Learning Complex Neural Network Policies with Trajectory Optimization

Sergey Levine SVLEVINE@CS.STANFORD.EDU
Computer Science Department, Stanford University, Stanford, CA 94305 USA

Vladlen Koltun VLADLEN@ADOBE.COM
Adobe Research, San Francisco, CA 94103 USA

Abstract

Direct policy search methods offer the promise of automatically learning controllers for complex, high-dimensional tasks. However, prior applications of policy search often required specialized, low-dimensional policy classes, limiting their generality. In this work, we introduce a policy search algorithm that can directly learn high-dimensional, general-purpose policies, represented by neural networks. We formulate the policy search problem as an optimization over trajectory distributions, alternating between optimizing the policy to match the trajectories, and optimizing the trajectories to match the policy and minimize expected cost. Our method can learn policies for complex tasks such as bipedal push recovery and walking on uneven terrain, while outperforming prior methods.

1. Introduction

Direct policy search offers the promise of automatically learning controllers for complex, high-dimensional tasks. It has seen applications in fields ranging from robotics (Peters & Schaal, 2008; Theodorou et al., 2010; Deisenroth et al., 2013; Kober et al., 2013) and autonomous flight (Ross et al., 2013) to energy generation (Kolter et al., 2012). However, existing policy search methods usually require the policy class to be chosen carefully, so that a good policy can be found without falling into poor local optima. Research into new, specialized policy classes is an active area that has provided substantial improvements on real-world systems (Ijspeert et al., 2003; Paraschos et al., 2013). This specialization is necessary because most model-free policy search methods can only feasibly be applied to policies with a few hundred parameters (Deisenroth et al., 2013). Such specialized policy classes are limited in the types of behaviors they can represent, and engineering new policy classes requires considerable effort.

In recent work, we introduced a new class of policy search algorithms that can learn much more complex policies by using model-based trajectory optimization to guide the policy search (Levine & Koltun, 2013a;b). By optimizing trajectories in tandem with the policy, guided policy search methods combine the flexibility of trajectory optimization with the generality of policy search. These methods can scale to highly complex policy classes and can be used to train general-purpose neural network controllers that do not require task-specific engineering. Furthermore, the training trajectories can be initialized with examples for learning from demonstration.

A key challenge in guided policy search is ensuring that the trajectories are useful for learning the policy, since not all trajectories can be realized by policies from a particular policy class. For example, a policy provided with partial observations cannot make decisions based on unobserved state variables. In this paper, we present a constrained guided policy search algorithm that gradually brings the trajectories into agreement with the policy, ensuring that the trajectories and policy match at convergence. This is accomplished by gradually enforcing a constraint between the trajectories and the policy using dual gradient descent, resulting in an algorithm that iterates between optimizing the trajectories to minimize cost and agree with the policy, optimizing the policy to agree with the trajectories, and updating the dual variables to improve constraint satisfaction.

By enforcing agreement between the policy and the trajectories, our algorithm can discover policies for highly complex behaviors. We evaluate our approach on a set of challenging locomotion tasks, including a push recovery task that requires the policy to combine multiple recovery strategies learned in parallel from multiple trajectories. Our approach successfully learned a policy that could not only perform multiple different recoveries, but could also correctly choose the best strategy under new conditions.

Proceedings of the 31^{st} International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32.

2. Preliminaries and Overview

Policy search is an optimization over the parameters θ of a policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$, which is a distribution over actions $\mathbf{u}_t$ conditioned on states $\mathbf{x}_t$, with respect to the expected value of a cost function $\ell(\mathbf{x}_t, \mathbf{u}_t)$, denoted $E_{\pi_\theta}[\sum_{t=1}^{T} \ell(\mathbf{x}_t, \mathbf{u}_t)]$. The expectation is taken under the policy and the system dynamics $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$, and together they induce a distribution over trajectories. We will therefore abbreviate the expectation as $E_{\pi_\theta}[\ell(\tau)]$, where $\tau = (\mathbf{x}_{1..T}, \mathbf{u}_{1..T})$ denotes a trajectory. In continuous spaces, this expectation cannot be computed exactly, since the number of states is infinite. Many policy search methods approximate this quantity, typically by sampling (Peters & Schaal, 2008).

Sampling-based policy search methods do not need or use the dynamics distribution $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$. However, in practice this advantage often also becomes a weakness: without the use of a system model, the policy search is forced to rely on "trial and error" exploration strategies. While this works wells for simple problems, for example when either the state space or the dimensionality of θ is small, such model-free methods often do not scale to policies with more than a few hundred parameters (Deisenroth et al., 2013). Scaling policy search up to powerful, expressive, general-purpose policy classes, such as large neural networks, is extremely challenging with such methods.

When a dynamics model is available, trajectories can be optimized directly with respect to their actions, without a parametric policy.[1] Trajectory optimization is easier than general policy search, because the policy parameters couple the actions at different time steps. Our constrained guided policy search algorithm employs trajectory optimization to guide the policy search process, avoiding the need for "trial and error" random exploration. The algorithm alternates between optimizing a set of trajectories to minimize cost and match the current policy, and optimizing the policy to follow the actions in each trajectory. However, simply training a policy on individual trajectories usually fails to produce effective policies, since a small error at each time step can quickly compound and place the policy in costly, unexplored parts of the space (Ross et al., 2011).

To avoid compounding errors, the policy must be trained on data sampled from a distribution over states. The ideal distribution is the one induced by the optimal policy, but it is unknown. The initial policy often has a broad state distribution that visits very costly states, where it is inefficient and unnecessary to determine the optimal actions. Instead, we train the policy on distributions over good trajectories, denoted $q(\tau)$, which are produced using trajectory optimization. Alternating policy and trajectory optimization, $q(\tau)$ and $\pi_\theta(\tau)$ are gradually brought into agreement, so that the final policy is trained on its own state distribution.

Since $q(\tau)$ may not match $\pi_\theta(\tau)$ before convergence, we make $q(\tau)$ as broad as possible, so that the policy learns stable feedbacks from a wide range of states, reducing the chance that compounding errors will place it into unexplored regions. To that end, we use the maximum entropy objective $E_q[\ell(\tau)] - \mathcal{H}(q(\tau))$, which has previously been proposed for control and reinforcement learning (Todorov, 2006; Ziebart, 2010; Kappen et al., 2012).

This objective minimizes cost and maximizes entropy, producing broad distributions over good trajectories. It corresponds to the KL-divergence $D_{\mathrm{KL}}(q(\tau)\|\rho(\tau))$, where $\rho(\tau) \propto \exp(\ell(\tau))$, making $q(\tau)$ an I-projection of $\rho(\tau)$. In the absence of policy constraints, a Gaussian $q(\tau)$ can be approximately optimized by a variant of the iLQG algorithm (Todorov & Li, 2005), as described in previous work (Levine & Koltun, 2013b). In the next section, we derive a similar algorithm that also gradually enforces a constraint on the action conditionals $q(\mathbf{u}_t|\mathbf{x}_t)$, to force them to match the policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ at convergence. As with iLQG, our trajectory optimization algorithm uses a local linearization of the dynamics and a quadratic expansion of the cost, which corresponds to using a Laplace approximation of $\rho(\tau)$. We solve the constrained problem by optimizing its Lagrangian with respect to $q(\tau)$ and θ, and iteratively updating the Lagrange multipliers by means of dual gradient descent (Boyd & Vandenberghe, 2004).

3. Policy Search via Trajectory Optimization

We begin by reformulating the policy optimization task as the following constrained problem:

$$\begin{aligned} \min_{\theta, q(\tau)} \; & D_{\mathrm{KL}}(q(\tau)\|\rho(\tau)) \\ \text{s.t. } & q(\mathbf{x}_1) = p(\mathbf{x}_1), \\ & q(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t), \\ & D_{\mathrm{KL}}(q(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)\|q(\mathbf{x}_t, \mathbf{u}_t)) = 0. \end{aligned} \tag{1}$$

The first two constraints ensure that the distribution $q(\tau)$ is consistent with the domain's initial state distribution and dynamics, and are enforced implicitly by our trajectory optimization algorithm. The last constraint ensures that the conditional action distributions $q(\mathbf{u}_t|\mathbf{x}_t)$ match the policy distribution $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$. Since the action distributions fully determine the state distribution, $q(\tau)$ and $\pi_\theta(\tau)$ become identical when the constraints are satisfied, making the constrained optimization equivalent to optimizing the policy with respect to $D_{\mathrm{KL}}(\pi_\theta(\tau)\|\rho(\tau))$. This objective differs from the expected cost but, as discussed in the previous section, it provides for more reasonable handling of trajectory distributions prior to convergence. In practice, a solution

[1] Action sequences can be viewed as open-loop policies, and linear feedback can be added to turn them into nonstationary closed-loop policies.

with very good expected cost can be obtained by increasing the magnitude of the cost over the course of the optimization. As this magnitude goes to infinity, the entropy term becomes irrelevant, though a good deterministic policy can usually be obtained by taking the mean of the stochastic policy optimized under even a moderate cost magnitude.

