Guided Policy Search

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

Abstract

Direct policy search can effectively scale to high-dimensional systems, but complex policies with hundreds of parameters often present a challenge for such methods, requiring numerous samples and often falling into poor local optima. We present a guided policy search algorithm that uses trajectory optimization to direct policy learning and avoid poor local optima. We show how differential dynamic programming can be used to generate suitable guiding samples, and describe a regularized importance sampled policy optimization that incorporates these samples into the policy search. We evaluate the method by learning neural network controllers for planar swimming, hopping, and walking, as well as simulated 3D humanoid running.

1. Introduction

Reinforcement learning is a powerful framework for controlling dynamical systems. Direct policy search methods are often employed in high-dimensional applications such as robotics, since they scale gracefully with dimensionality and offer appealing convergence guarantees (Peters & Schaal, 2008). However, it is often necessary to carefully choose a specialized policy class to learn the policy in a reasonable number of iterations without falling into poor local optima. Substantial improvements on real-world systems have come from specialized and innovative policy classes (Ijspeert et al., 2002). This specialization comes at a cost in generality, and can restrict the types of behaviors that can be learned. For example, a policy that tracks a single trajectory cannot choose different trajectories depending on the state. In this work, we aim to learn policies with very general and flexible representations, such as large neural networks, which can represent a broad range of behaviors. Learning such complex, nonlinear policies with standard policy gradient methods can require a huge number of iterations, and can be disastrously prone to poor local optima.

In this paper, we show how trajectory optimization can guide the policy search away from poor local optima. Our guided policy search algorithm uses differential dynamic programming (DDP) to generate "guiding samples," which assist the policy search by exploring high-reward regions. An importance sampled variant of the likelihood ratio estimator is used to incorporate these guiding samples directly into the policy search.

We show that DDP can be modified to sample from a distribution over high reward trajectories, making it particularly suitable for guiding policy search. Furthermore, by initializing DDP with example demonstrations, our method can perform learning from demonstration. The use of importance sampled policy search also allows us to optimize the policy with second order quasi-Newton methods for many gradient steps without requiring new on-policy samples, which can be crucial for complex, nonlinear policies.

Our main contribution is a guided policy search algorithm that uses trajectory optimization to assist policy learning. We show how to obtain suitable guiding samples, and we present a regularized importance sampled policy optimization method that can utilize guiding samples and does not require a learning rate or new samples at every gradient step. We evaluate our method on planar swimming, hopping, and walking, as well as 3D humanoid running, using general-purpose neural network policies. We also show that both the proposed sampling scheme and regularizer are essential for good performance, and that the learned policies can generalize successfully to new environments.

2. Preliminaries

Reinforcement learning aims to find a policy π to control an agent in a stochastic environment. At each time

Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

step t, the agent observes a state $\mathbf{x}_t$ and chooses an action according to $\pi(\mathbf{u}_t|\mathbf{x}_t)$, producing a state transition according to $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$. The goal is specified by the reward $r(\mathbf{x}_t, \mathbf{u}_t)$, and an optimal policy is one that maximizes the expected sum of rewards (return) from step 1 to T. We use ζ to denote a sequence of states and actions, with $r(\zeta)$ and $\pi(\zeta)$ being the total reward along ζ and its probability under π. We focus on finite-horizon tasks in continuous domains, though extensions to other formulations are also possible.

Policy gradient methods learn a parameterized policy π_θ by directly optimizing its expected return $E[J(\theta)]$ with respect to the parameters θ. In particular, likelihood ratio methods estimate the gradient $E[\nabla J(\theta)]$ using samples $\zeta_1, \ldots, \zeta_m$ drawn from the current policy π_θ, and then improve the policy by taking a step along this gradient. The gradient can be estimated using the following equation (Peters & Schaal, 2008):

$$E[\nabla J(\theta)] = E[r(\zeta)\nabla \log \pi_\theta(\zeta)] \approx \frac{1}{m}\sum_{i=1}^{m} r(\zeta_i)\nabla \log \pi_\theta(\zeta_i),$$

where $\nabla \log \pi_\theta(\zeta_i)$ decomposes to $\sum_t \nabla \log \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$, since the transition model $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ does not depend on θ. Standard likelihood ratio methods require new samples from the current policy at each gradient step, do not admit off-policy samples, and require the learning rate to be chosen carefully to ensure convergence. In the next section, we discuss how importance sampling can be used to lift these constraints.

3. Importance Sampled Policy Search

Importance sampling is a technique for estimating an expectation $E_p[f(x)]$ with respect to $p(x)$ using samples drawn from a different distribution $q(x)$:

$$E_p[f(x)] = E_q\left[\frac{p(x)}{q(x)}f(x)\right] \approx \frac{1}{Z}\sum_{i=1}^{m}\frac{p(x_i)}{q(x_i)}f(x_i).$$

Importance sampling is unbiased if $Z = m$, though we use $Z = \sum_i \frac{p(x_i)}{q(x_i)}$ throughout as it provides lower variance. Prior work proposed estimating $E[J(\theta)]$ with importance sampling (Peshkin & Shelton, 2002; Tang & Abbeel, 2010). This allows using off-policy samples and results in the following estimator:

$$E[J(\theta)] \approx \frac{1}{Z(\theta)}\sum_{i=1}^{m}\frac{\pi_\theta(\zeta_i)}{q(\zeta_i)}r(\zeta_i). \quad (1)$$

The variance of this estimator can be reduced further by observing that past rewards do not depend on future actions (Sutton et al., 1999; Baxter et al., 2001):

$$E[J(\theta)] \approx \sum_{t=1}^{T}\frac{1}{Z_t(\theta)}\sum_{i=1}^{m}\frac{\pi_\theta(\zeta_{i,1:t})}{q(\zeta_{i,1:t})}r\left(\mathbf{x}_t^i, \mathbf{u}_t^i\right), \quad (2)$$

where $\pi_\theta(\zeta_{i,1:t})$ denotes the probabiliy of the first t steps of ζ_i, and $Z_t(\theta)$ normalizes the weights. To include samples from multiple distributions, we follow prior work and use a fused distribution of the form $q(\zeta) = \frac{1}{n}\sum_j q_j(\zeta)$, where each q_j is either a previous policy or a guiding distribution constructed with DDP.

