between stochastic policies π and $\tilde{\pi}$, as defined in Equation* (15)*, and let L be defined as in Equation* (8)*. Then*

$$\eta(\tilde{\pi}) \leq L(\tilde{\pi}) + \alpha^2 \frac{2\gamma\epsilon}{(1-\gamma)^2} \tag{38}$$

where

$$\epsilon = \max_s \left\{ \frac{\sum_a (\tilde{\pi}(a|s) Q_\pi(s,a) - \pi(a|s) Q_\pi(s,a))}{\sum_a |\tilde{\pi}(a|s) - \pi(a|s)|} \right\} \tag{39}$$

Note that the ϵ defined in Theorem 39 is less than or equal to the ϵ defined in Theorem 1, i.e., Equation 13. So Theorem 1a is slightly stronger.

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 c (a cost function on state space) is a dual vector (i.e., linear functional on vectors), thus $c\rho$ is a scalar meaning the expected cost under density ρ. Note that $\eta(\pi) = cG\rho_0$, and $\eta(\tilde{\pi}) = c\tilde{G}\rho_0$. Let $\Delta = P_{\tilde{\pi}} - P_\pi$. We wish to bound $\eta(\tilde{\pi}) - \eta(\pi) = c(\tilde{G} - G)\rho_0$. We start with some standard perturbation theory manipulations.

$$\begin{aligned} G^{-1} - \tilde{G}^{-1} &= (1 - \gamma P_\pi) - (1 - \gamma P_{\tilde{\pi}}) \\ &= \gamma \Delta. \end{aligned} \tag{40}$$

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

$$\begin{aligned} \tilde{G} - G &= \gamma G \Delta \tilde{G} \\ \tilde{G} &= G + \gamma G \Delta \tilde{G} \end{aligned} \tag{41}$$

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

$$\tilde{G} = G + \gamma G \Delta G + \gamma^2 G \Delta G \Delta \tilde{G} \tag{42}$$

So we have

$$\eta(\tilde{\pi}) - \eta(\pi) = c(\tilde{G} - G)\rho = \gamma c G \Delta G \rho_0 + \gamma^2 c G \Delta G \Delta \tilde{G} \rho_0 \tag{43}$$

Let us first consider the leading term $\gamma c G \Delta G \rho_0$. Note that $cG = v$, i.e., the infinite-horizon cost-to-go 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 $L(\tilde{\pi}) - L(\pi)$.

$$
\begin{aligned}
L(\tilde{\pi}) - L(\pi) &= \sum_s \rho_\pi(s) \sum_a (\tilde{\pi}(a|s) - \pi(a|s)) A_\pi(s,a) \\
&= \sum_s \rho_\pi(s) \sum_a \left(\pi_\theta(a|s) - \pi_{\tilde{\theta}}(a|s)\right) \left[c(s) + \sum_{s'} p(s'|s,a)\gamma v(s') - v(s) \right] \\
&= \sum_s \rho_\pi(s) \sum_{s'} \sum_a \left(\pi(a|s) - \tilde{\pi}(a|s)\right) p(s'|s,a)\gamma v(s') \\
&= \sum_s \rho_\pi(s) \sum_{s'} (p_\pi(s'|s) - p_{\tilde{\pi}}(s'|s))\gamma v(s') \\
&= \gamma v \Delta \rho_\pi \qquad (44)
\end{aligned}
$$

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

$$
\begin{aligned}
(\gamma v\Delta)_s &= \sum_a (\tilde{\pi}(s,a) - \pi(s,a)) Q_\pi(s,a) \\
&= \sum_a |\tilde{\pi}(a|s) - \pi(a|s)| \frac{\sum_a (\tilde{\pi}(s,a) - \pi(s,a)) Q_\pi(s,a)}{\sum_a |\tilde{\pi}(a|s) - \pi(a|s)|} \\
&\le \alpha\epsilon \qquad (45)
\end{aligned}
$$

We bound the other portion $G\Delta\tilde{G}\rho$ using the ℓ_1 operator norm

$$
\|A\|_1 = \sup_\rho \left\{ \frac{\|A\rho\|_1}{\|\rho\|_1} \right\} \qquad (46)
$$

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

$$
\begin{aligned}
\|G\Delta\tilde{G}\rho\|_1 &\le \|G\|_1 \|\Delta\|_1 \|\tilde{G}\|_1 \|\rho\|_1 \\
&= \frac{1}{1-\gamma} \cdot \alpha \cdot \frac{1}{1-\gamma} \cdot 1 \qquad (47)
\end{aligned}
$$

