Variational Policy Search via Trajectory Optimization

Authored by:

Sergey Levine
Vladlen Koltun

Abstract

In order to learn effective control policies for dynamical systems, policy search methods must be able to discover successful executions of the desired task. While random exploration can work well in simple domains, complex and high-dimensional tasks present a serious challenge, particularly when combined with high-dimensional policies that make parameter-space exploration infeasible. We present a method that uses trajectory optimization as a powerful exploration strategy that guides the policy search. A variational decomposition of a maximum likelihood policy objective allows us to use standard trajectory optimization algorithms such as differential dynamic programming, interleaved with standard supervised learning for the policy itself. We demonstrate that the resulting algorithm can outperform prior methods on two challenging locomotion tasks.

1 Paper Body

Direct policy search methods have the potential to scale gracefully to complex, high-dimensional control tasks [12]. However, their effectiveness depends on discovering successful executions of the desired task, usually through random exploration. As the dimensionality and complexity of a task increases, random exploration can prove inadequate, resulting in poor local optima. We propose to decouple policy optimization from exploration by using a variational decomposition of a maximum likelihood policy objective. In our method, exploration is performed by a model-based trajectory optimization algorithm that is not constrained by the policy parameterization, but attempts to minimize both the cost and the deviation from the current policy, while the policy is simply optimized to match the resulting trajectory distribution. Since direct model-based trajectory optimization is usually much easier than policy search, this method can discover low cost regions much more easily. Intuitively, the trajectory optimization ?guides? the policy search toward regions of low cost. The trajectory optimization can be performed by a variant of the differential dynamic programming

algorithm [4], and the policy is optimized with respect to a standard maximum likelihood objective. We show that this alternating optimization maximizes a well-defined policy objective, and demonstrate experimentally that it can learn complex tasks in high-dimensional domains that are infeasible for methods that rely on random exploration. Our evaluation shows that the proposed algorithm produces good results on two challenging locomotion problems, outperforming prior methods.

2

Preliminaries

In standard policy search, we seek to find a distribution over actions ut in each state xt , denoted PT ?? (ut —xt), so as to minimize the sum of expected costs E[c(?)] = E[t=1 c(xt , ut)], where ? is a sequence of states and actions. The expectation is taken with respect to the system dynamics p(xt+1 —xt , ut) and the policy ?? (ut —xt), which is typically parameterized by a vector ?. An alternative to this standard formulation is to convert the task into an inference problem, by introducing a binary random variable Ot at each time step that serves as the indicator for ?optimality.? 1

We follow prior work and define the probability of Ot as p(Ot = 1—xt , ut) ? exp(?c(xt , ut)) [19]. Using the dynamics distribution p(xt+1 —xt , ut) and the policy ?? (ut —xt), we can define a dynamic Bayesian network that relates states, actions, and the optimality indicator. By setting Ot = 1 at all time steps and learning the maximum likelihood values for ?, we can perform policy optimization [20]. The corresponding optimization problem has the objective ! Z Z T T X Y p(O—?) = p(O—?)p(?—?)d? ? exp ? c(xt , ut) p(x1) ?? (ut —xt)p(xt+1 —xt , ut)d?. (1) t=1

t=1

Although this objective differs from the classical minimum average cost objective, previous work showed that it is nonetheless useful for policy optimization and planning [20, 19]. In Section 5, we discuss how this objective relates to the classical objective in more detail.

3

Variational Policy Search

Following prior work [11], we can decompose log p(O—?) by using a variational distribution q(?): log p(O—?) = L(q, ?) + DKL (q(?)kp(?—O, ?)), where the variational lower bound L is given by Z p(O—?)p(?—?) L(q, ?) = q(?) log d?, q(?) and the second term is the Kullback-Leibler (KL) divergence Z Z p(?—O, ?) p(O—?)p(?—?) DKL (q(?)kp(?—O, ?)) = ? q(?) log d? = ? q(?) log d?. q(?) q(?)p(O—?)