Prior methods optimized Equation (1) or (2) directly. Unfortunately, with complex policies and long rollouts, this often produces poor results, as the estimator only considers the relative probability of each sample and does not require any of these probabilities to be high. The optimum can assign a low probability to all samples, with the best sample slightly more likely than the rest, thus receiving the only nonzero weight. Tang and Abbeel (2010) attempted to mitigate this issue by constraining the optimization by the variance of the weights. However, this is ineffective when the distributions are highly peaked, as is the case with long rollouts, because very few samples have nonzero weights, and their variance tends to zero as the sample count increases. In our experiments, we found this problem to be very common in large, high-dimensional problems.

To address this issue, we augment Equation (2) with a novel regularizing term. A variety of regularizers are possible, but we found the most effective one to be the logarithm of the normalizing constant:

$$\Phi(\theta) = \sum_{t=1}^{T}\left[\frac{1}{Z_t(\theta)}\sum_{i=1}^{m}\frac{\pi_\theta(\zeta_{i,1:t})}{q(\zeta_{i,1:t})}r(\mathbf{x}_t^i, \mathbf{u}_t^i) + w_r \log Z_t(\theta)\right].$$

This objective is maximized using an analytic gradient derived in Appendix A of the supplement. It is easy to check that this estimator is consistent, since $Z_t(\theta) \to 1$ in the limit of infinite samples. The regularizer acts as a soft maximum over the logarithms of the weights, ensuring that at least some samples have a high probability under π_θ. Furthermore, by adaptively adjusting w_r, we can control how far the policy is allowed to deviate from the samples, which can be used to limit the optimization to regions that are better represented by the samples if it repeatedly fails to make progress.

4. Guiding Samples

Prior methods employed importance sampling to reuse samples from previous policies (Peshkin & Shelton, 2002; Kober & Peters, 2009; Tang & Abbeel, 2010). However, when learning policies with hundreds of parameters, local optima make it very difficult to find a good solution. In this section, we show how differential dynamic programming (DDP) can be used to supplement the sample set with off-policy guiding samples that guide the policy search to regions of high reward.

4.1. Constructing Guiding Distributions

An effective guiding distribution covers high-reward regions while avoiding large $q(\zeta)$ densities, which result in low importance weights. We posit that a good guiding distribution is an I-projection of $\rho(\zeta) \propto \exp(r(\zeta))$. An I-projection q of ρ minimizes the KL-divergence $D_{\mathrm{KL}}(q||\rho) = E_q[-r(\zeta)] - \mathcal{H}(q)$, where $\mathcal{H}$ denotes entropy. The first term forces q to be high only in regions of high reward, while the entropy maximization favors broad distributions. We will show that an approximate Gaussian I-projection of ρ can be computed using a variant of DDP called iterative LQR (Tassa et al., 2012), which optimizes a trajectory by repeatedly solving for the optimal policy under linear-quadratic assumptions. Given a trajectory $(\bar{\mathbf{x}}_1, \bar{\mathbf{u}}_1), \ldots, (\bar{\mathbf{x}}_T, \bar{\mathbf{u}}_T)$ and defining $\hat{\mathbf{x}}_t = \mathbf{x}_t - \bar{\mathbf{x}}_t$ and $\hat{\mathbf{u}}_t = \mathbf{u}_t - \bar{\mathbf{u}}_t$, the dynamics and reward are approximated as

$$\begin{aligned}\hat{\mathbf{x}}_{t+1} &\approx f_{\mathbf{x}t}\hat{\mathbf{x}}_t + f_{\mathbf{u}t}\hat{\mathbf{u}}_t \\ r(\mathbf{x}_t, \mathbf{u}_t) &\approx \hat{\mathbf{x}}_t^{\mathrm{T}} r_{\mathbf{x}t} + \hat{\mathbf{u}}^{\mathrm{T}} r_{\mathbf{u}t} + \frac{1}{2}\hat{\mathbf{x}}_t^{\mathrm{T}} r_{\mathbf{x}\mathbf{x}t}\hat{\mathbf{x}}_t + \frac{1}{2}\hat{\mathbf{u}}_t^{\mathrm{T}} r_{\mathbf{u}\mathbf{u}t}\hat{\mathbf{u}}_t \\ &\quad + \hat{\mathbf{u}}_t^{\mathrm{T}} r_{\mathbf{u}\mathbf{x}t}\hat{\mathbf{x}}_t + r(\bar{\mathbf{x}}_t, \bar{\mathbf{u}}_t).\end{aligned}$$

Subscripts denote the Jacobians and Hessians of the dynamics f and reward r, which are assumed to exist.[1] Iterative LQR recursively estimates the Q-function:

$$\begin{aligned} Q_{\mathbf{x}\mathbf{x}t} &= r_{\mathbf{x}\mathbf{x}t} + f_{\mathbf{x}t}^{\mathrm{T}} V_{\mathbf{x}\mathbf{x}t+1} f_{\mathbf{x}t} & Q_{\mathbf{x}t} &= r_{\mathbf{x}t} + f_{\mathbf{x}t}^{\mathrm{T}} V_{\mathbf{x}t+1} \\ Q_{\mathbf{u}\mathbf{u}t} &= r_{\mathbf{u}\mathbf{u}t} + f_{\mathbf{u}t}^{\mathrm{T}} V_{\mathbf{x}\mathbf{x}t+1} f_{\mathbf{u}t} & Q_{\mathbf{u}t} &= r_{\mathbf{u}t} + f_{\mathbf{u}t}^{\mathrm{T}} V_{\mathbf{x}t+1} \\ Q_{\mathbf{u}\mathbf{x}t} &= r_{\mathbf{u}\mathbf{x}t} + f_{\mathbf{u}t}^{\mathrm{T}} V_{\mathbf{x}\mathbf{x}t+1} f_{\mathbf{x}t}, \end{aligned}$$

as well as the value function and linear policy terms:

$$\begin{aligned} V_{\mathbf{x}t} &= Q_{\mathbf{x}t} - Q_{\mathbf{u}\mathbf{x}t}^{\mathrm{T}} Q_{\mathbf{u}\mathbf{u}t}^{-1} Q_{\mathbf{u}} & \mathbf{k}_t &= -Q_{\mathbf{u}\mathbf{u}t}^{-1} Q_{\mathbf{u}t} \\ V_{\mathbf{x}\mathbf{x}t} &= Q_{\mathbf{x}\mathbf{x}t} - Q_{\mathbf{u}\mathbf{x}t}^{\mathrm{T}} Q_{\mathbf{u}\mathbf{u}t}^{-1} Q_{\mathbf{u}\mathbf{x}} & \mathbf{K}_t &= -Q_{\mathbf{u}\mathbf{u}t}^{-1} Q_{\mathbf{u}\mathbf{x}t}. \end{aligned}$$