Our method approximately optimizes Equation 1 with dual gradient descent (Boyd & Vandenberghe, 2004) and local linearization, leading to an iterative algorithm that alternates between optimizing one or more Gaussian distributions $q_i(\tau)$ with dynamic programming, and optimizing the policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ to match $q(\mathbf{u}_t|\mathbf{x}_t)$. A separate Gaussian $q_i(\tau)$ is used for each initial condition (for example when learning to control a bipedal walker that must respond to different lateral pushes), but because the trajectories are only coupled by the policy parameters θ we will drop the subscript i in our derivation and consider just a single $q(\tau)$.

We first write the Lagrangian of Equation 1, omitting the implicitly enforced initial state and dynamics constraints:

$$\mathcal{L}(\theta, q, \lambda) = D_{\mathrm{KL}}(q(\tau)\|\rho(\tau)) + \sum_{t=1}^{T} \lambda_t D_{\mathrm{KL}}(q(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)\|q(\mathbf{x}_t, \mathbf{u}_t)).$$

Dual gradient descent alternates between optimizing $\mathcal{L}$ with respect to $q(\tau)$ and θ, and updating the dual variables λ_t with subgradient descent, using a step size η:

$$\lambda_t \leftarrow \lambda_t + \eta D_{\mathrm{KL}}(q(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)\|q(\mathbf{x}_t, \mathbf{u}_t)). \quad (2)$$

The KL-divergence is estimated from samples, and the inner optimization over $q(\tau)$ and θ is performed in alternating fashion, first over each $q(\tau)$ and then over θ. Although neither the objective nor the constraints are in general convex, we found this approach to yield a good local optimum. The full method is summarized in Algorithm 1. We initialize each trajectory on line 1, either with unconstrained trajectory optimization or from example demonstrations. We then optimize the policy for K iterations of dual gradient descent. In each iteration, we optimize each trajectory distribution $q_i(\tau)$ on line 3, using a few iterations of the algorithm described in Section 3.1. This step can be parallelized over all trajectories. We then optimize the policy to match all of the distributions $q_i(\tau)$ on line 4, using a simple supervised learning procedure described in Section 3.2. Finally, we update the dual variables according to Equation 2.

Algorithm 1 Constrained guided policy search

1: Initialize the trajectories $\{q_1(\tau), \ldots, q_M(\tau)\}$
2: **for** iteration $k = 1$ to K **do**
3: Optimize each $q_i(\tau)$ with respect to $\mathcal{L}(\theta, q_i(\tau), \lambda_i)$
4: Optimize θ with respect to $\sum_{i=1}^{M} \mathcal{L}(\theta, q_i(\tau), \lambda_i)$
5: Update dual variables λ using Equation 2
6: **end for**
7: **return** optimized policy parameters θ

3.1. Trajectory optimization

The trajectory optimization phase optimizes each $q_i(\tau)$ with respect to the Lagrangian $\mathcal{L}(\theta, q_i(\tau), \lambda_i)$. Since the trajectories can be optimized independently, we again drop the subscript i in this section. Similarly to iLQG, the optimization uses locally linearized dynamics, though the policy KL-divergence constraints necessitate a novel algorithm. One iteration of this trajectory optimization algorithm is summarized in Algorithm 2.

Each $q(\tau)$ has a mean $\hat{\tau} = (\hat{\mathbf{x}}_{1..T}, \hat{\mathbf{u}}_{1..T})$, a conditional action distribution $q(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{u}_t + \mathbf{K}\mathbf{x}_t, \mathbf{A}_t)$ at each time step, and dynamics $q(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) = \mathcal{N}(f_{\mathbf{x}t}\mathbf{x}_t + f_{\mathbf{u}t}\mathbf{u}_t, \mathbf{F}_t)$, which corresponds to locally linearized Gaussian dynamics with mean $f_t(\mathbf{x}_t, \mathbf{u}_t)$ and covariance $\mathbf{F}_t$ (subscripts in $f_{\mathbf{x}t}$ and $f_{\mathbf{u}t}$ denote derivatives). We will assume without loss of generality that all $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{u}}_t$ are initially zero, and $\mathbf{x}_t$ and $\mathbf{u}_t$ denote a perturbation from a nominal trajectory, which is updated at every iteration. Given this definition of $q(\tau)$, we can rewrite the objective as

$$\mathcal{L}(q) = \sum_{t=1}^{T} E_{q(\mathbf{x}_t, \mathbf{u}_t)}[\ell(\mathbf{x}_t, \mathbf{u}_t)] - \frac{1}{2}\log|\mathbf{A}_t| + \lambda_t E_{q(\mathbf{x}_t)}[D_{\mathrm{KL}}(\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)\|q(\mathbf{u}_t|\mathbf{x}_t))].$$

We can evaluate both expectations with the Laplace approximation, which models the policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ as a (locally) linear Gaussian with mean $\mu_t^\pi(\mathbf{x}_t)$ and covariance Σ_t^π, and the cost with its first and second derivatives $\ell_{\mathbf{xu}t}$ and $\ell_{\mathbf{xu},\mathbf{xu}t}$ (subscripts again denote derivatives, in this case twice with respect to both the state and action):

$$\mathcal{L}(q) \approx \sum_{t=1}^{T} \frac{1}{2}\begin{bmatrix}\hat{\mathbf{x}}_t \\ \hat{\mathbf{u}}_t\end{bmatrix}^{\mathrm{T}} \ell_{\mathbf{xu},\mathbf{xu}t}\begin{bmatrix}\hat{\mathbf{x}}_t \\ \hat{\mathbf{u}}_t\end{bmatrix} + \begin{bmatrix}\hat{\mathbf{x}}_t \\ \hat{\mathbf{u}}_t\end{bmatrix}^{\mathrm{T}} \ell_{\mathbf{xu}t} + \frac{1}{2}\mathrm{tr}\left(\Sigma_t \ell_{\mathbf{xu},\mathbf{xu}t}\right) - \frac{1}{2}\log|\mathbf{A}_t| + \frac{\lambda_t}{2}\log|\mathbf{A}_t| + \frac{\lambda_t}{2}(\hat{\mathbf{u}}_t - \mu_t^\pi(\hat{\mathbf{x}}_t))^{\mathrm{T}}\mathbf{A}_t^{-1}(\hat{\mathbf{u}}_t - \mu_t^\pi(\hat{\mathbf{x}}_t)) + \frac{\lambda_t}{2}\mathrm{tr}\left(\mathbf{A}_t^{-1}\Sigma_t^\pi\right) + \frac{\lambda_t}{2}\mathrm{tr}\left(\mathbf{S}_t\left(\mathbf{K}_t - \mu_{\mathbf{x}t}^\pi(\hat{\mathbf{x}}_t)\right)^{\mathrm{T}}\mathbf{A}_t^{-1}\left(\mathbf{K}_t - \mu_{\mathbf{x}t}^\pi(\hat{\mathbf{x}}_t)\right)\right),$$

where constants are omitted, $\mu_{\mathbf{x}t}^\pi(\hat{\mathbf{x}}_t)$ is the gradient of the policy mean at $\hat{\mathbf{x}}_t$, and Σ_t denotes the joint marginal covariance over states and actions in $q(\mathbf{x}_t, \mathbf{u}_t)$. We use $\mathbf{S}_t$ to refer to the covariance of $q(\mathbf{x}_t)$ and $\mathbf{A}_t$ to refer to the *conditional* covariance of $q(\mathbf{u}_t|\mathbf{x}_t)$.