(2)

We can then optimize the maximum likelihood objective in Equation 1 by iteratively minimizing the KL divergence with respect to q(?) and maximizing the bound L(q, ?) with respect to ?. This is the standard formulation for expectation maximization [9], and has been applied to policy optimization in previous work [8, 21, 3, 11]. However, prior policy optimization methods typically represent q(?) by sampling trajectories from the current policy ?? (ut —xt) and reweighting them, for example by the exponential of their cost. While this

can improve policies that already visit regions of low cost, it relies on random policy-driven exploration to discover those low cost regions. We propose instead to directly optimize q(?) to minimize both its expected cost and its divergence from the current policy ?? (ut —xt) when a model of the dynamics is available. In the next section, we show that, for a Gaussian distribution q(?), the KL divergence in Equation 2 can be minimized by a variant of the differential dynamic programming (DDP) algorithm [4].

4

Trajectory Optimization

DDP is a trajectory optimization algorithm based on Newton?s method [4]. We build off of a variant of DDP called iterative LQR, which linearizes the dynamics around the current trajectory, computes the optimal linear policy under linear-quadratic assumptions, executes this policy, and repeats the process around the new trajectory until convergence [17]. We show how this procedure can be used to minimize the KL divergence in Equation 2 when q(?) is a Gaussian distribution over trajectories. This derivation follows previous work [10], but is repeated here and expanded for completeness. Iterative LQR is a dynamic programming algorithm that recursively computes the value function backwards through time. Because of the linear-quadratic assumptions, the value function is always quadratic, and the dynamics are Gaussian with the mean at f (xt , ut) and noise . Given a trajectory ? 1), . . . , (? ? T) and defining x ? t = xt ? x ? t and u ? t = ut ? u ? t , the dynamics and cost function (? x1 , u xT , u are then approximated as following, with subscripts x and u denoting partial derivatives: ? t+1 ? fxt x ? t + fut u ?t + x 1 T 1 T ?T ? T cut + x ? cxxt x ?t + u ? cuut u ?t + u ?T ? t + c(? ? t). c(xt , ut) ? x xt , u t cuxt x t cxt + u 2 t 2 t 2

Under this approximation, we can recursively compute the Q-function as follows: T Qxxt = cxxt +fxt Vxxt+1 fxt

T Quut = cuut +fut Vxxt+1 fut

T Qxt = cxt +fxt Vxt+1

T Qut = cut +fut Vxt+1 ,

T Quxt = cuxt +fut Vxxt+1 fxt

as well as the value function and linear policy terms: ?1 Vxt = Qxt ? QT uxt Quut Qu ?1 Vxxt = Qxxt ? QT uxt Quut Qux

kt = ?Q?1 uut Qut Kt = ?Q?1 uut Quxt .

The deterministic optimal policy is then given by ? t + kt + Kt (xt ? x ? t). g(xt) = u ? t and u ?t By repeatedly computing the optimal policy around the current trajectory and updating x based on the new policy, iterative LQR converges to a locally optimal solution [17]. In order to use this algorithm to minimize the KL divergence in Equation 2, we introduce a modified cost function c?(xt , ut) = c(xt , ut) ? log ?? (ut —xt). The optimal trajectory for this cost function approximately1 minimizes the KL divergence when q(?) is a Dirac delta function, since " T # Z X DKL (q(?)kp(?—O, ?)) = q(?) c(xt , ut) ? log ?? (ut —xt) ? log p(xt+1 —xt , ut) d? + const. t=1

