Continuous Deep Q-Learning with Model-based Acceleration

Shixiang Gu[1 2 3] SG717@CAM.AC.UK
Timothy Lillicrap[4] COUNTZERO@GOOGLE.COM
Ilya Sutskever[3] ILYASU@GOOGLE.COM
Sergey Levine[3] SLEVINE@GOOGLE.COM
[1]University of Cambridge [2]Max Planck Institute for Intelligent Systems [3]Google Brain [4]Google DeepMind

Abstract

Model-free reinforcement learning has been successfully applied to a range of challenging problems, and has recently been extended to handle large neural network policies and value functions. However, the sample complexity of model-free algorithms, particularly when using high-dimensional function approximators, tends to limit their applicability to physical systems. In this paper, we explore algorithms and representations to reduce the sample complexity of deep reinforcement learning for continuous control tasks. We propose two complementary techniques for improving the efficiency of such algorithms. First, we derive a continuous variant of the Q-learning algorithm, which we call normalized adantage functions (NAF), as an alternative to the more commonly used policy gradient and actor-critic methods. NAF representation allows us to apply Q-learning with experience replay to continuous tasks, and substantially improves performance on a set of simulated robotic control tasks. To further improve the efficiency of our approach, we explore the use of learned models for accelerating model-free reinforcement learning. We show that iteratively refitted local linear models are especially effective for this, and demonstrate substantially faster learning on domains where such models are applicable.

1. Introduction

Model-free reinforcement learning (RL) has been successfully applied to a range of challenging problems (Kober & Peters, 2012; Deisenroth et al., 2013), and has recently been extended to handle large neural network policies and value functions (Mnih et al., 2015; Lillicrap et al., 2016; Wang et al., 2015; Heess et al., 2015; Hausknecht & Stone, 2015; Schulman et al., 2015). This makes it possible to train policies for complex tasks with minimal feature and policy engineering, using the raw state representation directly as input to the neural network. However, the sample complexity of model-free algorithms, particularly when using very high-dimensional function approximators, tends to be high (Schulman et al., 2015), which means that the benefit of reduced manual engineering and greater generality is not felt in real-world domains where experience must be collected on real physical systems, such as robots and autonomous vehicles. In such domains, the methods of choice have been efficient model-free algorithms that use more suitable, task-specific representations (Peters et al., 2010; Deisenroth et al., 2013), as well as model-based algorithms that learn a model of the system with supervised learning and optimize a policy under this model (Deisenroth & Rasmussen, 2011; Levine et al., 2015). Using task-specific representations dramatically improves efficiency, but limits the range of tasks that can be learned and requires greater domain knowledge. Using model-based RL also improves efficiency, but limits the policy to only be as good as the learned model. For many real-world tasks, it may be easier to represent a good policy than to learn a good model. For example, a simple robotic grasping behavior might only require closing the fingers at the right moment, while the corresponding dynamics model requires learning the complexities of rigid and deformable bodies undergoing frictional contact. It is therefore desirable to bring the generality of model-free deep reinforcement learning into real-world domains by reducing their sample complexity.

In this paper, we propose two complementary techniques for improving the efficiency of deep reinforcement learning in continuous control domains: we derive a variant of Q-learning that can be used in continuous domains, and we propose a method for combining this continuous Q-learning algorithm with learned models so as to accelerate learning while preserving the benefits of model-free RL. Model-free reinforcement learning in domains with continuous actions is typically handled with policy search methods (Peters & Schaal, 2006; Peters et al., 2010). Integrating value function estimation into these techniques results in actor-critic algorithms (Hafner & Riedmiller, 2011; Lillicrap et al., 2016; Schulman et al., 2015), which combine the benefits of policy search and value function estimation, but at the cost of training two separate function approximators. Our proposed Q-learning algorithm for continuous domains, which we call normalized advantage functions (NAF), avoids the need for a second actor or policy function, resulting in a simpler algorithm. The simpler optimization objective and the choice of value function parameterization result in an algorithm that is substantially more sample-efficient when used with large neural network function approximators on a range of continuous control domains.

Beyond deriving an improved model-free deep reinforcement learning algorithm, we also seek to incorporate elements of model-based RL to accelerate learning, without giving up the strengths of model-free methods. One approach is for off-policy algorithms such as Q-learning to incorporate off-policy experience produced by a model-based planner. However, while this solution is a natural one, our empirical evaluation shows that it is ineffective at accelerating learning. As we discuss in our evaluation, this is due in part to the nature of value function estimation algorithms, which must experience both good and bad state transitions to accurately model the value function landscape. We propose an alternative approach to incorporating learned models into our continuous-action Q-learning algorithm based on *imagination rollouts*: on-policy samples generated under the learned model, analogous to the Dyna-Q method (Sutton, 1990). We show that this is extremely effective when the learned dynamics model perfectly matches the true one, but degrades dramatically with imperfect learned models. However, we demonstrate that iteratively fitting local linear models to the latest batch of on-policy or off-policy rollouts provides sufficient *local* accuracy to achieve substantial improvement using short imagination rollouts in the vicinity of the real-world samples.

Our paper provides three main contributions: first, we derive and evaluate a Q-function representation that allows for effective Q-learning in continuous domains. Second, we evaluate several naïve options for incorporating learned models into model-free Q-learning, and we show that they are minimally effective on our continuous control tasks. Third, we propose to combine locally linear models with local on-policy imagination rollouts to accelerate model-free continuous Q-learning, and show that this produces a large improvement in sample complexity. We evaluate our method on a series of simulated robotic tasks and compare to prior methods.

2. Related Work

Deep reinforcement learning has received considerable attention in recent years due to its potential to automate the design of representations in RL. Deep reinforcement learning and related methods have been applied to learn policies to play Atari games (Mnih et al., 2015; Schaul et al., 2015) and perform a wide variety of simulated and real-world robotic control tasks (Hafner & Riedmiller, 2011; Lillicrap et al., 2016; Levine & Koltun, 2013; de Bruin et al., 2015; Hafner & Riedmiller, 2011). While the majority of deep reinforcement learning methods in domains with discrete actions, such as Atari games, are based around value function estimation and Q-learning (Mnih et al., 2015), continuous domains typically require explicit representation of the policy, for example in the context of a policy gradient algorithm (Schulman et al., 2015). If we wish to incorporate the benefits of value function estimation into continuous deep reinforcement learning, we must typically use two networks: one to represent the policy, and one to represent the value function (Schulman et al., 2015; Lillicrap et al., 2016). In this paper, we instead describe how the simplicity and elegance of Q-learning can be ported into continuous domains, by learning a single network that outputs both the value function and policy. Our Q-function representation is related to dueling networks (Wang et al., 2015), though our approach applies to continuous action domains. Our empirical evaluation demonstrates that our continuous Q-learning algorithm achieves faster and more effective learning on a set of benchmark tasks compared to continuous actor-critic methods, and we believe that the simplicity of this approach will make it easier to adopt in practice. Our Q-learning method is also related to the work of Rawlik et al. (2013), but the form of our Q-function update is more standard.

As in standard RL, model-based deep reinforcement learning methods have generally been more efficient (Li & Todorov, 2004; Watter et al., 2015; Li & Todorov, 2004; Wahlström et al., 2015; Levine & Koltun, 2013), while model-free algorithms tend to be more generally applicable but substantially slower (Schulman et al., 2015; Lillicrap et al., 2016). Combining model-based and model-free learning has been explored in several ways in the literature. The method closest to our imagination rollouts approach is Dyna-Q (Sutton, 1990), which uses simulated experience in a learned model to supplement real-world on-policy rollouts. As we show in our evaluation, using Dyna-Q style methods to accelerate model-free RL is very effective when the learned model perfectly matches the true model, but degrades rapidly as the model becomes worse. We demonstrate that using iteratively refitted local linear models achieves substantially better results with imagination rollouts than more complex neural network models. We hypothesize that this is likely due to the fact that the more expressive models themselves require substantially more data, and that otherwise efficient algorithms like Dyna-Q are vulnerable to poor model approximations.