Each iteration of trajectory optimization first forms this approximate Lagrangian by computing the derivatives of the dynamics and cost function on line 1. A dynamic programming algorithm then computes the gradients and Hessians

Algorithm 2 Trajectory optimization iteration

1: Compute $f_{\mathbf{xu}t}$, $\mu^{\pi}_{\mathbf{x}t}$, $\ell_{\mathbf{xu}t}$, $\ell_{\mathbf{xu},\mathbf{xu}t}$ around $\hat{\tau}$
2: **for** $t = T$ to 1 **do**
3: Compute $\mathbf{K}_t$ and $\mathbf{k}_t$ using Equations 3 and 4
4: Compute $\mathcal{L}_{\mathbf{x}t}$ and $\mathcal{L}_{\mathbf{x},\mathbf{x}t}$ using Equation 5
5: **end for**
6: Initialize $\alpha \leftarrow 1$
7: **repeat**
8: Obtain new trajectory $\hat{\tau}'$ using $\mathbf{u}_t = \alpha\mathbf{k}_t + \mathbf{K}_t\mathbf{x}_t$
9: Decrease step size α
10: **until** $\mathcal{L}(\hat{\tau}', \mathbf{K}, \mathbf{A}) > \mathcal{L}(\hat{\tau}, \mathbf{K}, \mathbf{A})$
11: Compute $f_{\mathbf{xu}t}$, $\mu^{\pi}_{\mathbf{x}t}$, $\ell_{\mathbf{xu}t}$, $\ell_{\mathbf{xu},\mathbf{xu}t}$ around $\hat{\tau}'$
12: **repeat**
13: Compute $\mathbf{S}_t$ using current $\mathbf{A}_t$ and $\mathbf{K}_t$
14: **for** $t = T$ to 1 **do**
15: **repeat**
16: Compute $\mathbf{K}_t$ using Equation 7
17: Compute $\mathbf{A}_t$ by solving CARE in Equation 8
18: **until** $\mathbf{A}_t$ and $\mathbf{K}_t$ converge (about 5 iterations)
19: Compute $\mathcal{L}_{\mathbf{S}t}$ using Equation 6
20: **end for**
21: **until** all $\mathbf{S}_t$ and $\mathbf{A}_t$ converge (about 2 iterations)
22: **return** new mean $\hat{\tau}'$ and covariance terms $\mathbf{A}_t$, $\mathbf{K}_t$

with respect to the mean state and action at each time step (keeping $\mathbf{A}_t$ fixed), as summarized on lines 2-5, allowing us to take a Newton-like step by multiplying the gradient by the inverse Hessian, analogously to iLQG (Todorov & Li, 2005). The action then becomes a linear function of the corresponding state, resulting in linear feedback terms $\mathbf{K}_t$. After taking this Newton-like step, we perform a line search to ensure improvement on lines 7-10, and then update $\mathbf{A}_t$ at each time step on lines 12-21. To derive the gradients and Hessians, it will be convenient to define the following quantities, which incorporate information about future costs analogously to the Q-function:

$$Q_{\mathbf{xu}t} = \ell_{\mathbf{xu}t} + f_{\mathbf{xu}}^{\mathrm{T}}\mathcal{L}_{\mathbf{x}t+1}$$
$$Q_{\mathbf{xu},\mathbf{xu}t} = \ell_{\mathbf{xu},\mathbf{xu}t} + f_{\mathbf{xu}}^{\mathrm{T}}\mathcal{L}_{\mathbf{x},\mathbf{x}t+1}f_{\mathbf{xu}},$$

where the double subscripts $\mathbf{xu}$ again denote derivatives with respect to $(\mathbf{x}_t, \mathbf{u}_t)^{\mathrm{T}}$, and $\mathcal{L}_{\mathbf{x}t+1}$ and $\mathcal{L}_{\mathbf{x},\mathbf{x}t+1}$ are the gradient and Hessian of the objective with respect to $\hat{\mathbf{x}}_{t+1}$. As with iLQG, we assume locally linear dynamics and ignore the higher order dynamics derivatives. Proceeding recursively backwards through time, the first and second derivatives with respect to $\hat{\mathbf{u}}_t$ are then given by

$$\mathcal{L}_{\mathbf{u}t} = Q_{\mathbf{u},\mathbf{u}t}\hat{\mathbf{u}}_t + Q_{\mathbf{u},\mathbf{x}t}\hat{\mathbf{x}}_t + Q_{\mathbf{u}t} + \lambda_t\mathbf{A}_t^{-1}(\hat{\mathbf{u}}_t - \mu^{\pi}_t(\hat{\mathbf{x}}_t))$$
$$\mathcal{L}_{\mathbf{u},\mathbf{u}t} = Q_{\mathbf{u},\mathbf{u}t} + \lambda_t\mathbf{A}_t^{-1},$$

where the application of the chain rule to include the effect of $\hat{\mathbf{u}}_t$ on subsequent time steps is subsumed inside $Q_{\mathbf{u}t}$, $Q_{\mathbf{u},\mathbf{u}t}$, and $Q_{\mathbf{u},\mathbf{x}t}$. Since we assume that $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{u}}_t$ are both zero, we can solve for the optimal change to the action $\mathbf{k}_t$:

$$\mathbf{k}_t = -\left(Q_{\mathbf{u},\mathbf{u}t} + \lambda_t\mathbf{A}_t^{-1}\right)^{-1}\left(Q_{\mathbf{u}t} - \lambda_t\mathbf{A}_t^{-1}\mu^{\pi}_t(\hat{\mathbf{x}}_t)\right). \quad (3)$$

The feedback $\mathbf{K}_t$ is the derivative of $\mathbf{k}_t$ with respect to $\hat{\mathbf{x}}_t$:

$$\mathbf{K}_t = -\left(Q_{\mathbf{u},\mathbf{u}t} + \lambda_t\mathbf{A}_t^{-1}\right)^{-1}\left(Q_{\mathbf{u},\mathbf{x}t} - \lambda_t\mathbf{A}_t^{-1}\mu^{\pi}_{\mathbf{x}t}(\hat{\mathbf{x}}_t)\right). \quad (4)$$

To complete the dynamic programming step, we can now differentiate the objective with respect to $\hat{\mathbf{x}}_t$, treating $\hat{\mathbf{u}}_t = \mathbf{k}_t + \mathbf{K}_t\hat{\mathbf{x}}_t$ as a function of $\hat{\mathbf{x}}_t$:

$$\begin{aligned}
\mathcal{L}_{\mathbf{x}t} &= Q_{\mathbf{x},\mathbf{x}t}\hat{\mathbf{x}}_t + Q_{\mathbf{x},\mathbf{u}t}(\mathbf{k}_t + \mathbf{K}_t\hat{\mathbf{x}}_t) + \mathbf{K}_t^{\mathrm{T}}Q_{\mathbf{u},\mathbf{x}t}\hat{\mathbf{x}}_t + \\
&\quad \mathbf{K}_t^{\mathrm{T}}Q_{\mathbf{u},\mathbf{u}t}(\mathbf{K}_t\hat{\mathbf{x}}_t + \mathbf{k}_t) + Q_{\mathbf{x}t} + \mathbf{K}_t^{\mathrm{T}}Q_{\mathbf{u}t} + \\
&\quad \lambda_t(\mathbf{K}_t - \mu^{\pi}_{\mathbf{x}t}(\hat{\mathbf{x}}_t))^{\mathrm{T}}\mathbf{A}_t^{-1}(\mathbf{K}_t\hat{\mathbf{x}}_t + \mathbf{k}_t - \mu^{\pi}_t(\hat{\mathbf{x}}_t)) \\
&= Q_{\mathbf{x},\mathbf{u}t}\mathbf{k}_t + \mathbf{K}_t^{\mathrm{T}}Q_{\mathbf{u},\mathbf{u}t}\mathbf{k}_t + Q_{\mathbf{x}t} + \mathbf{K}_t^{\mathrm{T}}Q_{\mathbf{u}t} + \\
&\quad \lambda_t(\mathbf{K}_t - \mu^{\pi}_{\mathbf{x}t}(\hat{\mathbf{x}}_t))^{\mathrm{T}}\mathbf{A}_t^{-1}(\mathbf{k}_t - \mu^{\pi}_t(\hat{\mathbf{x}}_t)) \\
\mathcal{L}_{\mathbf{x},\mathbf{x}t} &= Q_{\mathbf{x},\mathbf{x}t} + Q_{\mathbf{x},\mathbf{u}t}\mathbf{K}_t + \mathbf{K}_t^{\mathrm{T}}Q_{\mathbf{u},\mathbf{x}t} + \mathbf{K}_t^{\mathrm{T}}Q_{\mathbf{u},\mathbf{u}t}\mathbf{K}_t + \\
&\quad \lambda_t(\mathbf{K}_t - \mu^{\pi}_{\mathbf{x}t}(\hat{\mathbf{x}}_t))^{\mathrm{T}}\mathbf{A}_t^{-1}(\mathbf{K}_t - \mu^{\pi}_{\mathbf{x}t}(\hat{\mathbf{x}}_t)), \quad (5)
\end{aligned}$$

where the simplification again happens because $\hat{\mathbf{x}}_t$ is zero.