However, we can also obtain a Gaussian q(?) by using the framework of linearly solvable MDPs [16] and the closely related concept of maximum en-

tropy control [23]. The optimal policy ?G under this framework minimizes an augmented cost function, given by c?(xt , ut) = c?(xt , ut) ? H(?G), where H(?G) is the entropy of a stochastic policy ?G (ut —xt), and c?(xt , ut) includes log ?? (ut —xt) as above. Ziebart [23] showed that the optimal policy can be written as ?G (ut —xt) = exp(?Qt (xt , ut) + Vt (xt)), where V is a ?softened? value function given by Z Vt (xt) = log exp (Qt (xt , ut)) dut . Under linear dynamics and quadratic costs, V has the same form as in the LQR derivation above, which means that ?G (ut —xt) is a linear Gaussian with mean g(xt) and covariance Q?1 uut [10]. Together with the linearized dynamics, the resulting policy specifies a Gaussian distribution over trajectories with Markovian independence: q(?) = p?(xt)

T Y

?G (ut —xt)? p(xt+1 —xt , ut),

t=1

where ?G (ut —xt) = N (g(xt), Q?1 ?(xt) is an initial state distribution, and p?(xt+1 —xt , ut) = uut), p ? t +fut u ?t +x ? t+1 , ?f t) is the linearized dynamics with Gaussian noise ?f t . This distribution N (fxt x also corresponds to a Laplace approximation for p(?—O, ?), which is formed from the exponential of the second order Taylor expansion of log p(?—O, ?) [15]. Once we compute ?G (ut —xt) using iterative LQR/DDP, it is straightforward to obtain the marginal distributions q(xt), which will be useful in the next section for minimizing the variational bound L(q, ?). Using ?t and ?t to denote the mean and covariance of the marginal at time t and assuming that the initial state distribution at t = 1 is given, the marginals can be computed recursively as

?t ?t+1 = [fxt fut] ? t + kt + Kt (?t ? x ?t) u

?t ?t KT T t ?t+1 = [fxt fut] [fxt fut] + ?f t . T Kt ?t Q?1 uut + Kt ?t Kt 1 The minimization is not exact if the dynamics p(xt+1 —xt , ut) are not deterministic, but the result is very close if the dynamics have much lower entropy than the policy and exponentiated cost, which is often the case.

3

Algorithm 1 Variational Guided Policy Search 1: Initialize q(?) using DDP with cost c?(xt , ut) = ?0 c(xt , ut) 2: for iteration k = 1 to K do 3: Compute marginals (?1 , ?t), . . . , (?T , ?T) for q(?) 4: Optimize L(q, ?) with respect to ? using standard nonlinear optimization methods

k 5: Set ?k based on annealing schedule, for example ?k = exp K?k K log ?0 + K log ?K 6: Optimize q(?) using DDP with cost c?(xt , ut) = ?k c(xt , ut) ? log ?? (ut —xt) 7: end for 8: Return optimized policy ?? (ut —xt)

When the dynamics are nonlinear or the modified cost c?(xt , ut) is non-quadratic, this solution only approximates the minimum of the KL divergence. In practice, the approximation is quite good when the dynamics and the cost c(xt , ut) are smooth. Unfortunately, the policy term log ?? (ut —xt) in the modified cost c?(xt , ut) can be quite jagged early on in the optimization, particularly for nonlinear policies. To mitigate this issue, we compute the derivatives of the policy not only along the current trajectory, but also at samples drawn from the current marginals q(xt), and average them together. This averages

out local perturbations in log ?? (ut —xt) and improves the approximation. In Section 8, we discuss more sophisticated techniques that could be used in future work to handle highly nonlinear dynamics for which this approximation may be inadequate.