3. Background

In reinforcement learning, the goal is to learn a policy to control a system with states $\boldsymbol{x} \in \mathcal{X}$ and actions $\boldsymbol{u} \in \mathcal{U}$ in environment E, so as to maximize the expected sum of returns according to a reward function $r(\boldsymbol{x}, \boldsymbol{u})$. The dynamical system is defined by an initial state distribution $p(\boldsymbol{x}_1)$ and a dynamics distribution $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$. At each time step $t \in [1, T]$, the agent chooses an action $\boldsymbol{u}_t$ according to its current policy $\pi(\boldsymbol{u}_t|\boldsymbol{x}_t)$, and observes a reward $r(\boldsymbol{x}_t, \boldsymbol{u}_t)$. The agent then experiences a transition to a new state sampled from the dynamics distribution, and we can express the resulting state visitation frequency of the policy π as $\rho^\pi(\boldsymbol{x}_t)$. Define $R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r(\boldsymbol{x}_i, \boldsymbol{u}_i)$, the goal is to maximize the expected sum of returns, given by $R = \mathbb{E}_{r_{i\geq 1}, \boldsymbol{x}_{i\geq 1} \sim E, \boldsymbol{u}_{i\geq 1} \sim \pi}[R_1]$, where γ is a discount factor that prioritizes earlier rewards over later ones. With $\gamma < 1$, we can also set $T = \infty$, though we use a finite horizon for all of the tasks in our experiments. The expected return R can be optimized using a variety of model-free and model-based algorithms. In this section, we review several of these methods that we build on in our work.

Model-Free Reinforcement Learning. When the system dynamics $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$ are not known, as is often the case with physical systems such as robots, policy gradient methods (Peters & Schaal, 2006) and value function or Q-function learning with function approximation (Sutton et al., 1999) are often preferred. Policy gradient methods provide a simple, direct approach to RL, which can succeed on high-dimensional problems, but potentially requires a large number of samples (Schulman et al., 2015; 2016). Off-policy algorithms that use value or Q-function approximation can in principle achieve better data efficiency (Lillicrap et al., 2016). However, adapting such methods to continuous tasks typically requires optimizing two function approximators on different objectives. We instead build on standard Q-learning, which has a single objective. We summarize Q-learning in this section. The Q function $Q^\pi(\boldsymbol{x}_t, \boldsymbol{u}_t)$ corresponding to a policy π is defined as the expected return from $\boldsymbol{x}_t$ after taking action $\boldsymbol{u}_t$ and following the policy π thereafter:

$$Q^\pi(\boldsymbol{x}_t, \boldsymbol{u}_t) = \mathbb{E}_{r_{i\geq t}, \boldsymbol{x}_{i>t} \sim E, \boldsymbol{u}_{i>t} \sim \pi}[R_t|\boldsymbol{x}_t, \boldsymbol{u}_t] \quad (1)$$

Q-learning learns a greedy deterministic policy $\boldsymbol{\mu}(\boldsymbol{x}_t) = \arg\max_{\boldsymbol{u}} Q(\boldsymbol{x}_t, \boldsymbol{u}_t)$, which corresponds to $\pi(\boldsymbol{u}_t|\boldsymbol{x}_t) = \delta(\boldsymbol{u}_t = \boldsymbol{\mu}(\boldsymbol{x}_t))$. Let θ^Q parametrize the action-value function, and β be an arbitrary exploration policy, the learning objective is to minimize the Bellman error, where we fix the target y_t:

$$\begin{aligned} L(\theta^Q) &= \mathbb{E}_{\boldsymbol{x}_t \sim \rho^\beta, \boldsymbol{u}_t \sim \beta, r_t \sim E}[(Q(\boldsymbol{x}_t, \boldsymbol{u}_t|\theta^Q) - y_t)^2] \\ y_t &= r(\boldsymbol{x}_t, \boldsymbol{u}_t) + \gamma Q(\boldsymbol{x}_{t+1}, \boldsymbol{\mu}(\boldsymbol{x}_{t+1})) \end{aligned} \quad (2)$$

For continuous action problems, Q-learning becomes difficult, because it requires maximizing a complex, nonlinear function at each update. For this reason, continuous domains are often tackled using actor-critic methods (Konda & Tsitsiklis, 1999; Hafner & Riedmiller, 2011; Silver et al., 2014; Lillicrap et al., 2016), where a separate parameterized "actor" policy π is learned in addition to the Q-function or value function "critic," such as Deep Deterministic Policy Gradient (DDPG) algorithm (Lillicrap et al., 2016).

In order to describe our method in the following sections, it will be useful to also define the value function $V^\pi(\boldsymbol{x}_t, \boldsymbol{u}_t)$ and advantage function $A^\pi(\boldsymbol{x}_t, \boldsymbol{u}_t)$ of a given policy π:

$$\begin{aligned} V^\pi(\boldsymbol{x}_t) &= \mathbb{E}_{r_{i\geq t}, \boldsymbol{x}_{i>t} \sim E, \boldsymbol{u}_{i\geq t} \sim \pi}[R_t|\boldsymbol{x}_t, \boldsymbol{u}_t] \\ A^\pi(\boldsymbol{x}_t, \boldsymbol{u}_t) &= Q^\pi(\boldsymbol{x}_t, \boldsymbol{u}_t) - V^\pi(\boldsymbol{x}_t). \end{aligned} \quad (3)$$

Model-Based Reinforcement Learning. If we know the dynamics $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$, or if we can approximate them with some learned model $\hat{p}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$, we can use model-based RL and optimal control. While a wide range of model-based RL and control methods have been proposed in the literature (Deisenroth et al., 2013; Kober & Peters, 2012), two are particularly relevant for this work: iterative LQG (iLQG) (Li & Todorov, 2004) and Dyna-Q (Sutton, 1990). The iLQG algorithm optimizes trajectories by iteratively constructing locally optimal linear feedback controllers under a local linearization of the dynamics $\hat{p}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t) = \mathcal{N}(\boldsymbol{f}_{\boldsymbol{x}t}\boldsymbol{x}_t + \boldsymbol{f}_{\boldsymbol{u}t}\boldsymbol{u}_t, \boldsymbol{F}_t)$ and a quadratic expansion of the rewards $r(\boldsymbol{x}_t, \boldsymbol{u}_t)$ (Tassa et al., 2012). Under linear dynamics and quadratic rewards, the action-value function $Q(\boldsymbol{x}_t, \boldsymbol{u}_t)$ and value function $V(\boldsymbol{x}_t)$ are locally quadratic and can be computed by dynamics programming. The optimal policy can be derived analytically from the quadratic $Q(\boldsymbol{x}_t, \boldsymbol{u}_t)$ and $V(\boldsymbol{x}_t)$ functions, and corresponds to a linear feedback controller $\boldsymbol{g}(\boldsymbol{x}_t) = \hat{\boldsymbol{u}}_t + \boldsymbol{k}_t + \boldsymbol{K}_t(\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t)$, where $\boldsymbol{k}_t$ is an open-loop term, $\boldsymbol{K}_t$ is the closed-loop feedback matrix, and $\hat{\boldsymbol{x}}_t$ and $\hat{\boldsymbol{u}}_t$ are the states and actions of the nominal trajectory, which is the average trajectory of the controller. Employing the maximum entropy objective (Levine & Koltun, 2013), we can also construct a linear-Gaussian controller, where c is a scalar to adjust for arbitrary scaling of the reward magnitudes:

$$\pi_t^{iLQG}(\boldsymbol{u}_t|\boldsymbol{x}_t) = \mathcal{N}(\hat{\boldsymbol{u}}_t + \boldsymbol{k}_t + \boldsymbol{K}_t(\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t), -cQ_{\boldsymbol{u},\boldsymbol{u}t}^{-1}) \quad (4)$$

When the dynamics are not known, a particularly effective way to use iLQG is to combine it with learned time-varying linear models $\hat{p}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$. In this variant of the algorithm, trajectories are sampled from the controller in Equation (4) and used to fit time-varying linear dynamics with linear regression. These dynamics are then used with iLQG to obtain a new controller, typically using a KL-divergence constraint to enforce a trust region, so that the new controller doesn't deviate too much from the region in which the samples were generated (Levine & Abbeel, 2014).