The deterministic optimal policy is then given by

$$g(\mathbf{x}_t) = \bar{\mathbf{u}}_t + \mathbf{k}_t + \mathbf{K}_t(\mathbf{x}_t - \bar{\mathbf{x}}_t). \tag{3}$$

By repeatedly computing this policy and following it to obtain a new trajectory, this algorithm converges to a locally optimal solution. We now show how the same algorithm can be used with the framework of linearly solvable MDPs (Dvijotham & Todorov, 2010) and the related concept of maximum entropy control (Ziebart, 2010) to build an approximate Gaussian I-projection of $\rho(\zeta) \propto \exp(r(\zeta))$. Under this framework, the optimal policy $\pi_{\mathcal{G}}$ maximizes an augmented reward, given by

$$\tilde{r}(\mathbf{x}_t, \mathbf{u}_t) = r(\mathbf{x}_t, \mathbf{u}_t) - D_{\mathrm{KL}}(\pi_{\mathcal{G}}(\cdot|\mathbf{x}_t)||p(\cdot|\mathbf{x}_t)),$$

where p is a "passive dynamics" distribution. If p is uniform, the expected return of a policy $\pi_{\mathcal{G}}$ is

$$E_{\pi_{\mathcal{G}}}[\tilde{r}(\zeta)] = E_{\pi_{\mathcal{G}}}[r(\zeta)] + \mathcal{H}(\pi_{\mathcal{G}}),$$

which means that if $\pi_{\mathcal{G}}$ maximizes this return, it is an I-projection of ρ. Ziebart (2010) showed that the optimal policy under uniform passive dynamics is

$$\pi_{\mathcal{G}}(\mathbf{u}_t|\mathbf{x}_t) = \exp(Q_t(\mathbf{x}_t, \mathbf{u}_t) - V_t(\mathbf{x}_t)), \tag{4}$$

where V is a modified value function given by

$$V_t(\mathbf{x}_t) = \log \int \exp\left(Q_t(\mathbf{x}_t, \mathbf{u}_t)\right) d\mathbf{u}_t.$$

Under linear dynamics and quadratic rewards, it can be shown that V has the same form as the derivation above, and Equation 4 is a linear Gaussian with the mean given by $g(\mathbf{x}_t)$ and the covariance given by $-Q_{\mathbf{u}\mathbf{u}t}^{-1}$. This stochastic policy corresponds approximately to a Gaussian distribution over trajectories. We can therefore sample from an approximate Gaussian I-projection of ρ by following the stochastic policy

$$\pi_{\mathcal{G}}(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{G}(\mathbf{u}_t; g(\mathbf{x}_t), -Q_{\mathbf{u}\mathbf{u}t}^{-1}).$$

It should be noted that $\pi_{\mathcal{G}}(\zeta)$ is only Gaussian under linear dynamics. When the dynamics are nonlinear, $\pi_{\mathcal{G}}(\zeta)$ approximates a Gaussian around the nominal trajectory. Fortunately, the feedback term usually keeps the samples close to this trajectory, making them suitable guiding samples for the policy search.

4.2. Adaptive Guiding Distributions

The distribution in the preceding section captures high-reward regions, but does not consider the current policy π_θ. We can adapt it to π_θ by sampling from an I-projection of $\rho_\theta(\zeta) \propto \exp(r(\zeta))\pi_\theta(\zeta)$, which is the optimal distribution for estimating $E_{\pi_\theta}[\exp(r(\zeta))]$.[2] To construct the approximate I-projection of ρ_θ, we simply run the DDP algorithm with the reward $\bar{r}(\mathbf{x}_t, \mathbf{u}_t) = r(\mathbf{x}_t, \mathbf{u}_t) + \log \pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$. The resulting distribution is then an approximate I-projection of $\rho_\theta(\zeta) \propto \exp(\bar{r}(\zeta))$.

In practice, we found that many domains do not require adaptive samples. As discussed in the evaluation, adaptation becomes necessary when the initial samples cannot be reproduced by any policy, such as when they act differently in similar states. In that case, adapted samples will avoid different actions in similar states, making them more suited for guiding the policy.

[1] In our implementation, the dynamics are differentiated with finite differences and the reward is differentiated analytically.

[2] While we could also consider the optimal sampler for $E_{\pi_\theta}[r(\zeta)]$, such a sampler would also need to also cover regions with very low reward, which are often very large and would not be represented well by a Gaussian I-projection.

4.3. Incorporating Guiding Samples

We incorporate guiding samples into the policy search by building one or more initial DDP solutions and supplying the resulting samples to the importance sampled policy search algorithm. These solutions can be initialized with human demonstrations or with an offline planning algorithm. When learning from demonstrations, we can perform just one step of DDP starting from the example demonstration, thus constructing a Gaussian distribution around the example. If adaptive guiding distributions are used, they are constructed at each iteration of the policy search starting from the previous DDP solution.

Although our policy search component is model-free, DDP requires a model of the system dynamics. Numerous recent methods have proposed to learn the model (Abbeel et al., 2006; Deisenroth & Rasmussen, 2011; Ross & Bagnell, 2012), and if we use initial examples, only local models are required. In Section 8, we also discuss model-free alternatives to DDP.