5

Variational Guided Policy Search

The variational guided policy search (variational GPS) algorithm alternates between minimizing the KL divergence in Equation 2 with respect to q(?) as described in the previous section, and maximizing the bound L(q, ?) with respect to the policy parameters ?. Minimizing the KL divergence reduces the difference between L(q, ?) and log p(O—?), so that the maximization of L(q, ?) becomes a progressively better approximation for the maximization of log p(O—?). The method is summarized in Algorithm 1. The bound L(q, ?) can be maximized by a variety of standard optimization methods, such as stochastic gradient descent (SGD) or LBFGS. The gradient is given by Z T M T X 1 XX ?L(q, ?) = q(?) ? log ?? (ut —xt)d? ? ? log ?? (uit —xit), (3) M t=1 i=1 t=1 where the samples (xit , uit) are drawn from the marginals q(xt , ut). When using SGD, new samples can be drawn at every iteration, since sampling from q(xt , ut) only requires the precomputed marginals from the preceding section. Because the marginals are computed using linearized dynamics, we can be assured that the samples will not deviate drastically from the optimized trajectory, regardless of the true dynamics. The resulting SGD optimization is analogous to a supervised learning task with an infinite training set. When using LBFGS, a new sample set can generated every n LBFGS iterations. We found that values of n from 20 to 50 produced good results. When choosing the policy class, it is common to use deterministic policies with additive Gaussian noise. In this case, we can optimize the policy more quickly and with many fewer samples by only sampling states and evaluating the integral over actions analytically. Letting ??xt , ??xt and ?qxt , ?qxt denote the means and covariances of ?? (ut —xt) and q(ut —xt), we can write L(q, ?) as M T Z 1 XX L(q, ?) ? q(ut —xit) log ?? (ut —xit)dut + const M i=1 t=1

=

M T

1 T

1

1 XX 1 ?

q ? ?xi ? ?qxi ???1 ??xi ? ?qxi ? log ??xi ? tr ???1 + const. i ?x i x t x t t t t t t t t M i=1 t=1 2 2 2

Two additional details should be taken into account in order to obtain the best results. First, although model-based trajectory optimization is more powerful than random exploration, complex tasks such as bipedal locomotion, which we address in the following section, are too difficult to solve entirely with trajectory optimization. To solve such tasks, we can initialize the procedure from a good initial 4

trajectory, typically provided by a demonstration. This trajectory is only used for initialization and need not be reproducible by any policy, since it will be

modified by subsequent DDP invocations. Second, unlike the average cost objective, the maximum likelihood objective is sensitive to the magnitude of the cost. Specifically, the logarithm of Equation 1 corresponds to a soft minimum over all likely trajectories under the current policy, with the softness of the minimum inversely proportional to the cost magnitude. As the magnitude increases, this objective scores policies based primarily on their best-case cost, rather than the average case. As the magnitude decreases, the objective becomes more similar to the classic average cost. Because of this, we found it beneficial to gradually anneal the cost by multiplying it by ?k at the k th iteration, starting with a high magnitude to favor aggressive exploration, and ending with a low magnitude to optimize average case performance. In our experiments, ?k begins at 1 and is reduced exponentially to 0.1 by the 50th iteration. Since our method produces both a parameterized policy ?? (ut —xt) and a DDP solution ?G (ut —xt), one might wonder why the DDP policy itself is not a suitable controller. The issue is that ?? (ut —xt) can have an arbitrary parameterization, and admits constraints on available information, stationarity, etc., while ?G (ut —xt) is always a nonstationary linear feedback policy. This has three major advantages: first, only the learned policy may be usable at runtime if the information available at runtime differs from the information during training, for example if the policy is trained in simulation and executed on a physical system with limited sensors. Second, if the policy class is chosen carefully, we might hope that the learned policy would generalize better than the DDP solution, as shown in previous work [10]. Third, multiple trajectories can be used to train a single policy from different initial states, creating a single controller that can succeed in a variety of situations.