Once we compute $\mathbf{k}_t$ and $\mathbf{K}_t$ at each time step, we perform a simulator rollout using the deterministic policy $\mathbf{u}_t = \mathbf{k}_t + \mathbf{K}_t\mathbf{x}_t$ to obtain a new mean nominal trajectory. Since the dynamics may deviate from the previous linearization far from the previous trajectory, we perform a line search on $\mathbf{k}_t$ to ensure that the objective improves as a function of the new mean, as summarized on lines 7-10.

Once the new mean is found, both the policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ and the dynamics are relinearized around the new nominal trajectory on line 2, and we perform a second dynamic programming pass to update the covariance $\mathbf{A}_t$ and feedback terms $\mathbf{K}_t$. As before, we introduce a variable that incorporates gradient information from future time steps:

$$Q_{\mathbf{xu},\mathbf{xu}t} = \ell_{\mathbf{xu},\mathbf{xu}t} + 2f_{\mathbf{xu}t}^{\mathrm{T}}\mathcal{L}_{\mathbf{S}t+1}f_{\mathbf{xu}t}.$$

This equation is obtained by using the chain rule to include the effect of the covariance Σ_t on future time steps using the covariance dynamics $\mathbf{S}_{t+1} = f_{\mathbf{xu}t}\Sigma_t f_{\mathbf{xu}t}^{\mathrm{T}} + \mathbf{F}_t$, so that

$$\frac{1}{2}\mathrm{tr}\left(\Sigma_t\ell_{\mathbf{xu},\mathbf{xu}t}\right) + \frac{\partial\mathbf{S}_{t+1}}{\partial\Sigma_t}\cdot\mathcal{L}_{\mathbf{S}t+1} = \frac{1}{2}\mathrm{tr}\left(\Sigma_t Q_{\mathbf{xu},\mathbf{xu}t}\right).$$

This allows us to simply substitute $Q_{\mathbf{xu},\mathbf{xu}t}$ for $\ell_{\mathbf{xu},\mathbf{xu}t}$ in the objective to include all the effect of the covariance on future time steps. To then derive the gradient of the objective, first with respect to $\mathbf{A}_t$ and $\mathbf{K}_t$ for optimization, and then with respect to $\mathbf{S}_t$ to complete the dynamic programming step, we first note that

$$\Sigma_t = \begin{bmatrix} \mathbf{S}_t & \mathbf{S}_t\mathbf{K}_t^{\mathrm{T}} \\ \mathbf{K}_t\mathbf{S}_t & \mathbf{K}_t\mathbf{S}_t\mathbf{K}_t^{\mathrm{T}} + \mathbf{A}_t \end{bmatrix}.$$

This allows us to expand the term $\mathrm{tr}\left(\Sigma_t Q_{\mathbf{xu},\mathbf{xu}t}\right)$ to get

$$\mathrm{tr}\left(\Sigma_t Q_{\mathbf{xu},\mathbf{xu}t}\right) = \mathrm{tr}\left(\mathbf{S}_t Q_{\mathbf{x},\mathbf{x}t}\right) + \mathrm{tr}\left(\mathbf{S}_t \mathbf{K}_t^{\mathrm{T}} Q_{\mathbf{u},\mathbf{x}t}\right) + \mathrm{tr}\left(\mathbf{S}_t Q_{\mathbf{x},\mathbf{u}t} \mathbf{K}_t\right) + \mathrm{tr}\left(\mathbf{A}_t Q_{\mathbf{u},\mathbf{u}t}\right) + \mathrm{tr}\left(\mathbf{K}_t \mathbf{S}_t \mathbf{K}_t^{\mathrm{T}} Q_{\mathbf{u},\mathbf{u}t}\right).$$

Using this identity, we can obtain the derivatives of the objective with respect to $\mathbf{K}_t$, $\mathbf{A}_t$, and $\mathbf{S}_t$:

$$\begin{aligned}
\mathcal{L}_{\mathbf{A}t} &= \frac{1}{2} Q_{\mathbf{u},\mathbf{u}t} + \frac{\lambda_t - 1}{2} \mathbf{A}_t^{-1} - \frac{\lambda_t}{2} \mathbf{A}_t^{-1} \mathbf{M} \mathbf{A}_t^{-1} \\
\mathcal{L}_{\mathbf{K}t} &= Q_{\mathbf{u},\mathbf{u}t} \mathbf{K}_t \mathbf{S}_t + Q_{\mathbf{u},\mathbf{x}t} \mathbf{S}_t + \lambda_t \mathbf{A}_t^{-1} \mathbf{K}_t \mathbf{S}_t - \\
&\quad \lambda_t \mathbf{A}_t^{-1} \mu^\pi_{\mathbf{x}t}(\hat{\mathbf{x}}_t) \mathbf{S}_t \\
\mathcal{L}_{\mathbf{S}t} &= \frac{1}{2} \left[Q_{\mathbf{x},\mathbf{x}t} + \mathbf{K}_t^{\mathrm{T}} Q_{\mathbf{u},\mathbf{x}t} + Q_{\mathbf{x},\mathbf{u}t} \mathbf{K}_t + \mathbf{K}_t^{\mathrm{T}} Q_{\mathbf{u},\mathbf{u}t} \mathbf{K}_t \right. \\
&\quad \left. + \lambda_t (\mathbf{K}_t - \mu^\pi_{\mathbf{x}t}(\hat{\mathbf{x}}_t))^{\mathrm{T}} \mathbf{A}_t^{-1} (\mathbf{K}_t - \mu^\pi_{\mathbf{x}t}(\hat{\mathbf{x}}_t)) \right], \qquad (6)
\end{aligned}$$

where $\mathbf{M} = \Sigma_t^\pi + (\hat{\mathbf{u}}_t - \mu_t^\pi(\hat{\mathbf{x}}_t))(\hat{\mathbf{u}}_t - \mu_t^\pi(\hat{\mathbf{x}}_t))^{\mathrm{T}} + (\mathbf{K}_t - \mu^\pi_{\mathbf{x}t}(\hat{\mathbf{x}}_t))\mathbf{S}_t(\mathbf{K}_t - \mu^\pi_{\mathbf{x}t}(\hat{\mathbf{x}}_t))^{\mathrm{T}}$. The equation for $\mathcal{L}_{\mathbf{S}t}$ is simply half of $\mathcal{L}_{\mathbf{x},\mathbf{x}t}$, which indicates that $Q_{\mathbf{xu},\mathbf{xu}t}$ is the same as during the first backward pass. Solving for $\mathbf{K}_t$, we also obtain the same equation as before:

$$\mathbf{K}_t = -\left(Q_{\mathbf{u},\mathbf{u}t} + \lambda_t \mathbf{A}_t^{-1}\right)^{-1} \left(Q_{\mathbf{u},\mathbf{x}t} - \lambda_t \mathbf{A}_t^{-1} \mu^\pi_{\mathbf{x}t}(\hat{\mathbf{x}}_t)\right). \qquad (7)$$

To solve for $\mathbf{A}_t$, we set the derivative to zero and multiply both sides on both the left and right by $\sqrt{2}\mathbf{A}_t$ to get

$$\mathbf{A}_t Q_{\mathbf{u},\mathbf{u}t} \mathbf{A}_t + (\lambda_t - 1)\mathbf{A}_t - \lambda_t \mathbf{M} = 0. \qquad (8)$$

The above equation is a continuous-time algebraic Riccati equation (CARE),[2] and can be solved in comparable time to an eigenvalue decomposition (Arnold & Laub, 1984). Our implementation uses the MATLAB CARE solver.