Besides enabling iLQG and other planning-based algorithms, a learned model of the dynamics can allow a model-free algorithm to generate synthetic experience by performing rollouts in the learned model. A particularly relevant method of this type is Dyna-Q (Sutton, 1990), which performs real-world rollouts using the policy π, and then generates synthetic rollouts using a model learned from these samples. The synthetic rollouts originate at states visited by the real-world rollouts, and serve as supplementary data for a variety of possible reinforcement learning algorithms. However, most prior Dyna-Q methods have focused on relatively small, discrete domains. In Section 5, we describe how our method can be extended into a variant of Dyna-Q to achieve substantially faster learning on a range of continuous control tasks with complex neural network policies, and in Section 6, we empirically analyze the sensitivity of this method to imperfect learned dynamics models.

4. Continuous Q-Learning with Normalized Advantage Functions

We first propose a simple method to enable Q-learning in continuous action spaces with deep neural networks, which we refer to as normalized advantage functions (NAF). The idea behind normalized advantage functions is to represent the Q-function $Q(\boldsymbol{x}_t, \boldsymbol{u}_t)$ in Q-learning in such a way that its maximum, $\arg\max_{\boldsymbol{u}} Q(\boldsymbol{x}_t, \boldsymbol{u}_t)$, can be determined easily and analytically during the Q-learning update. While a number of representations are possible that allow for analytic maximization, the one we use in our implementation is based on a neural network that separately outputs a value function term $V(\boldsymbol{x})$ and an advantage term $A(\boldsymbol{x}, \boldsymbol{u})$, which is parameterized as a quadratic function of nonlinear features of the state:

$$Q(\boldsymbol{x}, \boldsymbol{u}|\theta^Q) = A(\boldsymbol{x}, \boldsymbol{u}|\theta^A) + V(\boldsymbol{x}|\theta^V)$$

$$A(\boldsymbol{x}, \boldsymbol{u}|\theta^A) = -\frac{1}{2}(\boldsymbol{u} - \boldsymbol{\mu}(\boldsymbol{x}|\theta^\mu))^T \boldsymbol{P}(\boldsymbol{x}|\theta^P)(\boldsymbol{u} - \boldsymbol{\mu}(\boldsymbol{x}|\theta^\mu))$$

$\boldsymbol{P}(\boldsymbol{x}|\theta^P)$ is a state-dependent, positive-definite square matrix, which is parametrized by $\boldsymbol{P}(\boldsymbol{x}|\theta^P) = \boldsymbol{L}(\boldsymbol{x}|\theta^P)\boldsymbol{L}(\boldsymbol{x}|\theta^P)^T$, where $\boldsymbol{L}(\boldsymbol{x}|\theta^P)$ is a lower-triangular matrix whose entries come from a linear output layer of a neural network, with the diagonal terms exponentiated. While this representation is more restrictive than a general neural network function, since the Q-function is quadratic in $\boldsymbol{u}$, the action that maximizes the Q-function is always given by $\boldsymbol{\mu}(\boldsymbol{x}|\theta^\mu)$. We use this representation with a deep Q-learning algorithm analogous to Mnih et al. (2015), using target networks and a replay buffers as described by (Lillicrap et al., 2016). NAF, given by Algorithm 1, is considerably simpler than DDPG.

Algorithm 1 Continuous Q-Learning with NAF

```
Randomly initialize normalized Q network $Q(\boldsymbol{x}, \boldsymbol{u}|\theta^Q)$.
Initialize target network $Q'$ with weight $\theta^{Q'} \leftarrow \theta^Q$.
Initialize replay buffer $R \leftarrow \emptyset$.
for episode=1, $M$ do
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state $\boldsymbol{x}_1 \sim p(\boldsymbol{x}_1)$
  for t=1, $T$ do
    Select action $\boldsymbol{u}_t = \mu(\boldsymbol{x}_t|\theta^\mu) + \mathcal{N}_t$
    Execute $\boldsymbol{u}_t$ and observe $r_t$ and $\boldsymbol{x}_{t+1}$
    Store transition $(\boldsymbol{x}_t, \boldsymbol{u}_t, r_t, \boldsymbol{x}_{t+1})$ in $R$
    for iteration=1, $I$ do
      Sample a random minibatch of $m$ transitions from $R$
      Set $y_i = r_i + \gamma V'(\boldsymbol{x}_{i+1}|\theta^{Q'})$
      Update $\theta^Q$ by minimizing the loss: $L = \frac{1}{N}\sum_i(y_i - Q(\boldsymbol{x}_i, \boldsymbol{u}_i|\theta^Q))^2$
      Update the target network: $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
    end for
  end for
end for
```

Decomposing Q into an advantage term A and a state-value term V was suggested by Baird III (1993); Harmon & Baird III (1996), and was recently explored by Wang et al. (2015) for discrete action problems. Normalized action-value functions have also been proposed by Rawlik et al. (2013) in the context of an alternative temporal difference learning algorithm. However, our method is the first to combine such representations with deep neural networks into an algorithm that can be used to learn policies for a range of challenging continuous control tasks. In general, A does not need to be quadratic, and exploring other parametric forms such as multimodal distributions is an interesting avenue for future work. The appendix provides details on adaptive exploration rule derivation with experimental results, and a variational interpretation of Q-learning which gives an intuitive explanation of the behavior of NAF that conforms with empirical results.

5. Accelerating Learning with Imagination Rollouts

While NAF provides some advantages over actor-critic model-free RL methods in continuous domains, we can improve their data efficiency substantially under some additional assumptions by exploiting learned models. We will show that incorporating a particular type of learned model into Q-learning with NAFs significantly improves sample efficiency, while still allowing the final policy to be finetuned with model-free learning to achieve good performance without the limitations of imperfect models.

5.1. Model-Guided Exploration

One natural approach to incorporating a learned model into an off-policy algorithm such as Q-learning is to use the learned model to generate good exploratory behaviors using planning or trajectory optimization. To evalaute this idea, we utilize the iLQG algorithm to generate good trajectories under the model, and then mix these trajectories together with on-policy experience by appending them to the replay buffer. Interestingly, we show in our evaluation that, even when planning under the true model, the improvement obtained from this approach is often quite small, and varies significantly across domains and choices of exploration noise. The intuition behind this result is that off-policy iLQG exploration is too different from the learned policy, and Q-learning must consider alternatives in order to ascertain the optimality of a given action. That is, it's not enough to simply show the algorithm *good* actions, it must also experience bad actions to understand which actions are better and which are worse.