One might also wonder why the DDP policy $\pi_{\mathcal{G}}$ is not itself a suitable controller. The issue is that $\pi_{\mathcal{G}}$ is time-varying and only valid around a single trajectory, while the final policy can be learned from many DDP solutions in many situations. Guided policy search can be viewed as transforming a collection of trajectories into a controller. This controller can adhere to any parameterization, reflecting constraints on computation or available sensors in partially observed domains. In our evaluation, we show that such a policy generalizes to situations where the DDP policy fails.

Algorithm 1 Guided Policy Search

1: Generate DDP solutions $\pi_{\mathcal{G}_1}, \ldots, \pi_{\mathcal{G}_n}$
2: Sample $\zeta_1, \ldots, \zeta_m$ from $q(\zeta) = \frac{1}{n}\sum_i \pi_{\mathcal{G}_i}(\zeta)$
3: Initialize $\theta^\star \leftarrow \arg\max_\theta \sum_i \log \pi_{\theta^\star}(\zeta_i)$
4: Build initial sample set $\mathcal{S}$ from $\pi_{\mathcal{G}_1}, \ldots, \pi_{\mathcal{G}_n}, \pi_{\theta^\star}$
5: **for** iteration $k = 1$ to K **do**
6: Choose current sample set $\mathcal{S}_k \subset \mathcal{S}$
7: Optimize $\theta_k \leftarrow \arg\max_\theta \Phi_{\mathcal{S}_k}(\theta)$
8: Append samples from π_{θ_k} to $\mathcal{S}_k$ and $\mathcal{S}$
9: Optionally generate adaptive guiding samples
10: Estimate the values of π_{θ_k} and $\pi_{\theta^\star}$ using $\mathcal{S}_k$
11: **if** π_{θ_k} is better than $\pi_{\theta^\star}$ **then**
12: Set $\theta^\star \leftarrow \theta_k$
13: Decrease w_r
14: **else**
15: Increase w_r
16: Optionally, resample from $\pi_{\theta^\star}$
17: **end if**
18: **end for**
19: Return the best policy $\pi_{\theta^\star}$

5. Guided Policy Search

Algorithm 1 summarizes our method. On line 1, we build one or more DDP solutions, which can be initialized from demonstrations. Initial guiding samples are sampled from these solutions and used on line 3 to pretrain the initial policy $\pi_{\theta^\star}$. Since the samples are drawn from stochastic feedback policies, $\pi_{\theta^\star}$ can already learn useful feedback rules during this pretraining stage. The sample set $\mathcal{S}$ is constructed on line 4 from the guiding samples and samples from $\pi_{\theta^\star}$, and the policy search then alternates between optimizing $\Phi(\theta)$ and gathering new samples from the current policy π_{θ_k}. If the sample set $\mathcal{S}$ becomes too big, a subset $\mathcal{S}_k$ is chosen on line 6. In practice, we simply choose the samples with high importance weights under the current best policy $\pi_{\theta^\star}$, as well as the guiding samples.

The objective $\Phi(\theta)$ is optimized on line 7 with LBFGS. This objective can itself be susceptible to local optima: when the weight on a sample is very low, it is effectively ignored by the optimization. If the guiding samples have low weights, the optimization cannot benefit from them. To mitigate this issue, we repeat the optimization twice, once starting from the best current parameters $\theta^\star$, and once by initializing the parameters with an optimization that maximizes the log weight on the highest-reward sample. Prior work suggested restarting the optimization from each previous policy (Tang & Abbeel, 2010), but this is very slow with complex policies, and still fails to explore the guiding samples, for which there are no known policy parameters.

Once the new policy π_{θ_k} is optimized, we add samples from π_{θ_k} to $\mathcal{S}$ on line 8. If we are using adaptive guiding samples, the adaptation is done on line 9 and new guiding samples are also added. We then use Equation 2 to estimate the returns of both the new policy and the current best policy $\pi_{\theta^\star}$ on line 10. Since $\mathcal{S}_k$ now contains samples from both policies, we expect the estimator to be accurate. If the new policy is better, it replaces the best policy. Otherwise, the regularization weight w_r is increased. Higher regularization causes the next optimization to stay closer to the samples, making the estimated return more accurate. Once the policy search starts making progress, the weight is decreased. In practice, we clamp w_r between 10^{-2} and 10^{-6} and adjust it by a factor of 10. We also found that the policy sometimes failed to improve if the samples from $\pi_{\theta^\star}$ had been unusually good by chance. To address this, we draw additional samples from $\pi_{\theta^\star}$ on line 16 to prevent the policy search from getting stuck due to an overestimate of the best policy's value.

6. Experimental Evaluation

We evaluated our method on planar swimming, hopping, and walking, as well as 3D running. Each task was simulated with the MuJoCo simulator (Todorov et al., 2012), using systems of rigid links with noisy motors at the joints.[3] The policies were represented by neural network controllers that mapped current joint angles and velocities directly to joint torques. The reward function was a sum of three terms:

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

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 weights for each task are given in Appendix B of the supplement, along with descriptions of each simulated robot.

The policy was represented by a neural network with one hidden layer, with a soft rectifying nonlinearity $a = \log(1 + \exp(z))$ at the first layer and linear connections to the output layer. Gaussian noise was added to the output to create a stochastic policy. The policy search ran for 80 iterations, with 40 initial guiding samples, 10 samples added from the current policy at each iteration, and up to 100 samples selected for the active set $\mathcal{S}_k$. Adaptive guiding distributions were only used in Section 6.2. Adaptation was performed in the first 40 iterations, with 50 DDP iterations each time.

The initial guiding distributions for the swimmer and hopper were generated directly with DDP. To illustrate the capacity of our method to learn from demonstration, the initial example trajectory for the walker was obtained from a prior locomotion system (Yin et al., 2007), while the 3D humanoid was initialized with example motion capture of human running.

When using examples, we regularized the reward with a tracking term equal to the squared deviation from the example states and actions, with a weight of 0.05. This term serves to keep the guiding samples close to the examples and ensures that the policy covariance is positive definite. The same term was also used with the swimmer and hopper for consistency. The tracking term was only used for the initial guiding distributions, and was not used during the policy search.