6

Experimental Evaluation

We evaluated our method on two simulated planar locomotion tasks: swimming and bipedal walking. For both tasks, the policy sets joint torques on a simulated robot consisting of rigid links. The swimmer has 3 links and 5 degrees of freedom, including the root position, and a 10-dimensional state space that includes joint velocities. The walker has 7 links, 9 degrees of freedom, and 18 state dimensions. Due to the high dimensionality and nonlinear dynamics, these tasks represent a significant challenge for direct policy learning. The cost function for the walker was given by c(x, u) = wu kuk2 + (vx ? vx?)2 + (py ? p?y)2 , where vx and vx? are the current and desired horizontal velocities, py and p?y are the current and desired heights of the hips, and the torque penalty was set to wu = 10?4 . The swimmer cost excludes the height term and uses a lower torque penalty of wu = 10?5 . As discussed in the previous section, the magnitude of the cost was decreased by a factor of 10 during the first 50 iterations, and then remained fixed. Following previous work [10], the trajectory for the walker was initialized with a demonstration from a hand-crafted locomotion system [22]. The policy was represented by a neural network with one hidden layer and a soft rectifying nonlinearity of the form a = log(1 + exp(z)), with Gaussian noise at the output. Both the weights of the neural network and the diagonal covariance of the output noise were learned as part of

the policy optimization. The number of policy parameters ranged from 63 for the 5-unit swimmer to 246 for the 10-unit walker. Due to its complexity and nonlinearity, this policy class presents a challenge to traditional policy search algorithms, which often focus on compact, linear policies [8]. Figure 1 shows the average cost of the learned policies on each task, along with visualizations of the swimmer and walker. Methods that sample from the current policy use 10 samples per iteration, unless noted otherwise. To ensure a fair comparison, the vertical axis shows the average cost E[c(?)] rather than the maximum likelihood objective log p(O—?). The cost was evaluated for both the actual stochastic policy (solid line), and a deterministic policy obtained by setting the variance of the Gaussian noise to zero (dashed line). Each plot also shows the cost of the initial DDP solution. Policies with costs significantly above this amount do not succeed at the task, either falling in the case of the walker, or failing to make forward progress in the case of the swimmer. Our method learned successful policies for each task, and often converged faster than previous methods, though performance during early iterations was often poor. We believe this is because the variational bound L(q, ?) does not become a good proxy for log p(O—?) until after several invocations of DDP, at which point the algorithm is able to rapidly improve the policy. 5

swimmer, 10 hidden units
400
350
average cost
average cost
400
300 250 200 150 100
4000
20
40
60
iteration
80
walker, 10 hidden units
250 200 150 20
40
60
iteration
80
100
DDP solution variational GPS GPS adapted GPS cost-weighted cost-weighted 1000 DAGGER weighted DAGGER adapted DAGGER
walker, 5 hidden units
3500
average cost
average cost
300

4000
3500 3000
3000
2500
2500
2000
2000
1500
1500
1000
1000
500 0
350
100
100
swimmer, 5 hidden units
20
40
60
iteration
80
100
500 0
20
40
60
iteration
80
100

Figure 1: Comparison of variational guided policy search (VGPS) with prior methods. The average cost of the stochastic policy is shown with a solid line, and the average cost of the deterministic policy without Gaussian noise is shown with a dashed line. The bottom-right panel shows plots of the swimmer and walker, with the center of mass trajectory under the learned policy shown in blue, and the initial DDP solution shown in black.

The first method we compare to is guided policy search (GPS), which uses importance sampling to introduce samples from the DDP solution into a likelihood ratio policy search [10]. The GPS algorithm first draws a fixed number of samples from the DDP solution, and then adds on-policy samples at each iteration. Like our method, GPS uses DDP to explore regions of low cost, but the policy optimization is done using importance sampling, which can be susceptible to degenerate weights in high dimensions. Since standard GPS only samples from the initial DDP solution, these samples are only useful if they can be reproduced by the policy class. Otherwise, GPS must rely on random exploration to improve the solution. On the easier swimmer task, the GPS policy can reproduce the initial trajectory and succeeds immediately. However, GPS