5.2. Imagination Rollouts

As discussed in the previous section, incorporating off-policy exploration from good, narrow distributions, such as those induced by iLQG, often does not result in significant improvement for Q-learning. These results suggest that Q-learning, which learns a policy based on minimizing temporal differences, inherently requires noisy on-policy actions to succeed. In real-world domains such as robots and autonomous vehicles, this can be undesirable for two reasons: first, it suggests that large amounts of on-policy experience are required in addition to good off-policy samples, and second, it implies that the policy must be allowed to make "its own mistakes" during training, which might involve taking undesirable or dangerous actions that can damage real-world hardware.

One way to avoid these problems while still allowing for a large amount of on-policy exploration is to generate synthetic on-policy trajectories under a learned model. Adding these synthetic samples, which we refer to as *imagination rollouts*, to the replay buffer effectively augments the amount of experience available for Q-learning. The particular approach we use is to perform rollouts in the real world using a mixture of planned iLQG trajectories and on-policy trajectories, with various mixing coefficients evaluated in our experiments, and then generate additional synthetic on-policy rollouts using the learned model from each state visited along the real-world rollouts. This approach can be viewed as a variant of the Dyna-Q algorithm (Sutton, 1990). However, while Dyna-Q has primarily been used with small and discrete systems, we show that using iteratively refitted linear models allows us to extend the approach to deep reinforcement learning on a range of continuous control domains. In some scenarios, we can even generate all or most of the real rollouts using off-policy iLQG controllers, which is desirable in safety-critic domains where poorly trained policies might take dangerous actions. The algorithm is given as Algorithm 2, and is an extension on Algorithm 1 combining model-based RL.

Algorithm 2 Imagination Rollouts with Fitted Dynamics and Optional iLQG Exploration

```
Randomly initialize normalized Q network $Q(\boldsymbol{x}, \boldsymbol{u}|\theta^Q)$.
Initialize target network $Q'$ with weight $\theta^{Q'} \leftarrow \theta^Q$.
Initialize replay buffer $R \leftarrow \emptyset$ and fictional buffer $R_f \leftarrow \emptyset$.
Initialize additional buffers $B \leftarrow \emptyset, B_{old} \leftarrow \emptyset$ with size $nT$.
Initialize fitted dynamics model $\mathcal{M} \leftarrow \emptyset$.
for $episode = 1, M$ do
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state $\boldsymbol{x}_1$
  Select $\mu'(\boldsymbol{x}, t)$ from $\{\mu(\boldsymbol{x}|\theta^\mu), \pi_t^{iLQG}(\boldsymbol{u}_t|\boldsymbol{x}_t)\}$ with probabilities $\{p, 1-p\}$
  for $t = 1, T$ do
    Select action $\boldsymbol{u}_t = \mu'(\boldsymbol{x}_t, t) + \mathcal{N}_t$
    Execute $\boldsymbol{u}_t$ and observe $r_t$ and $\boldsymbol{x}_{t+1}$
    Store transition $(\boldsymbol{x}_t, \boldsymbol{u}_t, r_t, \boldsymbol{x}_{t+1}, t)$ in $R$ and $B$
    if $\mod(episode \cdot T + t, m) = 0$ and $\mathcal{M} \neq \emptyset$ then
      Sample $m$ $(\boldsymbol{x}_i, \boldsymbol{u}_i, r_i, \boldsymbol{x}_{i+1}, i)$ from $B_{old}$
      Use $\mathcal{M}$ to simulate $l$ steps from each sample
      Store all fictional transitions in $R_f$
    end if
    Sample a random minibatch of $m$ transitions $I \cdot l$ times from $R_f$ and $I$ times from $R$, and update $\theta^Q, \theta^{Q'}$ as in Algorithm 1 per minibatch.
  end for
  if $B_f$ is full then
    $\mathcal{M} \leftarrow$ FitLocalLinearDynamics($B_f$) (see Section 5.3)
    $\pi^{iLQG} \leftarrow$ iLQG_OneStep($B_f, \mathcal{M}$) (see appendix)
    $B_{old} \leftarrow B_f, B_f \leftarrow \emptyset$
  end if
end for
```

Imagination rollouts can suffer from severe bias when the learned model is inaccurate. For example, we found it very difficult to train nonlinear neural network models for the dynamics that would actually improve the efficiency of Q-learning when used for imagination rollouts. As discussed in the following section, we found that using iteratively refitted time-varying linear dynamics produced substantially better results. In either case, we would still like to preserve the generality and optimality of model-free RL while deriving the benefits of model-based learning. To that end, we observe that most of the benefit of model-based learning is derived in the early stages of the learning process, when the policy induced by the neural network Q-function is poor. As the Q-function becomes more accurate, on-policy behavior tends to outperform model-based controllers. We therefore propose to switch off imagination rollouts after a given number of iterations.[1] In this framework, the imagination rollouts can be thought of as an inexpensive way to pretrain the Q-function, such that fine-tuning using real world experience can quickly converge to an optimal solution.

5.3. Fitting the Dynamics Model

In order to obtain good imagination rollouts and improve the efficiency of Q-learning, we needed to use an effective and data-efficient model learning algorithm. While prior methods propose a variety of model classes, including neural networks (Heess et al., 2015), Gaussian processes (Deisenroth & Rasmussen, 2011), and locally-weighted regression (Atkeson et al., 1997), we found that we could obtain good results by using iteratively refitted time-varying linear models, as proposed by Levine & Abbeel (2014). In this approach, instead of learning a good global model for all states and actions, we aim only to obtain a good local model around the latest set of samples. This approach requires a few additional assumptions: namely, it requires the initial state to be either deterministic or low-variance Gaussian, and it requires the states and actions to all be continuous. To handle domains with more varied initial states, we can use a mixture of Gaussian initial states with separate time-varying linear models for each one. The model itself is given by $p_t(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t) = \mathcal{N}(\mathbf{F}_t[\boldsymbol{x}_t; \boldsymbol{u}_t] + \mathbf{f}_t, \mathbf{N}_t)$. Every n episodes, we refit the parameters $\mathbf{F}_t$, $\mathbf{f}_t$, and $\mathbf{N}_t$ by fitting a Gaussian distribution at each time step to the vectors $[\boldsymbol{x}_t^i; \boldsymbol{u}_t^i; \boldsymbol{x}_{t+1}^i]$, where i indicates the sample index, and conditioning this Gaussian on $[\boldsymbol{x}_t; \boldsymbol{u}_t]$ to obtain the parameters of the linear-Gaussian dynamics at that step. We use $n = 5$ in our experiments. Although this approach introduces additional assumptions beyond the standard model-free RL setting, we show in our evaluation that it produces impressive gains in sample efficiency on tasks where it can be applied.

6. Experiments

We evaluated our approach on a set of simulated robotic tasks using the MuJoCo simulator (Todorov et al., 2012). The tasks were based on the benchmarks described by Lillicrap et al. (2016). Although we attempted to replicate the tasks in previous work as closely as possible, discrepancies in the simulator parameters and the contact model produced results that deviate slightly from those reported in prior work. In all experiments, the input to the policy consisted of the state of the system, defined in terms of joint angles and root link positions. Angles were often converted to sine and cosine encoding.

For both our method and the prior DDPG (Lillicrap et al., 2016) algorithm in the comparisons, we used neural networks with two layers of 200 rectified linear units (ReLU) to produce each of the output parameters – the Q-function and policy in DDPG, and the value function V, the advantage matrix $\boldsymbol{L}$, and the mean $\boldsymbol{\mu}$ for NAF. Since Q-learning was done with a replay buffer, we applied the Q-learning update 5 times per each step of experience to accelerate learning ($I = 5$). To ensure a fair comparison, DDPG also updates both the Q-function and policy parameters 5 times per step.