The swimmer, hopper, and walker are shown in Figure 1, with plots of the root position under the learned policies and under the initial DDP solution. The 3D humanoid is shown in Figure 5.

[3]The standard deviation of the noise at each joint was set to 10% of the standard deviation of its torques in the initial DDP-generated gait.

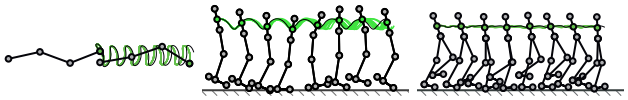


Figure 1. Plots of the root trajectory for the swimmer, hopper, and walker. The green lines are samples from learned policies, and the black line is the DDP solution.

6.1. Comparisons

In the first set of tests, we compared to three variants of our algorithm and three prior methods, using the planar swimmer, hopper, and walker domains. The time horizon was 500, and the policies had 50 hidden units and up to 1256 parameters. The first variant did not use the regularization term.[4] The second, referred to as "non-guided," did not use the guiding samples during the policy search. The third variant also did not use the guiding samples in the policy objective, but used the guiding distributions as "restart distributions" to specify new initial state distributions that cover states we expect to visit under a good policy, analogously to prior work (Kakade & Langford, 2002; Bagnell et al., 2003). This comparison was meant to check that the guiding samples actually aided policy learning, rather than simply focusing the policy search on "good" states. All variants initialize the policy by pretraining on the guiding samples.

The first prior method, referred to as "single example," initialized the policy using the single initial example, as in prior work on learning from demonstration, and then improved the policy with importance sampling but no guiding samples, again as in prior work (Peshkin & Shelton, 2002). The second method used standard policy gradients, with a PGT or GPOMDP-type estimator (Sutton et al., 1999; Baxter et al., 2001). The third method was DAGGER, an imitation learning algorithm that aims to find a policy that matches the "expert" DDP actions (Ross et al., 2011).

The results are shown in Figure 2 in terms of the mean reward of the 10 samples at each iteration, along with the value of the initial example and a shaded interval indicating two standard deviations of the guiding sample values. Our algorithm learned each gait, matching the reward of the initial example. The comparison to the unregularized variant shows that the proposed regularization term is crucial for obtaining a good policy. The non-guided variant, which only used the guiding samples for pretraining, sometimes found a good solution because even a partially successful initial policy

[4]We also evaluated the ESS constraint proposed by Tang and Abbeel, but found that it performed no better on our tasks than the unregularized variant.

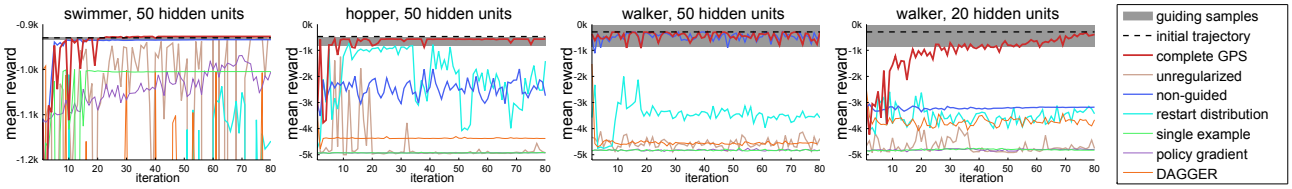
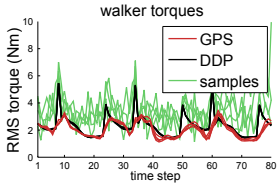


Figure 2. Comparison of guided policy search (GPS) with ablated variants and prior methods. Our method successfully learns each gait, while methods that do not use guiding samples or regularization fail to make progress. All methods use 10 rollouts per iteration. Guided variants (GPS, unregularized, restart, DAGGER) also use 40 guiding samples.

may succeed on one of its initial samples, which then serves the same function as the guiding samples by indicating high-reward regions. However, a successful non-guided outcome hinged entirely on the initial policy. This is illustrated by the fourth graph in Figure 2, which shows a walking policy with 20 hidden units that is less successful initially. The full algorithm was still able to improve using the guiding samples, while the non-guided variant did not make progress. The restart distribution variant performed poorly. Wider initial state distributions greatly increased the variance of the estimator, and because the guiding distributions were only used to sample states, their ability to also point out good actions was not leveraged.

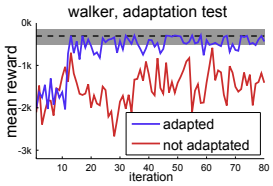
Standard policy gradient and "single example" learning from demonstration methods failed to learn any of the gaits. This suggests that guiding samples are crucial for learning such complex behaviors, and initialization from a single example is insufficient. DAGGER also performed poorly, since it assumed that the expert could provide optimal actions in all states, while the DDP policy was actually only valid close to the example. DAGGER therefore wasted a lot of effort on states where the DDP policy was not valid, such as after the hopper or walker fell.



Interestingly, the GPS policies often used less torque than the initial DDP solution. The plot on the right shows the torque magnitudes for the walking policy, the deterministic DDP policy around the example, and the guiding samples. The smoother, more compliant behavior of the learned policy can be explained in part by the fact that the neural network can produce more subtle corrections than simple linear feedback.

6.2. Adaptive Resampling

The initial trajectories in the previous section could all be reproduced by neural networks with sufficient training, making adaptive guiding distributions unnecessary. In the next experiment, we constructed a walking example that switched to another gait after 2.5s, making the initial guiding samples difficult to recreate with a stationary policy. As discussed in Section 4.2, adapting the guiding distribution to the policy can produce more suitable samples in such cases.



The plot on the right shows the results with and without adaptive samples. With adaptation, our algorithm quickly learned a successful gait, while without it, it made little progress.[5] The supplemental video shows that the final adapted example has a single gait that resembles both initial gaits. In practice, adaptation is useful if the examples are of low quality, or if the observations are insufficient to distinguish states where they takes different actions. As the examples are adapted, the policy encourages the DDP solution to be more regular. In future work, it would be interesting to see if this iterative process can find trajectories that are too complex to be found by either method alone.