So we have that

$$
\begin{aligned}
\gamma^2 cG\Delta G\Delta\tilde{G}\rho &\le \gamma \|\gamma cG\Delta\|_\infty \|G\Delta\tilde{G}\rho\|_1 \\
&\le \gamma \cdot \alpha\epsilon \cdot \frac{2\alpha}{(1-\gamma)^2} \\
&= \alpha^2 \frac{2\gamma\epsilon}{(1-\gamma)^2} \qquad (48)
\end{aligned}
$$

□

D Efficiently Solving the Trust-Region Constrained Optimization Problem

Consider the constrained optimization problem we solve on each policy update. We will first describe how to use the analytic Fisher information and the conjugate gradient algorithm to take a natural gradient step. Then we will describe how to approximately solve the trust region optimization problem by taking one or more conjugate gradient steps.

Recall the natural policy gradient takes the quadratic approximation to $\overline{D}_{\mathrm{KL}}(\theta, \theta_{\mathrm{old}})$ and then takes the step $\theta_{\mathrm{new}} = \theta_{\mathrm{old}} - A^{-1}g$, where

$$
g = \nabla_\theta \hat{L}(\theta)\big|_{\theta=\theta_{\mathrm{old}}}, \qquad (49)
$$

$$
A = \frac{\partial^2}{\partial\theta^2} \overline{D}_{\mathrm{KL}}(\theta_{\mathrm{old}}, \theta)\big|_{\theta=\theta_{\mathrm{old}}} \qquad (50)
$$

where $\hat{L}(\theta)$ is empirical estimate of $L(\theta)$ (see Equation (29)). We can compute $A^{-1}g$ approximately without actually forming the matrix A using the conjugate gradient algorithm, where we only use A by implicitly taking Hessian-vector products Ax, and we perform several iterations to approximately solve $Ax = g$. See [16], chapter 8 for a discussion of automatic differentiation and using conjugate gradient to take a Newton step, and see [30] for a recent discussion of natural gradient in neural networks.

The analytic Hessian-vector product can be computed with the help of an automatic differentiation package that does R-mode differentiation. We used Theano [31]. The procedure for obtaining an analytic Hessian-vector product for a function f is as follows. (We specifically use $f(\theta) = \overline{D}_{\mathrm{KL}}(\theta_{\mathrm{old}}, \theta)$.)

- Construct the analytical expression for $f(\theta)$.
- Differentiate it to obtain the computation graph for $\nabla_\theta f(\theta)$.
- Construct the scalar-valued "gradient-vector product" function $gvp(\theta, p) = \nabla_\theta f(\theta) \cdot p$, where p is a vector argument with $\dim p = \dim \theta$.
- Differentiate gvp with respect to θ it to get the analytic Hessian-vector product $hvp(\theta, p) = \nabla_\theta(\nabla_\theta f(\theta) \cdot p) = (\partial_\theta^2 f(\theta)) \cdot p = Ap$.

One can also compute a Hessian-vector product by numerical differentiation, using the fact that it is the directional derivative of the gradient. However, we found that the errors due to finite differencing caused problems with the conjugate gradient algorithm.

Next, suppose we are given a desired step size δ and we would like to take a step so that $\overline{D}_{\mathrm{KL}}(\theta_{\mathrm{old}}, \theta_{\mathrm{new}}) \approx \frac{1}{2}(\theta_{\mathrm{new}} - \theta_{\mathrm{old}})^T A(\theta_{\mathrm{new}} - \theta_{\mathrm{old}}) = \delta$. The procedure described above allows us to compute a step $s = -A^{-1}g$. We would like to scale this step appropriately. Let $\theta_{\mathrm{new}} = \theta_{\mathrm{old}} + \beta s$. Then we just need to ensure that $\frac{1}{2}(\beta s)^T A(\beta s) = \delta$, i.e., $\beta = \sqrt{2\delta / s^T A s}$. After computing s, we compute the Hessian-vector product As, and then we can compute the dot product $s^T \cdot As$. Thus we obtain a step βs that gives the right change in KL divergence.

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 ($\overline{D}_{\mathrm{KL}}$)	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	$\approx$ 30 hrs
SP: computation time	$\approx$ 30 hrs

Table 3: Parameters used for Atari domain.