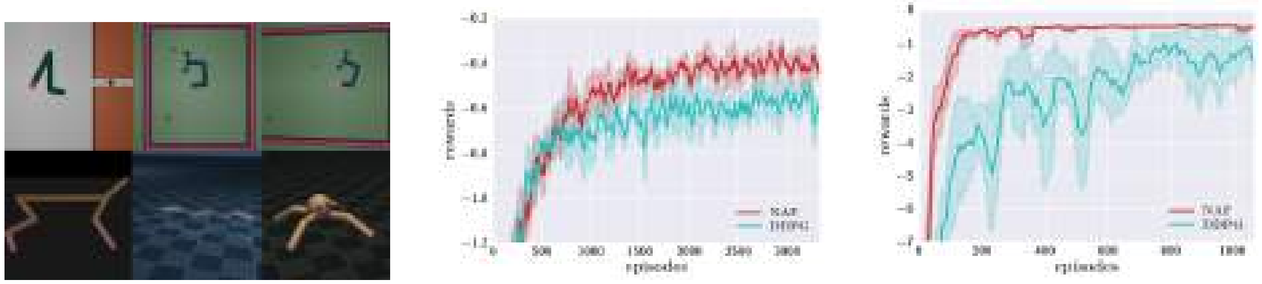
6.1. Normalized Advantage Functions

In this section, we compare NAF and DDPG on 10 representative domains from Lillicrap et al. (2016), with three additional domains: a four-legged 3D ant, a six-joint 2D swimmer, and a 2D peg (see the appendix for the descriptions of task domains). We found the most sensitive hyperparameters to be presence or absence of batch normalization, base learning rate for ADAM (Kingma & Ba, 2014) $\in \{1e^{-4}, 1e^{-3}, 1e^{-2}\}$, and exploration noise scale $\in \{0.1, 0.3, 1.0\}$. We report the best performance for each domain. We were unable to achieve good results with the method of Rawlik et al. (2013) on our domains, likely due to the complexity of high-dimensional neural network function approximators.

Figure 1b, Figure 1c, and additional figures in the appendix show the performances on the three-joint reacher, peg insertion, and a gripper with mobile base. While the numerical gap in reacher may be small, qualitatively there is also a very noticeable difference between NAF and DDPG. DDPG converges to a solution where the deterministic policy causes the tip to fluctuate continuously around the target, and does not reach it precisely. NAF, on the other hand, learns a smooth policy that makes the tip slow down and stabilize at the target. This difference is more noticeable in peg insertion and moving gripper, as shown by the much faster convergence rate to the optimal solution. Precision is very important in many real-world robotic tasks, and these result suggest that NAF may be preferred in such domains.

On locomotion tasks, the performance of the two methods is relatively similar. On the six-joint swimmer task and four-legged ant, NAF slightly outperforms DDPG in terms of the convergence speed; however, DDPG is faster on cheetah and finds a better policy on walker2d. The loss in performance of NAF can potentially be explained by downside of the mode-seeking behavior as analyzed in the appendix, where it is hard to explore other modes once the quadratic advantage function finds a good one. Choosing a parametric form that is more expressive than a quadratic

[1] In future work, it would be interesting to select this iteration adaptively based on the expected relative performance of the Q-function policy and model-based planning.



(a) Example task domains. (b) NAF and DDPG on multi-target reacher. (c) NAF and DDPG on peg insertion.

Figure 1. (a) Task domains: top row from left (manipulation tasks: peg, gripper, mobile gripper), bottom row from left (locomotion tasks: cheetah, swimmer6, ant). (b,c) NAF vs DDPG results on three-joint reacher and peg insertion. On reacher, the DDPG policy continuously fluctuates the tip around the target, while NAF stabilizes well at the target.

could be used to address this limitation in future work.

The results on all of the domains are summarized in Table 1. Overall, NAF outperformed DDPG on the majority of tasks, particularly manipulation tasks that require precision and suffer less from the lack of multimodal Q-functions. This makes this approach particularly promising for efficient learning of real-world robotic tasks.

Domains	-	DDPG	episodes	NAF	episodes
Cartpole	-2.1	-0.601	420	-0.604	**190**
Reacher	-2.3	-0.509	1370	**-0.331**	**1260**
Peg	-11	-0.950	690	**-0.438**	**130**
Gripper	-29	1.03	2420	**1.81**	**1920**
GripperM	-90	-20.2	1350	**-12.4**	**730**
Canada2d	-12	-4.64	1040	**-4.21**	900
Cheetah	-0.3	**8.23**	**1590**	7.91	2390
Swimmer6	-325	-174	220	**-172**	**190**
Ant	-4.8	-2.54	2450	-2.58	**1350**
Walker2d	0.3	**2.96**	**850**	1.85	1530

Table 1. Best test rewards of DDPG and NAF policies, and the episodes it requires to reach within 5% of the best value. "-" denotes scores by a random agent.

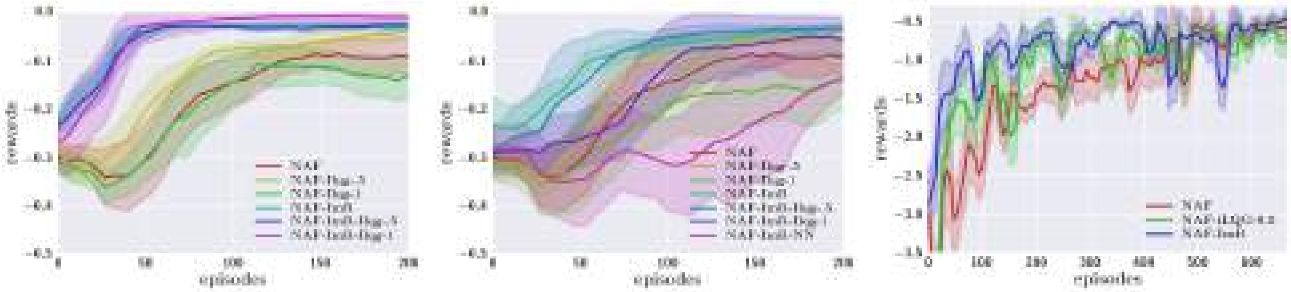
6.2. Evaluating Best-Case Model-Based Improvement with True Models

In order to determine how best to incorporate model-based components to accelerate model-free Q-learning, we tested several approaches using the ground truth dynamics, to control for challenges due to model fitting. We evaluated both of the methods discussed in Section 5: the use of model-based planning to generate good off-policy rollouts in the real world, and the use of the model to generate on-policy synthetic rollouts.

Figure 2a shows the effect of mixing off-policy iLQG experience and imagination rollouts on the three-joint reacher. It is noticeable that mixing the good off-policy experience does not significantly improve data-efficiency, while imagination rollouts always improve data-efficiency or final performance significantly. In the context of Q-learning, this result is not entirely surprising: Q learning must experience both good and bad actions in order to determine which actions are preferred, while the good model-based rollouts are so far removed from the policy in the early stages of learning that they provide little useful information. Figure 2a also evaluates two different variants of the imagination rollouts approach, where the rollouts in the real world are performed either using the learned policy, or using model-based planning with iLQG. In the case of this task, the iLQG rollouts achieve slightly better results, since the on-policy imagination rollouts sampled around these off-policy states provide Q-learning with additional information about alternative action not taken by the iLQG planner. In general, we did not find that off-policy rollouts were consistently better than on-policy rollouts across all tasks, but they did consistently produce good results. Performing off-policy rollouts with iLQG may be desirable in real-world domains, where a partially learned policy might take undesirable or dangerous actions. Further details of these experiments are provided in the appendix.