6.3. Generalization

Next, we explore how our policies generalize to new environments. We trained a policy with 100 hidden units to walk 4m on flat ground, and then climb a 10° incline. The policy could not perceive the slope, and the root height was only given relative to the ground. We then moved the start of the incline and compared our method to the initial DDP policy, as well as a simplified trajectory-based dynamic programming (TBDP) approach that aggregates local DDP policies and uses the policy of the nearest neighbor to the current state (Atkeson & Stephens, 2008). Prior TBDP methods add new trajectories until the TBDP policy achieves good results. Since the initial DDP policy already succeeds on the training environment, only this initial policy was used. Both methods therefore used the same DDP solution: TBDP used it to build a nonparametric policy, and GPS used it to generate guiding samples.

[5]Note that the adapted variant used 20 samples per iteration: 10 on-policy and 10 adaptive guiding samples.

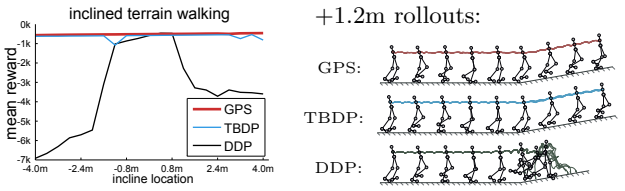


Figure 3. Comparison of GPS, TBDP, and DDP at varying incline locations (left) and plots of their rollouts (right). GPS and TBDP generalized to all incline locations.

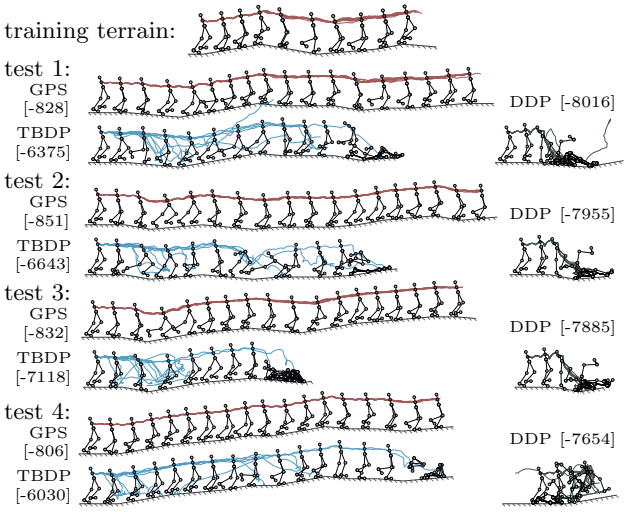


Figure 4. Rollouts of GPS, TBDP, and DDP on test terrains, shown as colored trajectories, with mean rewards in brackets. All DDP rollouts and most TBDP rollouts fall within a few steps, and all TBDP rollouts fall before reaching the end, while GPS generalizes successfully.

As shown in Figure 3, GPS generalized to all positions, while the DDP policy, which expected the incline to start at a specific time, could only climb it in a small interval around its original location. The TBDP solution was also able to generalize successfully.

In the second experiment, we trained a walking policy on terrain consisting of 1m and 2m segments with varying slopes, and tested on four random terrains. The results in Figure 4 show that GPS generalized to the new terrains, while the nearest-neighbor TBDP policy did not, with all rollouts eventually failing on each test terrain. Unlike TBDP, the GPS neural network learned generalizable rules for balancing on sloped terrain. While these rules might not generalize to much steeper inclines without additional training, they indicate a degree of generalization significantly greater than nearest-neighbor lookup. Example rollouts from each policy are shown in the supplemental video.[6]

[6]The videos can be viewed on the project website at http://graphics.stanford.edu/projects/gpspaper/index.htm

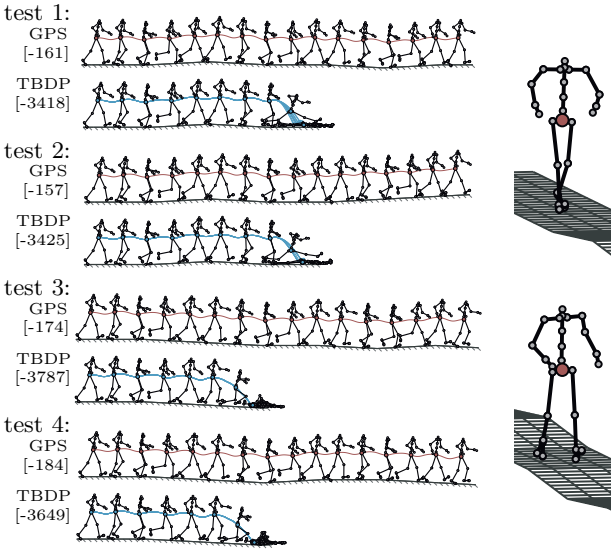


Figure 5. Humanoid running on test terrains, with mean rewards in brackets (left), along with illustrations of the 3D humanoid model (right). Our approach again successfully generalized to the test terrains, while the TBDP policy is unable to maintain balance.

6.4. Humanoid Running

In our final experiment, we used our method to learn a 3D humanoid running gait. This task is highly dynamic, requires good timing and balance, and has 63 state dimensions, making it well suited for exploring the capacity of our method to scale to high-dimensional problems. Previous locomotion methods often rely on hand-crafted components to introduce prior knowledge or ensure stability (Tedrake et al., 2004; Whitman & Atkeson, 2009), while our method used only general purpose neural networks.

As in the previous section, the policy was trained on terrain of varying slope, and tested on four random terrains. Due to the complexity of this task, we used three different training terrains and a policy with 200 hidden units. The motor noise was reduced from 10% to 1%. The guiding distributions were initialized from motion capture of a human run, and DDP was used to find the torques that realized this run on each training terrain. Since the example was recorded on flat ground, we used more mild 3° slopes.

Rollouts from the learned policy on the test terrains are shown in Figure 5, with comparisons to TBDP. Our method again generalized to all test terrains, while TBDP did not. This indicates that the task required nontrivial generalization (despite the mild slopes), and that GPS was able to learn generalizable rules to maintain speed and balance. As shown in the supplemental video, the learned gait also retained the smooth, life-like appearance of the human demonstration.