is unable to find a successful walking policy with only 5 hidden units, which requires modifications to the initial trajectory. In addition, although the deterministic GPS policy performs well on the walker with 10 hidden units, the stochastic policy fails more often. This suggests that the GPS optimization is not learning a good variance for the Gaussian policy, possibly because the normalized importance sampled estimator places greater emphasis on the relative probability of the samples than their absolute probability. The adaptive variant of GPS runs DDP at every iteration and adapts to the current policy, in the same manner as our method. However, samples from this adapted DDP solution are then included in the policy optimization with importance sampling, while our approach optimizes the variational bound L(q, ?). In the GPS estimator, each sample ?i is weighted by an importance weight dependent on ?? (?i), while the samples in our optimization are not weighted. When a sample has a low probability under the current policy, it is ignored by the importance sampled optimizer. Because of this, although the adaptive variant of GPS improves on the standard variant, it is still unable to learn a walking policy with 5 hidden units, while our method quickly discovers an effective policy. We also compared to an imitation learning method called DAGGER. DAGGER aims to learn a policy that imitates an oracle [14], which in our case is the DDP solution. At each iteration, DAGGER adds samples from the current policy to a dataset, and then optimizes the policy to take the oracle action at each dataset state. While adjusting the current policy to match the DDP solution may appear similar to our approach, we found that DAGGER performed poorly on these tasks, since the on-policy samples initially visited states that were very far from the DDP solution, and therefore the DDP action at these states was large and highly suboptimal. To reduce the impact of these poor states, we implemented a variant of DAGGER which weighted the samples by their probability under the DDP marginals. This variant succeeded on the swimming tasks and eventually found a good deterministic policy for the walker with 10 hidden units, though the learned stochastic policy performed very poorly. We also implemented an adapted variant, where the DDP solution is reoptimized at each iteration to match the policy (in addition to weighting), but this variant performed 6

worse. Unlike DAGGER, our method samples from a Gaussian distribution around the current DDP solution, ensuring that all samples are drawn from good parts of the state space. Because of this, our method is much less sensitive to poor or unstable initial policies. Finally, we compare to an alternative variational policy search algorithm analogous to PoWER [8]. Although PoWER requires a linear policy parameterization and a specific exploration strategy, we can construct an analogous non-linear algorithm by replacing the analytic M-step with nonlinear optimization, as in our method. This algorithm is identical to ours, except that instead of using DDP to optimize q(?), the variational distribution is formed by taking samples from the current policy and reweighting them by the exponential of their cost. We call this method ?cost-weighted.? The policy is still initialized with supervised training to resemble the initial DDP solution, but otherwise this method does not benefit from trajectory optimization and relies entirely on random exploration. This kind of exploration

is generally inadequate for such complex tasks. Even if the number of samples per iteration is increased to 103 (denoted as ?cost-weighted 1000?), this method still fails to solve the harder walking task, suggesting that simply taking more random samples is not the solution. These results show that our algorithm outperforms prior methods because of two advantages: we use a model-based trajectory optimization algorithm instead of random exploration, which allows us to outperform model-free methods such as the ?cost-weighted? PoWER analog, and we decompose the policy search into two simple optimization problems that can each be solved efficiently by standard algorithms, which leaves us less vulnerable to local optima than more complex methods like GPS.

7

Previous Work

In optimizing a maximum likelihood objective, our method builds on previous work that frames control as inference [20, 19, 13]. Such methods often redefine optimality in terms of a log evidence probability, as in Equation 1. Although this definition differs from the classical expected return, our evaluation suggests that policies optimized with respect to this measure also exhibit a good average return. As we discuss in Section 5, this objective is risk seeking when the cost magnitude is high, and annealing can be used to gradually transition from an objective that favors aggressive exploration to one that resembles the average return. Other authors have also proposed alternative definitions of optimality that include appealing properties like maximization of entropy [23] or computational benefits [16]. However, our work is the first to our knowledge to show how trajectory optimization can be used to guide policy learning within the control-as-inference framework. Our variational decomposition follows prior work on policy search with variational inference [3, 11] and expectation maximization [8, 21]. Unlike these methods, our approach aims to find a variational distribution q(?) that is best suited for control and leverages a known dynamics model. We present an interpretation of the KL divergence minimization in Equation 2 as model-based exploration, which can be performed with a variant of DDP. As shown in our evaluation, this provides our method with a significant advantage over methods that rely on model-free random exploration, though at the cost of requiring a differentiable model of the dynamics. Interestingly, our algorithm never requires samples to be drawn from the current policy. This can be an advantage in applications where running an unstable, incompletely optimized policy can be costly or dangerous. Our use of DDP to guide the policy search parallels our previous Guided Policy Search (GPS) algorithm [10]. Unlike the proposed method, GPS incorporates samples from DDP directly into an importance-sampled estimator of the return. These samples are therefore only useful when the policy class can reproduce them effectively. As shown in the evaluation of the walker with 5 hidden units, GPS may be unable to discover a good policy when the policy class cannot reproduce the initial DDP solution. Adaptive GPS addresses this issue by reoptimizing the trajectory to resemble the current policy, but the policy is still optimized with respect to an importance-sampled return estimate, which leaves it highly prone to local optima, and the theoretical justification for adaptation is unclear. The pro-