6.3. Guided Imagination Rollouts with Fitted Dynamics

In this section, we evaluated the performance of imagination rollouts with learned dynamics. As seen in Figure 2b, we found that fitting time-varying linear models following the imagination rollout algorithm is substantially better than fitting neural network dynamics models for the tasks we considered. There is a fundamental tension between efficient learning and expressive models like neural nets. We cannot hope to learn useful neural network models with a small number of samples for complex tasks, which makes it difficult to acquire a good model with fewer samples than are necessary to acquire a good policy. While the model is trained with supervised learning, which is typically more sample efficient, it often needs to represent a more complex function (e.g. rigid body physics). However, having such expressive models is more crucial as we move to improve model accuracy. Figure 2b presents results that compare



(a) NAF on single-target reacher.

(b) NAF on single-target reacher.

(c) NAF on single-target gripper.

Figure 2. Results on NAF with iLQG-guided exploration and imagination rollouts (a) using true dynamics (b,c) using fitted dynamics. “ImR” denotes using the imagination rollout with $l = 10$ steps on the reacher and $l = 5$ steps on the gripper. “iLQG-x” indicates mixing x fraction of iLQG episodes. Fitted dynamics uses time-varying linear models with sample size $n = 5$, except “-NN” which fits a neural network to global dynamics.

fitted neural network models with the true dynamics when combined with imagination rollouts. These results indicate that the learned neural network models negate the benefits of imagination rollouts on our domains.

To evaluate imagination rollouts with fitted time-varying linear dynamics, we chose single-target variants of two of the manipulation tasks: the reacher and the gripper task. The results are shown in Figure 2b and 2c. We found that imagination rollouts of length 5 to 10 were sufficient for these tasks to achieve significant improvement over the fully model-free variant of NAF.

Adding imagination rollouts in these domains provided 2-5 factors of improvement in data efficiency. In order to retain the benefit of model-free learning and allow the policy to continue improving once it exceeds the quality possible under the learned model, we switch off the imagination rollouts after 130 episodes (20,000 steps) on the gripper domain. This produces a small transient drop in the performance of the policy, but the results quickly improve again. Switching off the imagination rollouts also ensures that Q-learning does not diverge after it reaches good values, as were often observed in the gripper. This suggests that imagination rollouts, in contrast to off-policy exploration discussed in the previous section, is an effective method for bootstrapping model-free deep RL.

It should be noted that, although time-varying linear models combined with imagination rollouts provide a substantial boost in sample efficiency, this improvement is provided at some cost in generality, since effective fitting of time-varying linear models requires relatively small initial state distributions. With more complex initial state distributions, we might cluster the trajectories and fit multiple models to account for different modes. Extending the benefits of time-varying linear models to less restrictive settings is a promising direction and build on prior work (Levine et al., 2015; Fu et al., 2015). That said, our results show that imagination rollouts are a very promising approach to accelerating model-free learning when combined with the right kind of dynamics model.

7. Discussion

In this paper, we explored several methods for improving the sample efficiency of model-free deep reinforcement learning. We first propose a method for applying standard Q-learning methods to high-dimensional, continuous domains, using the normalized advantage function (NAF) representation. This allows us to simplify the more standard actor-critic style algorithms, while preserving the benefits of nonlinear value function approximation, and allows us to employ a simple and effective adaptive exploration method. We show that, in comparison to recently proposed deep actor-critic algorithms, our method tends to learn faster and acquires more accurate policies. We further explore how model-free RL can be accelerated by incorporating learned models, without sacrificing the optimality of the policy in the face of imperfect model learning. We show that, although Q-learning can incorporate off-policy experience, learning primarily from off-policy exploration (via model-based planning) only rarely improves the overall sample efficiency of the algorithm. We postulate that this caused by the need to observe both successful and unsuccessful actions, in order to obtain an accurate estimate of the Q-function. We demonstrate that an alternative method based on synthetic on-policy rollouts achieves substantially improved sample complexity, but only when the model learning algorithm is chosen carefully. We demonstrate that training neural network models does not provide substantive improvement in our domains, but simple iteratively refitted time-varying linear models do provide substantial improvement on domains where they can be applied.

Acknowledgement

We thank Nicholas Heess for helpful discussion and Tom Erez, Yuval Tassa, Vincent Vanhoucke, and the Google Brain and DeepMind teams for their support.

References

Atkeson, Christopher G, Moore, Andrew W, and Schaal, Stefan. Locally weighted learning for control. In *Lazy learning*, pp. 75–113. Springer, 1997.

Baird III, Leemon C. Advantage updating. Technical report, DTIC Document, 1993.

de Bruin, Tim, Kober, Jens, Tuyls, Karl, and Babuška, Robert. The importance of experience replay database composition in deep reinforcement learning. *Deep Reinforcement Learning Workshop, NIPS*, 2015.

Deisenroth, Marc and Rasmussen, Carl E. Pilco: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning (ICML)*, pp. 465–472, 2011.

Deisenroth, Marc Peter, Neumann, Gerhard, Peters, Jan, et al. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142, 2013.

Fu, Justin, Levine, Sergey, and Abbeel, Pieter. One-shot learning of manipulation skills with online dynamics adaptation and neural network priors. *arXiv preprint arXiv:1509.06841*, 2015.

Hafner, Roland and Riedmiller, Martin. Reinforcement learning in feedback control. *Machine learning*, 84(1-2):137–169, 2011.

Harmon, Mance E and Baird III, Leemon C. Multi-player residual advantage learning with general function approximation. *Wright Laboratory, WL/AACF, Wright-Patterson Air Force Base, OH*, pp. 45433–7308, 1996.

Hausknecht, Matthew and Stone, Peter. Deep reinforcement learning in parameterized action space. *arXiv preprint arXiv:1511.04143*, 2015.

Heess, Nicolas, Wayne, Gregory, Silver, David, Lillicrap, Tim, Erez, Tom, and Tassa, Yuval. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 2926–2934, 2015.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kober, Jens and Peters, Jan. Reinforcement learning in robotics: A survey. In *Reinforcement Learning*, pp. 579–610. Springer, 2012.

Konda, Vijay R and Tsitsiklis, John N. Actor-critic algorithms. In *Advances in Neural Information Processing Systems (NIPS)*, volume 13, pp. 1008–1014, 1999.

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

Levine, Sergey and Koltun, Vladlen. Guided policy search. In *International Conference on Machine Learning (ICML)*, pp. 1–9, 2013.

Levine, Sergey, Finn, Chelsea, Darrell, Trevor, and Abbeel, Pieter. End-to-end training of deep visuomotor policies. *arXiv preprint arXiv:1504.00702*, 2015.

Li, Weiwei and Todorov, Emanuel. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *ICINCO (1)*, pp. 222–229, 2004.

Lillicrap, Timothy P, Hunt, Jonathan J, Pritzel, Alexander, Heess, Nicolas, Erez, Tom, Tassa, Yuval, Silver, David, and Wierstra, Daan. Continuous control with deep reinforcement learning. *International Conference on Learning Representations (ICLR)*, 2016.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Peters, Jan and Schaal, Stefan. Policy gradient methods for robotics. In *International Conference on Intelligent Robots and Systems (IROS)*, pp. 2219–2225. IEEE, 2006.

Peters, Jan, Mülling, Katharina, and Altun, Yasemin. Relative entropy policy search. In *AAAI*. Atlanta, 2010.

Rawlik, Konrad, Toussaint, Marc, and Vijayakumar, Sethu. On stochastic optimal control and reinforcement learning by approximate inference. *Robotics*, pp. 353, 2013.