7. Previous Work

Policy gradient methods often require on-policy samples at each gradient step, do not admit off-policy samples, and cannot use line searches or higher order optimization methods such as LBFGS, which makes them difficult to use with complex policy classes (Peters & Schaal, 2008). Our approach follows prior methods that use importance sampling to address these challenges (Peshkin & Shelton, 2002; Kober & Peters, 2009; Tang & Abbeel, 2010). While these methods recycle samples from previous policies, we also introduce guiding samples, which dramatically speed up learning and help avoid poor local optima. We also regularize the importance sampling estimator, which prevents the optimization from assigning low probabilities to all samples. The regularizer controls how far the policy deviates from the samples, serving a similar function to the natural gradient, which bounds the information loss at each iteration (Peters & Schaal, 2008). Unlike Tang and Abbeel's ESS constraint (2010), our regularizer does not penalize reliance on a few samples, but does avoid policies that assign a low probability to all samples. Our evaluation shows that the regularizer can be crucial for learning effective policies.

Since the guiding distributions point out high reward regions to the policy search, we can also consider prior methods that explore high reward regions by using restart distributions for the initial state (Kakade & Langford, 2002; Bagnell et al., 2003). Our restart distribution variant is similar to Kakade and Langford with $\alpha = 1$. In our evaluation, this approach suffers from the high variance of the restart distribution, and is outperformed significantly by GPS.

We also compare to a nonparametric trajectory-based dynamic programming method. While several TBDP methods have been proposed (Atkeson & Morimoto, 2002; Atkeson & Stephens, 2008), we used a simple nearest-neighbor variant with a single trajectory, which is suitable for episodic tasks with a single initial state. Unlike GPS, TBDP cannot learn arbitrary parametric policies, and in our experiments, GPS exhibited better generalization on rough terrain.

The guiding distributions can be built from expert demonstrations. Many prior methods use expert examples to aid learning. Imitation methods such as DAGGER directly mimic the expert (Ross et al., 2011), while our approach maximizes the actual return of the policy, making it less vulnerable to suboptimal experts. DAGGER fails to make progress on our tasks, since it assumes that the DDP actions are optimal in all states, while they are only actually valid near the example. Additional DDP optimization from every visited state could produce better actions, but would be quadratic in the trajectory length, and would still not guarantee optimal actions, since the local DDP method cannot recover from all failures.

Other previous methods follow the examples directly (Abbeel et al., 2006), or use the examples for supervised initialization of special policy classes (Ijspeert et al., 2002). The former methods usually produce nonstationary feedback policies, which have limited capacity to generalize. In the latter approach, the policy class must be chosen carefully, as supervised learning does not guarantee that the policy will reproduce the examples: a small mistake early on could cause drastic deviations. Since our approach incorporates the guiding samples into the policy search, it does not rely on supervised learning to learn the examples, and can therefore use flexible, general-purpose policy classes.

8. Discussion and Future Work

We presented a guided policy search algorithm that can learn complex policies with hundreds of parameters by incorporating guiding samples into the policy search. These samples are drawn from a distribution built around a DDP solution, which can be initialized from demonstrations. We evaluated our method using general-purpose neural networks on a range of challenging locomotion tasks, and showed that the learned policies generalize to new environments.

While our policy search is model-free, it is guided by a model-based DDP algorithm. A promising avenue for future work is to build the guiding distributions with model-free methods that either build trajectory following policies (Ijspeert et al., 2002) or perform stochastic trajectory optimization (Kalakrishnan et al., 2011).

Our rough terrain results suggest that GPS can generalize by learning basic locomotion principles such as balance. Further investigation of generalization is an exciting avenue for future work. Generalization could be improved by training on multiple environments, or by using larger neural networks with multiple layers or recurrent connections. It would be interesting to see whether such extensions could learn more general and portable concepts, such as obstacle avoidance, perturbation recoveries, or even higher-level navigation skills.

Acknowledgements

We thank Emanuel Todorov, Tom Erez, and Yuval Tassa for providing the simulator used in our experiments. Sergey Levine was supported by NSF Graduate Research Fellowship DGE-0645962.

References

Abbeel, P., Coates, A., Quigley, M., and Ng, A. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in Neural Information Processing Systems (NIPS 19)*, 2006.

Atkeson, C. and Morimoto, J. Nonparametric representation of policies and value functions: A trajectory-based approach. In *Advances in Neural Information Processing Systems (NIPS 15)*, 2002.

Atkeson, C. and Stephens, B. Random sampling of states in dynamic programming. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 38(4): 924–929, 2008.

Bagnell, A., Kakade, S., Ng, A., and Schneider, J. Policy search by dynamic programming. In *Advances in Neural Information Processing Systems (NIPS 16)*, 2003.

Baxter, J., Bartlett, P., and Weaver, L. Experiments with infinite-horizon, policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15: 351–381, 2001.

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

Dvijotham, K. and Todorov, E. Inverse optimal control with linearly-solvable MDPs. In *International Conference on Machine Learning (ICML)*, 2010.

Ijspeert, A., Nakanishi, J., and Schaal, S. Movement imitation with nonlinear dynamical systems in humanoid robots. In *International Conference on Robotics and Automation*, 2002.

Kakade, S. and Langford, J. Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2002.

Kalakrishnan, M., Chitta, S., Theodorou, E., Pastor, P., and Schaal, S. STOMP: stochastic trajectory optimization for motion planning. In *International Conference on Robotics and Automation*, 2011.

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

Peshkin, L. and Shelton, C. Learning from scarce experience. In *International Conference on Machine Learning (ICML)*, 2002.

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

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. *Journal of Machine Learning Research*, 15:627–635, 2011.

Sutton, R., McAllester, D., Singh, S., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NIPS 11)*, 1999.

Tang, J. and Abbeel, P. On a connection between importance sampling and the likelihood ratio policy gradient. In *Advances in Neural Information Processing Systems (NIPS 23)*, 2010.