posed method justifies the reoptimization of the trajectory under a variational framework, and uses standard maximum likelihood in place of the complex importance-sampled objective. We also compared our method to DAGGER [14], which uses a general-purpose supervised training algorithm to train the current policy to match an oracle, which in our case is the DDP solution. DAGGER matches actions from the oracle policy at states visited by the current policy, under the 7

assumption that the oracle can provide good actions in all states. This assumption does not hold for DDP, which is only valid in a narrow region around the trajectory. To mitigate the locality of the DDP solution, we weighted the samples by their probability under the DDP marginals, which allowed DAGGER to solve the swimming task, but it was still outperformed by our method on the walking task, even with adaptation of the DDP solution. Unlike DAGGER, our approach is relatively insensitive to the instability of the learned policy, since the learned policy is not sampled. Several prior methods also propose to improve policy search by using a distribution over high-value states, which might come from a DDP solution [6, 1]. Such methods generally use this ?restart? distribution as a new initial state distribution, and show that optimizing a policy from such a restart distribution also optimizes the expected return. Unlike our approach, such methods only use the states from the DDP solution, not the actions, and tend to suffer from the increased variance of the restart distribution, as shown in previous work [10].

8

Discussion and Future Work

We presented a policy search algorithm that employs a variational decomposition of a maximum likelihood objective to combine trajectory optimization with policy search. The variational distribution is obtained using differential dynamic programming (DDP), and the policy can be optimized with a standard nonlinear optimization algorithm. Model-based trajectory optimization effectively takes the place of random exploration, providing a much more effective means for finding low cost regions that the policy is then trained to visit. Our evaluation shows that this algorithm outperforms prior variational methods and prior methods that use trajectory optimization to guide policy search. Our algorithm has several interesting properties that distinguish it from prior methods. First, the policy search does not need to sample the learned policy. This may be useful in real-world applications where poor policies might be too risky to run on a physical system. More generally, this property improves the robustness of our method in the face of unstable initial policies, where on-policy samples have extremely high variance. By sampling directly from the Gaussian marginals of the DDP-induced distribution over trajectories, our approach also avoids some of the issues associated with unstable dynamics, requiring only that the task permit effective trajectory optimization. By optimizing a maximum likelihood objective, our method favors policies with good best-case performance. Obtaining good best-case performance is often the hardest part of policy search, since a policy that achieves good results occasionally is easier to improve with standard on-policy search methods than one that fails outright. However, modifying the