Since $\mathbf{K}_t$ depends on $\mathbf{A}_t$, which itself depends on both $\mathbf{K}_t$ and $\mathbf{S}_t$, we use the old values of each quantity, and repeat the solver for several iterations. On lines 15-18, we repeatedly solve for $\mathbf{K}_t$ and $\mathbf{A}_t$ at each time step, which usually converges in a few iterations. On lines 12-21, we also repeat the entire backward pass several times to update $\mathbf{S}_t$ based on the new $\mathbf{A}_t$, which converges even faster. In practice, we found that two backward passes were sufficient, and a simple test on the maximum change in the elements of $\mathbf{K}_t$ and $\mathbf{A}_t$ can be used to determine convergence.

This derivation allows us to optimize $q(\tau)$ under a Laplace approximation. Although the Laplace approximation provides a reasonable objective, the linearized policy may not reflect the real structure of $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ in the entire region where $q(\mathbf{x}_t)$ is high. Since the policy is trained by sampling states from $q(\mathbf{x}_t)$, it optimizes a different objective. This can lead to divergence when the policy is highly nonlinear.

[2] Although such equations often come up in the context of optimal control, our use of algebraic Riccati equations is unrelated to the manner in which they are usually employed.

To solve this problem, we can estimate the policy terms with M random samples $\mathbf{x}_{ti}$ drawn from $q(\mathbf{x}_t)$, rather than by linearizing around the mean. This corresponds to Monte Carlo evaluation of the expectation of the KL-divergence under $q(\mathbf{x}_t)$, as opposed to the more crude Laplace approximation. The resulting optimization algorithm has a similar structure, with $\mu_t^\pi(\hat{\mathbf{x}}_t)$ and $\mu^\pi_{\mathbf{x}t}(\hat{\mathbf{x}}_t)$ in the derivatives of the mean replaced by their averages over the samples. The gradients of the covariance terms become more complex, though simply substituting the sample averages of μ_t^π and $\mu^\pi_{\mathbf{x}t}$ into the above algorithm works well in practice, and is somewhat faster. A full derivation of the true gradients is provided in the supplementary appendix for completeness.

3.2. Policy Optimization

Once $q(\tau)$ is fixed, the policy optimization becomes a weighted supervised learning problem. Training points $\mathbf{x}_{ti}$ are sampled from $q(\mathbf{x}_t)$ at each time step, and the policy is trained to be an I-projection onto the corresponding conditional Gaussian. For example, if the policy is conditionally Gaussian, with the mean $\mu^\pi(\mathbf{x}_t)$ and covariance $\Sigma^\pi(\mathbf{x}_t)$ being any function of $\mathbf{x}_t$, the policy objective is given by

$$\begin{aligned}
\mathcal{L}(\theta) &= \sum_{t=1}^{T} \lambda_t \sum_{i=1}^{N} D_{\mathrm{KL}}(\pi_\theta(\mathbf{u}_t|\mathbf{x}_{ti}) \| q(\mathbf{u}_t|\mathbf{x}_{ti})) \\
&= \sum_{t=1}^{T} \lambda_t \sum_{i=1}^{N} \frac{1}{2} \left\{ \mathrm{tr}(\Sigma_t^\pi(\mathbf{x}_{ti}) \mathbf{A}_t^{-1}) - \log |\Sigma^\pi(\mathbf{x}_{ti})| + \right. \\
&\quad \left. (\mathbf{K}_t \mathbf{x}_{ti} + \mathbf{k}_t - \mu^\pi(\mathbf{x}_{ti}))^{\mathrm{T}} \mathbf{A}_t^{-1} (\mathbf{K}_t \mathbf{x}_{ti} + \mathbf{k}_t - \mu^\pi(\mathbf{x}_{ti})) \right\}.
\end{aligned}$$

This is a least-squares optimization on the policy mean, with targets $\mathbf{K}_t \mathbf{x}_{ti} + \mathbf{k}$ and weight matrix $\mathbf{A}_t^{-1}$, and can be performed by standard algorithms such as stochastic gradient descent (SGD) or, as in our implementation, LBFGS.

As the constraints are satisfied, $q(\mathbf{x}_t)$ approaches $\pi_\theta(\mathbf{x}_t)$, and $q(\mathbf{u}_t|\mathbf{x}_t)$ approaches the exponential of the Q-function of $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ under the maximum entropy objective. Minimizing $D_{\mathrm{KL}}(\pi_\theta(\mathbf{u}_t|\mathbf{x}_t) \| q(\mathbf{u}_t|\mathbf{x}_t))$ therefore resembles policy iteration with an "optimistic" approximate Q-function. This is an advantage over the opposite KL-divergence (where $q(\tau)$ is an I-projection of $\pi_\theta(\tau)$) suggested in our prior work (2013b), which causes the policy to be risk-seeking by optimizing the expected *exponential* reward (negative cost). In the next section, we show that this new formulation outperforms the previous risk-seeking variant, and we discuss the distinction further in Section 5.

4. Experimental Evaluation

We evaluated our method on planar swimming and walking tasks, as well as walking on uneven terrain and recovery from strong lateral pushes. Each task was executed on a simulated robot with torque motors at the joints, using

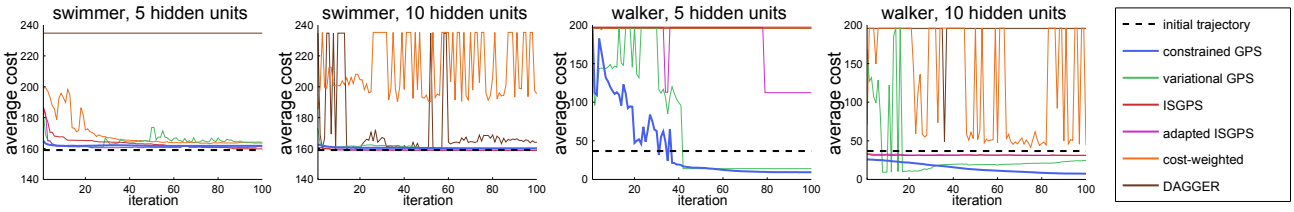

Figure 1. Comparison to prior work on swimming and walking tasks. Only constrained and variational GPS succeeded on every task. For the walker, costs significantly above the initial example indicate falling. Policies that fail and become unstable are off the scale, and are clamped to the maximum cost. Frequent vertical oscillations indicate a policy that oscillates between stable and unstable solutions.

the MuJoCo physics simulator (Todorov et al., 2012). The policies were general-purpose neural networks that mapped joint angles directly to torques at each time step. The cost function for each task consisted of a sum of three terms:

$$\ell(\mathbf{x}_t, \mathbf{u}_t) = w_{\mathbf{u}}\|\mathbf{u}_t\|^2 + w_v(v_x - v_x^\star)^2 + w_h(p_y - p_y^\star)^2,$$

where v_x and $v_x^\star$ are the current and desired horizontal velocities, p_y and $p_y^\star$ are the current and desired heights of the root link, and $w_{\mathbf{u}}$, w_v, and w_h determine the weight on each objective term. The swimmer and walker are shown in Figure 2, along with a schematic of the policy. The specific weights and a description of each robot are presented in the supplemental appendix. The initial trajectory for the swimmer was generated with unconstrained trajectory optimization using DDP, while the initial walking trajectory used a demonstration from a hand-crafted locomotion system (Yin et al., 2007), following our prior work (Levine & Koltun, 2013a). Initial push responses were generated by tracking a walking demonstration with DDP.

The policies consisted of neural networks with one hidden layer, with a soft rectifier $a = \log(1 + \exp(z))$ at the first layer and linear connections to the output layer. Gaussian noise with a learned diagonal covariance was added to the output to create a stochastic policy. When evaluating the cost of a policy, the noise was removed, yielding a deterministic controller. While this class of policies is very expressive, it poses a considerable challenge for policy search methods, due to its nonlinearity and high dimensionality.

As discussed in Section 3, the stochasticity of the policy depends on the cost magnitude. A low cost will produce broad trajectory distributions, which are good for learning, but will also produce a more stochastic policy, which might perform poorly. To speed up learning and still achieve a good final policy, we found it useful to gradually increase the cost by a factor of 10 over the first 50 iterations.