Schaul, Tom, Quan, John, Antonoglou, Ioannis, and Silver, David. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Schulman, John, Levine, Sergey, Abbeel, Pieter, Jordan, Michael I., and Moritz, Philipp. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, pp. 1889–1897, 2015.

Schulman, John, Moritz, Philipp, Levine, Sergey, Jordan, Michael, and Abbeel, Pieter. High-dimensional continuous control using generalized advantage estimation. *International Conference on Learning Representations (ICLR)*, 2016.

Silver, David, Lever, Guy, Heess, Nicolas, Degris, Thomas, Wierstra, Daan, and Riedmiller, Martin. Deterministic policy gradient algorithms. In *International Conference on Machine Learning (ICML)*, 2014.

Sutton, Richard S. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *International Conference on Machine Learning (ICML)*, pp. 216–224, 1990.

Sutton, Richard S, McAllester, David A, Singh, Satinder P, Mansour, Yishay, et al. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NIPS)*, volume 99, pp. 1057–1063, 1999.

Tassa, Yuval, Erez, Tom, and Todorov, Emanuel. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *International Conference on Intelligent Robots and Systems (IROS)*, pp. 4906–4913. IEEE, 2012.

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

Wahlström, Niklas, Schön, Thomas B, and Deisenroth, Marc Peter. From pixels to torques: Policy learning with deep dynamical models. *arXiv preprint arXiv:1502.02251*, 2015.

Wang, Ziyu, de Freitas, Nando, and Lanctot, Marc. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

Watter, Manuel, Springenberg, Jost, Boedecker, Joschka, and Riedmiller, Martin. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 2728–2736, 2015.

8. Appendix

8.1. iLQG

The iLQG algorithm optimizes trajectories by iteratively constructing locally optimal linear feedback controllers under a local linearization of the dynamics $p(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t) = \mathcal{N}(\boldsymbol{f}_{\boldsymbol{x}t}\boldsymbol{x}_t + \boldsymbol{f}_{\boldsymbol{u}t}\boldsymbol{u}_t, \boldsymbol{F}_t)$ and a quadratic expansion of the rewards $r(\boldsymbol{x}_t, \boldsymbol{u}_t)$ (Tassa et al., 2012). Under linear dynamics and quadratic rewards, the action-value function $Q(\boldsymbol{x}_t, \boldsymbol{u}_t)$ and value function $V(\boldsymbol{x}_t)$ are locally quadratic and can be computed by dynamics programming.[2]

$$
\begin{aligned}
Q_{\boldsymbol{xu},\boldsymbol{xu}t} &= r_{\boldsymbol{xu},\boldsymbol{xu}t} + \boldsymbol{f}_{\boldsymbol{xu}t}^T V_{\boldsymbol{x},\boldsymbol{x}t+1} \boldsymbol{f}_{\boldsymbol{xu}t} \\
Q_{\boldsymbol{xu}t} &= r_{\boldsymbol{xu}t} + \boldsymbol{f}_{\boldsymbol{xu}t}^T V_{\boldsymbol{x},\boldsymbol{x}t+1} \\
V_{\boldsymbol{x},\boldsymbol{x}t} &= Q_{\boldsymbol{x},\boldsymbol{x}t} - Q_{\boldsymbol{u},\boldsymbol{x}t}^T Q_{\boldsymbol{u},\boldsymbol{u}t}^{-1} Q_{\boldsymbol{u},\boldsymbol{x}t} \\
V_{\boldsymbol{x}t} &= Q_{\boldsymbol{x}t} - Q_{\boldsymbol{u},\boldsymbol{x}t}^T Q_{\boldsymbol{u},\boldsymbol{u}t}^{-1} Q_{\boldsymbol{u}t} \\
Q_{\boldsymbol{x},\boldsymbol{x}T} &= V_{\boldsymbol{x},\boldsymbol{x}T} = r_{\boldsymbol{x},\boldsymbol{x}T}
\end{aligned}
\tag{5}
$$

The time-varying linear feedback controller $\boldsymbol{g}(\boldsymbol{x}_t) = \hat{\boldsymbol{u}}_t + \boldsymbol{k}_t + \boldsymbol{K}_t(\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t)$ maximizes the locally quadratic Q, where $\boldsymbol{k}_t = -Q_{\boldsymbol{u},\boldsymbol{u}t}^{-1} Q_{\boldsymbol{u}t}$, $\boldsymbol{K}_t = -Q_{\boldsymbol{u},\boldsymbol{u}t}^{-1} Q_{\boldsymbol{u},\boldsymbol{x}t}$, and $\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{u}}_t$ denote states and actions of the current trajectory around which the partial derivatives are computed. Employing the maximum entropy objective (Levine & Koltun, 2013), we can also construct a linear-Gaussian controller, where c is a scalar to adjust for arbitrary scaling of the reward magnitudes,

$$
\pi_t^{iLQG}(\boldsymbol{u}_t|\boldsymbol{x}_t) = \mathcal{N}(\hat{\boldsymbol{u}}_t + \boldsymbol{k}_t + \boldsymbol{K}_t(\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t), -cQ_{\boldsymbol{u},\boldsymbol{u}t}^{-1})
\tag{6}
$$

When the dynamics are not known, a particularly effective way to use iLQG is to combine it with learned time-varying linear models $\hat{p}(\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{u}_t)$. In this variant of the algorithm, trajectories are sampled from the controller in Equation (6) and used to fit time-varying linear dynamics with linear regression. These dynamics are then used with iLQG to obtain a new controller, typically using a KL-divergence constraint to enforce a trust region, so that the new controller doesn't deviate too much from the region in which the samples were generated (Levine & Abbeel, 2014).

8.2. Locally-Invariant Exploration for Normalized Advantage Functions

Exploration is an essential component of reinforcement learning algorithms. The simplest and most common type of exploration involves randomizing the actions according to some distribution, either by taking random actions with some probability (Mnih et al., 2015), or adding Gaussian noise in continuous action spaces (Schulman et al., 2015). However, choosing the magnitude of the random exploration noise can be difficult, particularly in high-dimensional domains where different action dimensions require very different exploration scales. Furthermore, independent (spherical) Gaussian noise may be inappropriate for tasks where the optimal behavior requires correlation between action dimensions, as for example in the case of the swimming snake described in our experiments, which must coordinate the motion of different body joints to produce a synchronized undulating gait.

The NAF provides us with a simple and natural avenue to obtain an adaptive exploration strategy, analogously to Boltzmann exploration. The idea is to use the matrix in the quadratic component of the advantage function as the precision for a Gaussian action distribution. This naturally causes the policy to become more deterministic along directions where the advantage function varies steeply, and more random along directions where it is flat. The corresponding policy is given by

$$
\begin{aligned}
\pi(\boldsymbol{u}|\boldsymbol{x}) &= \exp^{Q(\boldsymbol{x},\boldsymbol{u}|\theta^Q)} / \int \exp^{Q(\boldsymbol{x},\boldsymbol{u}|\theta^Q)} d\boldsymbol{u} \\
&= \mathcal{N}(\boldsymbol{\mu}(\boldsymbol{x}|\theta^\mu), c\boldsymbol{P}(\boldsymbol{x}|\theta^P)^{-1}).
\end{aligned}
\tag{7}
$$

Previous work also noted that generating Gaussian exploration noise independently for each time step was not well-suited for many continuous control tasks, particularly simulated robotic tasks where the actions correspond to torques or velocities (Lillicrap et al., 2016). The intuition is that, as the length of the time-step decreases, temporally independent Gaussian exploration will cancel out between time steps. Instead, prior work proposed to sample noise from an Ornstein-Uhlenbech (OU) process to generate a temporally correlated noise sequence (Lillicrap et al., 2016). We adopt the same approach in our work, but sample the innovations for the OU process from the Gaussian distribution in Equation 7. Lastly, we note that the overall scale of $\boldsymbol{P}(\boldsymbol{x}|\theta^P)$ could vary significantly through the learning, and depends on the magnitude of the cost, which introduces an undesirable additional degree of freedom. We therefore use a heuristic adaptive-scaling trick to stabilize the noise magnitudes.