algorithm to optimize the standard average cost criterion could produce more robust controllers in the future. The use of local linearization in DDP results in only approximate minimization of the KL divergence in Equation 2 in nonlinear domains or with nonquadratic policies. While we mitigate this by averaging the policy derivatives over multiple samples from the DDP marginals, this approach could still break down in the presence of highly nonsmooth dynamics or policies. An interesting avenue for future work is to extend the trajectory optimization method to nonsmooth domains by using samples rather than linearization, perhaps analogously to the unscented Kalman filter [5, 18]. This could also avoid the need to differentiate the policy with respect to the inputs, allowing for richer policy classes to be used. Another interesting avenue for future work is to apply model-free trajectory optimization techniques [7], which would avoid the need for a model of the system dynamics, or to learn the dynamics from data, for example by using Gaussian processes [2]. It would also be straightforward to use multiple trajectories optimized from different initial states to learn a single policy that is able to succeed under a variety of initial conditions. Overall, we believe that trajectory optimization is a very useful tool for policy search. By separating the policy optimization and exploration problems into two separate phases, we can employ simpler algorithms such as SGD and DDP that are better suited for each phase, and can achieve superior performance on complex tasks. We believe that additional research into augmenting policy learning with trajectory optimization can further advance the performance of policy search techniques. Acknowledgments 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. 8

2 References

[1] A. Bagnell, S. Kakade, A. Ng, and J. Schneider. Policy search by dynamic programming. In Advances in Neural Information Processing Systems (NIPS), 2003. [2] M. Deisenroth and C. Rasmussen. PILCO: a model-based and data-efficient approach to policy search. In International Conference on Machine Learning (ICML), 2011. [3] T. Furmston and D. Barber. Variational methods for reinforcement learning. Journal of Machine Learning Research, 9:241?248, 2010. [4] D. Jacobson and D. Mayne. Differential Dynamic Programming. Elsevier, 1970. [5] S. Julier and J. Uhlmann. A new extension of the Kalman filter to nonlinear systems. In International Symposium on Aerospace/Defense Sensing, Simulation, and Control, 1997. [6] S. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In International Conference on Machine Learning (ICML), 2002. [7] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. STOMP: stochastic trajectory optimization for motion planning. In International Conference on Robotics and Automation, 2011. [8] J. Kober and J. Peters. Learning motor primitives for robotics. In International Conference on Robotics and Automation, 2009. [9] D. Koller and N. Friedman. Probabilistic Graphical Models: Principles and Techniques. MIT Press, 2009.

[10] S. Levine and V. Koltun. Guided policy search. In International Conference on Machine Learning (ICML), 2013. [11] G. Neumann. Variational inference for policy search in changing situations. In International Conference on Machine Learning (ICML), 2011. [12] J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. Neural Networks, 21(4):682?697, 2008. [13] K. Rawlik, M. Toussaint, and S. Vijayakumar. On stochastic optimal control and reinforcement learning by approximate inference. In Robotics: Science and Systems, 2012. [14] S. Ross, G. Gordon, and A. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. Journal of Machine Learning Research, 15:627?635, 2011. [15] L. Tierney and J. B. Kadane. Accurate approximations for posterior moments and marginal densities. Journal of the American Statistical Association, 81(393):82?86, 1986. [16] E. Todorov. Policy gradients in linearly-solvable MDPs. In Advances in Neural Information Processing Systems (NIPS 23), 2010. [17] E. Todorov and W. Li. A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In American Control Conference, 2005. [18] E. Todorov and Y. Tassa. Iterative local dynamic programming. In IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), 2009. [19] M. Toussaint. Robot trajectory optimization using approximate inference. In International Conference on Machine Learning (ICML), 2009. [20] M. Toussaint, L. Charlin, and P. Poupart. Hierarchical POMDP controller optimization by likelihood maximization. In Uncertainty in Artificial Intelligence (UAI), 2008. [21] N. Vlassis, M. Toussaint, G. Kontes, and S. Piperidis. Learning model-free robot control by a Monte Carlo EM algorithm. Autonomous Robots, 27(2):123?130, 2009. [22] K. Yin, K. Loken, and M. van de Panne. SIMBICON: simple biped locomotion control. ACM Transactions Graphics, 26(3), 2007. [23] B. Ziebart. Modeling purposeful adaptive behavior with the principle of maximum causal entropy. PhD thesis, Carnegie Mellon University, 2010.

9