4.1. Simple Locomotion

Comparisons to previous methods on the swimming and walking tasks are presented in Figure 1, using policies with either 5 or 10 hidden units. Our constrained guided policy search method (constrained GPS) succeeded on all of the tasks. The only other method to succeed on all tasks

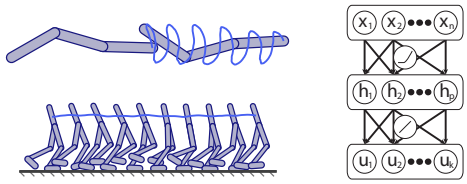

Figure 2. The swimmer and bipedal walker, next to a diagram of the neural network controller. Blue curves indicate root link trajectories of the learned locomotion policy.

was variational GPS, which also alternates policy and trajectory optimization (Levine & Koltun, 2013b). However, as we will show in the next sections, constrained GPS generalized better to more difficult domains, where the risk-seeking objective in variational GPS caused difficulties.

Importance-sampled guided policy search (ISGPS), which adds guiding samples from a trajectory distribution into the policy search via importance sampling (Levine & Koltun, 2013a), solved both swimming tasks but failed to learn a walking gait with 5 hidden units. Guiding samples are only useful if they can be reproduced by the policy. If the policy class is too restricted, for example by partial observability or limited expressiveness, or if the trajectory takes inconsistent actions in similar states, no policy can reproduce the guiding samples and ISGPS reverts to random exploration.

Adapted ISGPS optimizes the trajectory to match the policy, but still uses importance sampling, which can cause problems due to weight degeneracy and local optima. The adapted variant also could not find a 5 hidden unit walking gait. Both variants did converge faster than constrained GPS on the easier swimming tasks, since constrained GPS requires a few iterations to find reasonable Lagrange multipliers and bring the policy and trajectory into agreement.

We also compared to cost-weighted regression, which fits the policy to previous on-policy samples weighted by the exponential of their reward (negative cost) (Peters & Schaal, 2007; Kober & Peters, 2009). This approach is representative of a broad class of reinforcement learning methods, which use model-free random exploration and optimize the policy to increase the probability of low cost samples (Peters & Schaal, 2008; Theodorou et al., 2010). Since a dynamics model was available, we provided this method with 1000 samples at each iteration, but it was still only

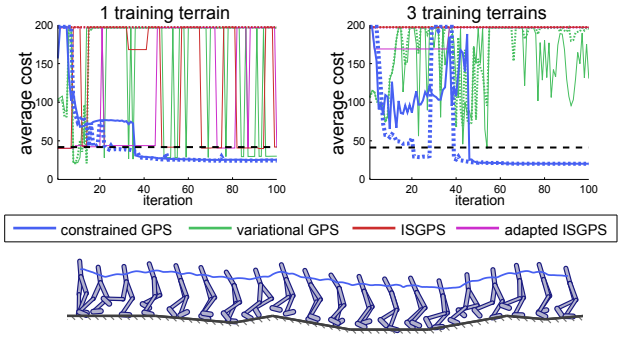

Figure 3. Uneven terrain. Solid lines show performance on the training terrains, dotted lines show generalization to unseen terrains. Constrained GPS was able to generalize from both training conditions. One test terrain with a trace of our policy is shown.

able to learn one swimming task, since the search space for the walker is far too large and complex to handle with model-free methods. While we also tested other RL algorithms, such as REINFORCE-style policy gradient, we found that they performed poorly in all of our experiments.

Lastly, we compared to DAGGER, an imitation learning method that mimics an oracle (Ross et al., 2011), which in our case was the initial trajectory distribution. At each iteration, DAGGER adds on-policy samples to its dataset and reoptimizes the policy to take the oracle action at each sample. Like ISGPS, DAGGER relies on being able to reproduce the oracle's behavior with some policy from the policy class, which may be impossible without adapting the trajectory. Furthermore, poor policies will deviate from the trajectory, and the oracle's linear feedback actions in these faraway states are highly suboptimal, violating DAGGER's assumptions. To mitigate this, we weighted the samples by their probability under the trajectory distribution, but DAGGER was still unable to learn a walking gait.

4.2. Uneven Terrain

In this section, we investigate the ability of each method to learn policies that generalize to new situations, which is a key advantage of parametric policies. Our first experiment is based on the uneven terrain task we proposed in prior work (Levine & Koltun, 2013a), where the walker traverses random uneven terrain with slopes of up to 10°. The walker was trained on one or three terrains, and tested on four other random terrains. The policies used 50 hidden units, for a total of 1206 parameters. We omitted the foot contact and forward kinematics features that we used in prior work (Levine, 2013), and instead trained policies that map the state vector directly to joint torques. This significantly increased the difficulty of the problem, to the point that prior methods could not discover a reliable policy.

The results are shown in Figure 3, with methods that failed on both tasks omitted for clarity. The dotted lines indicate

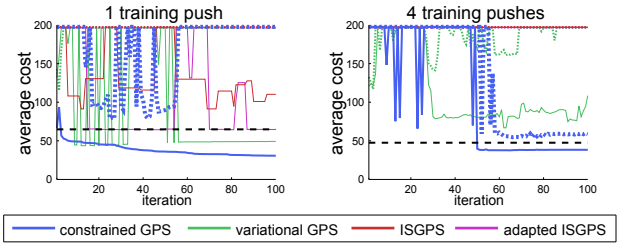

Figure 4. Push response results. Solid lines show training push results, dotted lines show generalization. Constrained GPS learned a successful, generalizable policy from four training pushes.

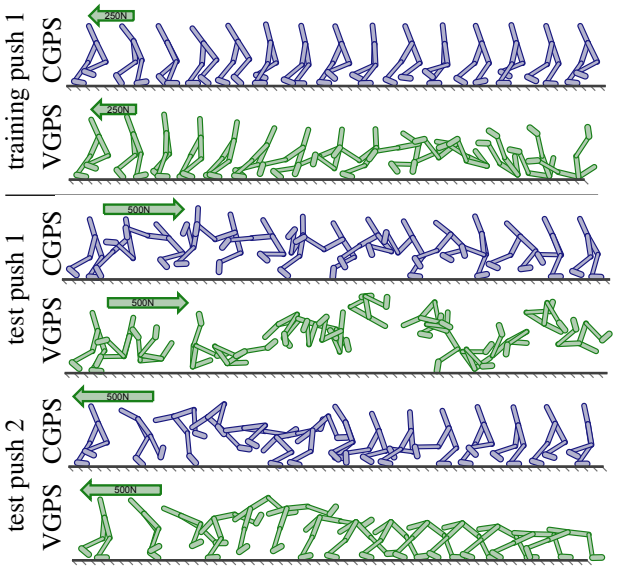

Figure 5. A few push responses with policies learned by constrained and variational GPS on four training pushes. The horizontal spacing between the figures is expanded for clarity.

the performance of each method on the test terrains. Constrained GPS was able to learn successful policies on both training sets, and in both cases the policies generalized well to new terrains. Prior methods struggled with this task, and though they were able to traverse the single terrain by the last iteration, none could traverse all three training terrains, and none could generalize successfully.

Like the locomotion tasks in the previous section, the main challenge in this task was to learn an effective gait. However, one advantage of general-purpose policies like neural networks is that a single policy can in principle learn multiple strategies, and deploy them as needed in response to the state. This issue is explored in the next section.

4.3. Push Recovery

To explore the ability of each method to learn policies that choose different strategies based on the state, we trained walking policies that recover from strong lateral pushes, in the range of 250 to 500 Newtons, delivered for 100ms. Responding to pushes of such magnitude requires not just disturbance rejection, but a well-timed recovery strategy, such

as a protective step or even standing up after a fall. Because the push can occur at different points in the gait, multiple recovery strategies must be learned simultaneously. A strategy that succeeds in one pose may fail in another.

We trained policies with 50 hidden units on one or four pushes, and tested them on four different pushes, with initial states sampled from an example gait cycle. The results are shown in Figure 4, with dotted lines indicating performance on the test pushes. Constrained GPS learned a policy from four training pushes that generalized well to the test set, while no prior method could find a policy that succeeded even on the entire training set. Although several methods learned a successful policy from one training push, none of these policies could generalize, underscoring the importance of learning multiple recovery strategies.

Figure 5 shows a few of the recoveries learned by constrained and variational GPS. Note that even the weaker 250 Newton pushes are quite violent, and sufficient to lift the 24kg walker off the ground completely. Nonetheless, constrained GPS was able to recover succesfully. The supplementary video[3] shows the policy responding to each of the training and test pushes, as well as a few additional pushes delivered in quick succession. The video illustrates some of the strategies learned by the policy, such as kicking against the ground to recover from a backward fall. The ability to learn such generalizable strategies is one of the key benefits of training expressive parametric controllers.