Tassa, Y., Erez, T., and Todorov, E. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

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

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

Whitman, E. and Atkeson, C. Control of a walking biped using a combination of simple policies. In *9th IEEE-RAS International Conference on Humanoid Robots*, 2009.

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. Objective Gradients

To compute the gradient $\nabla\Phi(\theta)$, we first write the gradient in terms of the gradients of $Z_t(\theta)$ and π_θ:

$$\nabla\Phi(\theta) = \sum_{t=1}^{T}\left[\frac{1}{Z_t(\theta)}\sum_{i=1}^{m}\frac{\nabla\pi_\theta(\zeta_{i,1:t})}{q(\zeta_{i,1:t})}r(\mathbf{x}_t^i,\mathbf{u}_t^i) - \frac{\nabla Z_t(\theta)}{Z_t(\theta)^2}\sum_{i=1}^{m}\frac{\pi_\theta(\zeta_{i,1:t})}{q(\zeta_{i,1:t})}r(\mathbf{x}_t^i,\mathbf{u}_t^i) + w_r\frac{\nabla Z_t(\theta)}{Z_t(\theta)}\right].$$

From the definition of $Z_t(\theta)$, we have that

$$\frac{\nabla Z_t(\theta)}{Z_t(\theta)} = \frac{1}{Z_t(\theta)}\sum_{i=1}^{m}\frac{\nabla\pi_\theta(\zeta_{i,1:t})}{q(\zeta_{i,1:t})}.$$

Letting $\tilde{J}_t(\theta) = \frac{1}{Z_t(\theta)}\sum_{i=1}^{m}\frac{\pi_\theta(\zeta_{i,1:t})}{q(\zeta_{i,1:t})}r(\mathbf{x}_t^i,\mathbf{u}_t^i)$, we can rewrite the gradient as

$$\begin{aligned}\nabla\Phi(\theta) &= \sum_{t=1}^{T}\frac{1}{Z_t(\theta)}\sum_{i=1}^{m}\frac{\nabla\pi_\theta(\zeta_{i,1:t})}{q(\zeta_{i,1:t})}\left[r(\mathbf{x}_t^i,\mathbf{u}_t^i) - \tilde{J}_t(\theta) + w_r\right] \\ &= \sum_{t=1}^{T}\frac{1}{Z_t(\theta)}\sum_{i=1}^{m}\frac{\pi_\theta(\zeta_{i,1:t})}{q(\zeta_{i,1:t})}\nabla\log\pi_\theta(\zeta_{i,1:t})\xi_t^i,\end{aligned}$$

using the identity $\nabla\pi_\theta(\zeta) = \pi_\theta(\zeta)\nabla\log\pi_\theta(\zeta)$. When the policy is represented by a large neural network, it is convenient to write the gradient as a sum where the output at each state appears only once, to produce a set of errors that can be fed into a standard back-propagation algorithm. For a neural network policy with uniform output noise σ and mean $\mu(\mathbf{x}_t)$, we have

$$\begin{aligned}\nabla\log\pi_\theta(\zeta_{i,1:t}) &= \sum_t \nabla\log\pi_\theta(\mathbf{u}_t|\mathbf{x}_t) \\ &= \sum_t \nabla\mu(\mathbf{x}_t)\frac{\mathbf{u}_t - \mu(\mathbf{x}_t)}{\sigma^2},\end{aligned}$$

and the gradient of the objective is given by

$$\nabla\Phi(\theta) = \sum_{t=1}^{T}\sum_{i=1}^{m}\nabla\mu(\mathbf{x}_t^i)\frac{\mathbf{u}_t^i - \mu(\mathbf{x}_t^i)}{\sigma^2}\sum_{t'=t}^{T}\frac{1}{Z_{t'}(\theta)}\frac{\pi_\theta(\zeta_{i,1:t'})}{q(\zeta_{i,1:t'})}\xi_{t'}^i.$$

The gradient can now be computed efficiently by feeding the terms after $\nabla\mu(\mathbf{x}_t^i)$ into the standard back-propagation algorithm.

B. Dynamic System Descriptions

This appendix describes the dynamical systems corresponding to the simulated robots in the swimming, hopping, and walking tasks. Images of each robot are provided in Figure 1 of the paper.

Swimmer: The swimmer is a 3-link snake, with 10 state dimensions for the position and angle of the head, the joint angles, and the corresponding velocities, as well as 2 action dimensions for the torques. The surrounding fluid applies a drag on each link, allowing the snake to propel itself. The simulation step is 0.05s, the reward weights are $w_{\mathbf{u}} = 0.0001$, $w_v = 1$, and $w_h = 0$, and the desired velocity is $v_x^\star = 2$m/s.

Hopper: The hopper has 4 links: torso, upper leg, lower leg, and foot. The state has 12 dimensions, and the actions have 3. To make it easier to optimize a gait with DDP, we employed a softened contact model as proposed in (Tassa et al., 2012). The reward weights are $w_{\mathbf{u}} = 0.001$, $w_v = 1$, and $w_h = 10$, and the desired velocity and height are $v_x^\star = 1.5$m/s and $p_y^\star = 1.5$m. A lower time step of 0.02s was used to handle contacts.

Walker: The walker has 7 links, corresponding to two legs and a torso, 18 state dimensions and 6 torques. The reward weights are $w_{\mathbf{u}} = 0.0001$, $w_v = 1$, and $w_h = 10$, and the desired velocity and height are $v_x^\star = 1.2$m/s and $p_y^\star = 1.5$m. The time step is 0.01s.

3D Humanoid: The humanoid consists of 13 links, with a free-floating 6 DoF base, 4 ball joints, 3 joints with 2 DoF, and 5 hinge joints, for a total of 29 degrees of freedom. Ball joints are represented by quaternions, while their velocities are represented by 3D vectors, so the entire model has 63 dimensions. The reward weights are are $w_{\mathbf{u}} = 0.00001$, $w_v = 1$, and $w_h = 10$, and the desired velocity and height are $v_x^\star = 2.5$m/s and $p_y^\star = 0.9$m. The time step is 0.01s. Due to the complexity of this model, the joint noise was reduced from 10% of example torque variance to 1%.