5. Discussion

We presented a constrained guided policy search algorithm for learning complex, high-dimensional policies, using trajectory optimization with a policy agreement constraint. We showed how the constrained problem can be solved efficiently with dual gradient descent and dynamic programming, and presented a comparison of our method to prior policy search algorithms. Our approach was able to learn a push recovery behavior for a bipedal walker that captured generalizable recovery strategies, and outperformed prior methods in direct comparisons.

Constrained GPS builds on our recent work on using trajectory optimization for policy search with importance sampling (Levine & Koltun, 2013a) and variational inference (Levine & Koltun, 2013b). Our evaluation shows that the constrained approach outperforms the prior methods. Importance sampling can be vulnerable to degeneracies in the importance weights. While variational GPS addresses this issue, it does so by optimizing a risk-seeking objective: the expected *exponential* reward. Such an objective rewards policies that succeed only occasionally, which can lead to unreliable results. On the other hand, variational GPS is somewhat easier to implement, and importance sampled GPS is simpler and faster when adaptation of the trajectory is not required, such as on the simpler locomotion tasks.

Variational GPS makes the policy an M-projection onto the trajectory distribution, forcing it to visit all regions where this distribution is high, even if it means visiting costly regions where it's not. We argue that an I-projection objective is more appropriate, as it forces the policy to avoid costly regions where the trajectory distribution is low, even at the cost of missing some high-density areas. As shown in Section 2, this corresponds to the expected cost plus an entropy term. Previous work has also argued for this sort of objective, on the basis of probabilistic robustness (Ziebart, 2010) and computational benefits (Todorov, 2006). The related area of stochastic optimal control has developed model-free reinforcement learning algorithms under a similar objective (Theodorou et al., 2010; Rawlik et al., 2012).

Concurrently with our work, Mordatch and Todorov (2014) proposed another trajectory optimization algorithm for guiding policy search. Further research into trajectory optimization techniques best suited for policy search is a promising direction for future work.

While we assume a known model of the dynamics, prior work has proposed learning the dynamics from data (Deisenroth & Rasmussen, 2011; Ross & Bagnell, 2012; Deisenroth et al., 2013), and using our method with learned models could allow for wider applications in the future.

Our method also has several limitations that could be addressed in future work. Our use of trajectory optimization requires the policy to be Markovian, so any hidden state carried across time steps must itself be incorporated into the policy search as additional state information. Our method is also local in nature. While we can use multiple trajectories to optimize a single policy, highly stochastic policies that visit large swathes of the state space would not be approximated well by our method.

Another avenue to explore in future work is the application of our method to learning rich control policies at scale. Since we can learn a single policy that unifies a variety of trajectories in different environments and with different initial conditions, a sufficiently expressive policy could in principle learn complex sensorimotor associations from a large number of trajectories, all optimized in parallel in their respective environments. This could offer superior generalization and discovery of higher-level motor skills.

Acknowledgments

We thank Emanuel Todorov, Tom Erez, and Yuval Tassa for the MuJoCo simulator, and the anonymous reviewers for their constructive feedback. Sergey Levine was supported by an NVIDIA Graduate Fellowship.

[3] http://graphics.stanford.edu/projects/cgpspaper/index.htm

References

Arnold, W.F., III and Laub, A.J. Generalized eigenproblem algorithms and software for algebraic Riccati equations. *Proceedings of the IEEE*, 72(12), 1984.

Boyd, S. and Vandenberghe, L. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

Deisenroth, M. and Rasmussen, C. PILCO: a model-based and data-efficient approach to policy search. In *International Conference on Machine Learning (ICML)*, 2011.

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

Ijspeert, A., Nakanishi, J., and Schaal, S. Learning attractor landscapes for learning motor primitives. In *Advances in Neural Information Processing Systems (NIPS)*, 2003.

Kappen, H. J., Gómez, V., and Opper, M. Optimal control as a graphical model inference problem. *Machine Learning*, 87(2), 2012.

Kober, J. and Peters, J. Learning motor primitives for robotics. In *International Conference on Robotics and Automation (ICRA)*, 2009.

Kober, J., Bagnell, J. A., and Peters, J. Reinforcement learning in robotics: A survey. *International Journal of Robotic Research*, 32(11), 2013.

Kolter, J. Z., Jackowski, Z., and Tedrake, R. Design, analysis and learning control of a fully actuated micro wind turbine. In *American Control Conference (ACC)*, 2012.

Levine, S. Exploring deep and recurrent architectures for optimal control. In *NIPS 2013 Workshop on Deep Learning*, 2013.

Levine, S. and Koltun, V. Guided policy search. In *International Conference on Machine Learning (ICML)*, 2013a.

Levine, S. and Koltun, V. Variational policy search via trajectory optimization. In *Advances in Neural Information Processing Systems (NIPS)*, 2013b.

Mordatch, I. and Todorov, E. Combining the benefits of function approximation and trajectory optimization. In *Robotics: Science and Systems (RSS)*, 2014.

Paraschos, A., Daniel, C., Peters, J., and Neumann, G. Probabilistic movement primitives. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

Peters, J. and Schaal, S. Applying the episodic natural actor-critic architecture to motor primitive learning. In *European Symposium on Artificial Neural Networks (ESANN)*, 2007.

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

Rawlik, K., Toussaint, M., and Vijayakumar, S. On stochastic optimal control and reinforcement learning by approximate inference. In *Robotics: Science and Systems (RSS)*, 2012.

Ross, S. and Bagnell, A. Agnostic system identification for model-based reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2012.

Ross, S., Gordon, G., and Bagnell, A. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

Ross, S., Melik-Barkhudarov, N., Shankar, K. Shaurya, Wendel, A., Dey, D., Bagnell, J. A., and Hebert, M. Learning monocular reactive UAV control in cluttered natural environments. In *International Conference on Robotics and Automation (ICRA)*, 2013.

Theodorou, E., Buchli, J., and Schaal, S. Reinforcement learning of motor skills in high dimensions: a path integral approach. In *International Conference on Robotics and Automation (ICRA)*, 2010.

Todorov, E. Linearly-solvable Markov decision problems. In *Advances in Neural Information Processing Systems (NIPS)*, 2006.

Todorov, E. and Li, W. A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *American Control Conference (ACC)*, 2005.

Todorov, E., Erez, T., and Tassa, Y. MuJoCo: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems (IROS)*, 2012.

Yin, K., Loken, K., and van de Panne, M. SIMBICON: simple biped locomotion control. *ACM Transactions Graphics*, 26(3), 2007.

Ziebart, B. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. PhD thesis, Carnegie Mellon University, 2010.

A. Simulation and Cost Details

The swimmer consisted of 3 links, with 10 state dimensions corresponding to joint angles, joint angular velocities, and the position, angle, velocity, and angular velocity of the head, with two action dimensions corresponding to the torques between the joints. The simulation applied drag on each link of the swimmer to roughly simulate a fluid, allowing it to propel itself. The simulation step was set to 0.05s, and the reward weights were $w_{\mathbf{u}} = 0.0001$, $w_v = 1$, and $w_h = 0$, with the desired velocity was $v_x^\star = 1$m/s.

The bipedal walker consisted of seven links: a torso and three links for each leg, for a total of 18 dimensions, including joint angles, the global position and orientation of the torso, and the corresponding velocities. The action space had six dimensions, corresponding to each of the joints. MuJoCo was used to simulate soft, differentiable contacts to allow gradient-based optimization to proceed even in the presence of contact forces. The simulation step was set to 0.01s, and the reward weights for all walker tasks were $w_{\mathbf{u}} = 0.0001$, $w_v = 1$, and $w_h = 10$, with desired velocity and height $v_x^\star = 2.1$m/s and $p_y^\star = 1.1$m.

B. Sample-Based Gradients

As discussed in Section 3.1, the Laplace approximation may not accurately capture the structure of $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ in the entire region where $q(\mathbf{x}_t)$ is large, and since the policy is trained by sampling states from $q(\mathbf{x}_t)$, policy optimization optimizes a different objective. This can lead to nonconvergence when the policy is highly nonlinear. To reconcile this problem, we can approximate the policy terms in the objective with M random samples $\mathbf{x}_{ti}$, drawn from $q(\mathbf{x}_t)$, rather than by using a linearization of the policy:

$$\begin{aligned}\mathcal{L}(q) \approx \sum_{t=1}^{T} \ell(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \frac{1}{2}\text{tr}\left(\Sigma_t \ell_{\mathbf{xu},\mathbf{xu}t}\right) - \frac{1}{2}\log|\mathbf{A}_t| + \\ \frac{\lambda_t}{2M}\sum_{i=1}^{M}(\mathbf{u}_{ti} - \mu_t^\pi(\mathbf{x}_{ti}))^{\mathrm{T}}\mathbf{A}_t^{-1}(\mathbf{u}_{ti} - \mu_t^\pi(\mathbf{x}_{ti})) + \\ \frac{\lambda_t}{2}\text{tr}\left(\mathbf{A}_t^{-1}\Sigma_t^\pi\right) + \frac{\lambda_t}{2}\log|\mathbf{A}_t|,\end{aligned}$$

where the actions are given by $\mathbf{u}_{ti} = \mathbf{K}_t\mathbf{x}_{ti} + \hat{\mathbf{u}}_t$. Note that the samples $\mathbf{x}_{ti}$ depend on $\hat{\mathbf{x}}_t$, according to $\mathbf{x}_{ti} = \hat{\mathbf{x}}_t + \mathbf{L}_t^{\mathrm{T}}\mathbf{s}_{ti}$, where $\mathbf{s}_{ti}$ is a sample from a zero-mean spherical Gaussian, and $\mathbf{L}_t$ is the upper triangular Cholesky decomposition of $\mathbf{S}_t$.[4] As before, we differentiate with respect to $\hat{\mathbf{u}}_t$, substituting $Q_{\mathbf{u},\mathbf{u}t}$ and $Q_{\mathbf{u}t}$ as needed:

$$\begin{aligned}\mathcal{L}_{\mathbf{u}t} &= Q_{\mathbf{u}t} + \lambda_t\mathbf{A}_t^{-1}\hat{\mu}_t^\pi \\ \mathcal{L}_{\mathbf{u},\mathbf{u}t} &= Q_{\mathbf{u},\mathbf{u}t} + \lambda_t\mathbf{A}_t^{-1},\end{aligned}$$

[4] Keeping the same samples $\mathbf{s}_{ti}$ across iterations reduces variance and can greatly improve convergence.

where $\hat{\mu}_t^\pi = \frac{1}{M}\sum_{i=1}^{M}(\mathbf{u}_{ti} - \mu_t^\pi(\mathbf{x}_{ti}))$ is the average difference between the linear feedback and the policy. This yields the following correction and feedback terms:

$$\begin{aligned}\mathbf{k}_t &= -\left(Q_{\mathbf{u},\mathbf{u}t} + \lambda_t\mathbf{A}_t^{-1}\right)^{-1}\left(Q_{\mathbf{u}t} + \lambda_t\mathbf{A}_t^{-1}\hat{\mu}_t^\pi\right) \\ \mathbf{K}_t &= -\left(Q_{\mathbf{u},\mathbf{u}t} + \lambda_t\mathbf{A}_t^{-1}\right)^{-1}\left(Q_{\mathbf{u},\mathbf{x}t} - \lambda_t\mathbf{A}_t^{-1}\hat{\mu}_{\mathbf{x}t}^\pi\right),\end{aligned}$$

where $\hat{\mu}_{\mathbf{x}t}^\pi = \frac{1}{M}\sum_{i=1}^{M}\mu_{\mathbf{x}t}^\pi(\mathbf{x}_{ti})$ is the average policy gradient. So far, the change to the mean is identical to simply averaging the policy values and policy gradients over all the samples. In fact, a reasonable approximation can be obtained by doing just that, and substituting the sample averages directly into the equations in Section 3.1. This is the approximation we use in our experiments, as it is slightly faster and does not appear to significantly degrade the results. However, the true gradients with respect to $\mathbf{A}_t$ are different. Below, we differentiate the objection with respect to $\mathbf{A}_t$ and $\hat{\mathbf{K}}_t$ as before, where we use $\hat{\mathbf{K}}_t$ to distinguish the covariance term from the feedback in the first dynamic programming pass, which is no longer identical. The derivatives with respect to $\mathbf{A}_t$ and $\hat{\mathbf{K}}_t$ are

$$\begin{aligned}\mathcal{L}_{\mathbf{A}t} &= \frac{1}{2}Q_{\mathbf{u},\mathbf{u}t} + \frac{\lambda_t - 1}{2}\mathbf{A}_t^{-1} - \frac{\lambda_t}{2}\mathbf{A}_t^{-1}\mathbf{M}\mathbf{A}_t^{-1} \\ \mathcal{L}_{\hat{\mathbf{K}}t} &= Q_{\mathbf{u},\mathbf{u}t}\hat{\mathbf{K}}_t\mathbf{S}_t + Q_{\mathbf{u},\mathbf{x}t}\mathbf{S}_t + \lambda_t\mathbf{A}_t^{-1}(\hat{\mathbf{K}}_t\hat{\mathbf{S}}_t - \mathbf{C}_t),\end{aligned}$$

where $\mathbf{M} = \Sigma_t^\pi + \sum_{i=1}^{M}(\mathbf{u}_{ti} - \mu_t^\pi(\mathbf{x}_{ti}))(\mathbf{u}_{ti} - \mu_t^\pi(\mathbf{x}_{ti}))$, $\hat{\mathbf{S}}_t = \frac{1}{M}\sum_{i=1}^{M}\mathbf{x}_{ti}\mathbf{x}_{ti}^{\mathrm{T}}$, and $\mathbf{C}_t = \frac{1}{M}\sum_{i=1}^{M}\mu_t^\pi(\mathbf{x}_{ti})\mathbf{x}_{ti}^{\mathrm{T}}$, where we simplified using the assumption that $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{u}}_t$ are zero. We again solve for $\mathbf{A}_t$ by solving the CARE in Equation 8. To solve for $\hat{\mathbf{K}}_t$, we rearrange the terms to get

$$\hat{\mathbf{K}}_t\hat{\mathbf{S}}_t\mathbf{S}_t^{-1} + \frac{1}{\lambda_t}\mathbf{A}_t Q_{\mathbf{u},\mathbf{u}t}\hat{\mathbf{K}}_t = \mathbf{C}_t\mathbf{S}_t^{-1} - \frac{1}{\lambda_t}\mathbf{A}_t Q_{\mathbf{u},\mathbf{x}t}.$$

This equation is linear in $\hat{\mathbf{K}}_t$, but requires solving a sparse linear system with dimensionality equal to the number of entries in $\hat{\mathbf{K}}_t$, which increases the computational cost.

Differentiating with respect to $\mathbf{S}_t$, we get:

$$\begin{aligned}\mathcal{L}_{\mathbf{S}t} = \frac{1}{2}\big[&Q_{\mathbf{x},\mathbf{x}t} + \mathbf{K}_t^{\mathrm{T}}Q_{\mathbf{u},\mathbf{x}t} + Q_{\mathbf{x},\mathbf{u}t}\mathbf{K}_t + \mathbf{K}_t^{\mathrm{T}}Q_{\mathbf{u},\mathbf{u}t}\mathbf{K}_t \\ &+\text{choldiff}(\mathbf{D}_t) + \text{choldiff}(\mathbf{D}_t)^{\mathrm{T}}\big]\end{aligned}$$

where $\mathbf{D}_t = \frac{\lambda_t}{M}\sum_{i=1}^{M}(\hat{\mathbf{K}}_t - \mu_{\mathbf{x}t}^\pi(\mathbf{x}_{ti}))^{\mathrm{T}}\mathbf{A}_t^{-1}(\mathbf{u}_{ti} - \mu_t^\pi(\mathbf{x}_{ti}))\mathbf{s}_{ti}^{\mathrm{T}}$ and $\text{choldiff}(\dots)$ indicates the differentiation of the Cholesky decomposition, for example using the method described by Giles in "An extended collection of matrix derivative results for forward and reverse mode algorithmic differentiation" (2008). While this will provide us with the correct gradient, $\text{choldiff}(\mathbf{D}_t) + \text{choldiff}(\mathbf{D}_t)^{\mathrm{T}}$ is not guaranteed to be positive definite. In this case, we found it useful to regularize by interpolating the gradient with the one obtained from the Laplace approximation.