Using the learned precision as the noise covariance for exploration allowed for convergence to a better policy on the "canada2d" task, which requires using an arm to strike a puck toward a target, as shown in Figure 3, but did not make a significant difference on the other domains.

8.3. Q-Learning as Variational Inference

NAF, with our choice of parametrization, can only fit a locally quadratic Q-function. To understand its implications and expected behaviors, one approach is to approximately view the Q-learning objective as minimizing the exclusive

[2] While standard iLQG notation denotes Q, V as discounted sum of costs, we denote them as sum of rewards to make them consistent with the rest of the paper

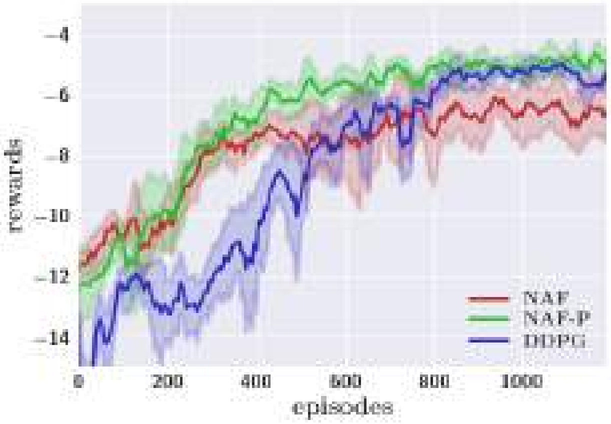


Figure 3. NAF with exploration noise generated using the precision term (NAF-P) slightly outperforms the best DDPG result. Precision term is not used until step 50,000.

Kullback-Leibler divergence (KL) between policy distributions induced by the fitted normalized Q-function $\hat{Q}$ and the true Q-function Q. This can be derived by assuming (1) no bootstrapping and the exact target Q is provided per step, (2) no additive exploration noise, i.e. fully on-policy, and (3) use KL loss instead of the least-square. Specifically, let π and $\hat{\pi}$ be corresponding policies of Q and $\hat{Q}$ respectively, an alternative form of Q-learning could be optimizing the following objective:

$$L_e(\hat{Q}) = \mathbb{E}_{\boldsymbol{x}_t \sim \rho^{\hat{\pi}}}[\tilde{KL}(\hat{\pi}||\pi)] = \mathbb{E}_{\boldsymbol{x}_t \sim \rho^{\hat{\pi}}, \boldsymbol{u}_t \sim \hat{\pi}}[\hat{Q} - Q] \tag{8}$$

We can thus intuitively interpret NAF as doing variational inference to fit a Gaussian to a distribution, and it has mode-seeking behavior. Empirically such behavior enables NAF to learn smoother and more precise controllers, as most effectively illustrated by three-joint reacher and peg insertion experiments, and substantial improvements in terms of convergence speeds in many other representative domains explored in the main paper.

8.4. Descriptions of Task Domains

Table 3 describes the task domains used in the experiments.

8.5. More Results on Normalized Advantage Functions

Figures 4a, 4b, and 4c provide additional results on the comparison experiments between DDPG and NAF. As shown in the main paper, NAF generally outperforms DDPG. In certain tasks that require precision, such as peg insertion, the difference is very noticeable. However, there are also few cases where NAF underperforms DDPG. The most consistent of such cases is cheetah. While both DDPG and NAF enable cheetah to run decent distances, it is often observed that the cheetah movements learned in NAF are little less natural than those from DDPG. We speculate such behaviors come from the mode-seeking behavior that is explained in Section 8.3, and thus exploring other parametric forms of NAF, such as multi-modal variants, is a promising avenue for future work.

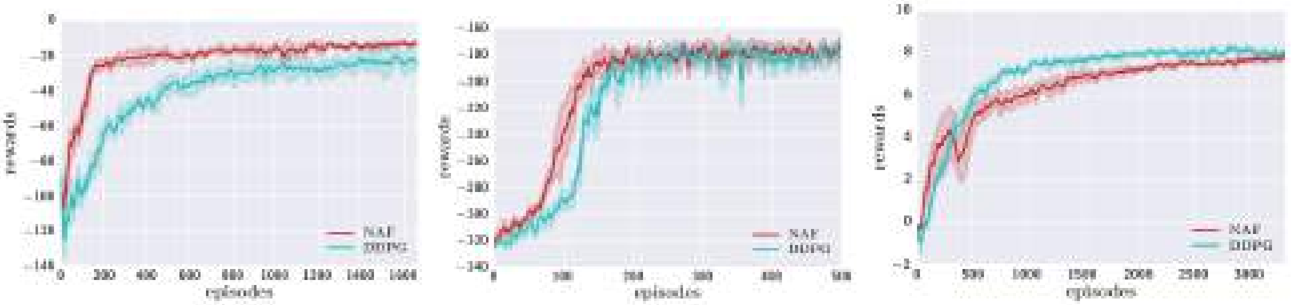
8.6. More Results on Evaluating Best-Case Model-Based Improvement with True Models

Domains	-	0.5	ImR	ImR,0.5	ImR,1
Reacher	-0.488	-0.449	-0.448	**-0.426**	-0.548
episodes	740	670	450	430	**90**
Canada2d	-6.23	-6.23	-5.89	**-5.88**	-12.0
episodes	1970	1580	570	**140**	210
Cheetah	7.00	7.10	**7.36**	7.29	6.43
episodes	580	1080	590	740	**390**

Table 2. Best-case model-based acceleration with true dynamics models. Best test rewards of NAF policies (first row), and the episodes it required to reach 5% of the best value (second row). “0.5” and “1” correspond to the fraction of MPC episodes. “ImR” means using imagination rollout with rollout length $l = 10$ for reacher, canada2d, and $l = 5$ for cheetah.

In the main paper, iLQG with true dynamics is used to generate guided exploration trajectories. While iLQG works for simple manipulation tasks with small number of initial states, it does not work well for random target reacher or complex locomotion tasks such as cheetah. We therefore run iLQG in model-predictive control (MPC) mode for the experiments reported in Figures 5c, 5b, and 5a, and Table 2. It is important to note that for those experiments, the hyperparameters were fixed (batch normalization is on, learning rate is 10^{-3}, and exploration size is 0.3), and thus the results differ slightly from the experiments in the previous section.

In cheetah and other complex locomotion tasks, MPC policy is usually sub-optimal, and thus poor performance of mixing MPC experience in Figure 5b is expected. On the other hand, MPC policy works reasonably in hard manipulation tasks such as canada2d, and there is significant gain from mixing MPC experience as Figure 5c shows. However, the most consistent gain comes from using imagination rollouts in all three domains. In particular, Figure 5c shows that in canada2d, MPC experiences gives very good trajectories, i.e. those that hit balls in roughly the right directions, and doing rollouts can generate more of this useful experience, enabling canada2d to learn very quickly. While with true dynamics having the imagination experience directly means more experience and such result may be trivial, it is still important to see the benefits of rollouts which only explore up to $l = 10$ steps away from the real experience, as reported here. This is an interesting result, since this means the dynamics model only needs to be accurate around the data trajectories and this significantly lessens the requirement on fitted models.

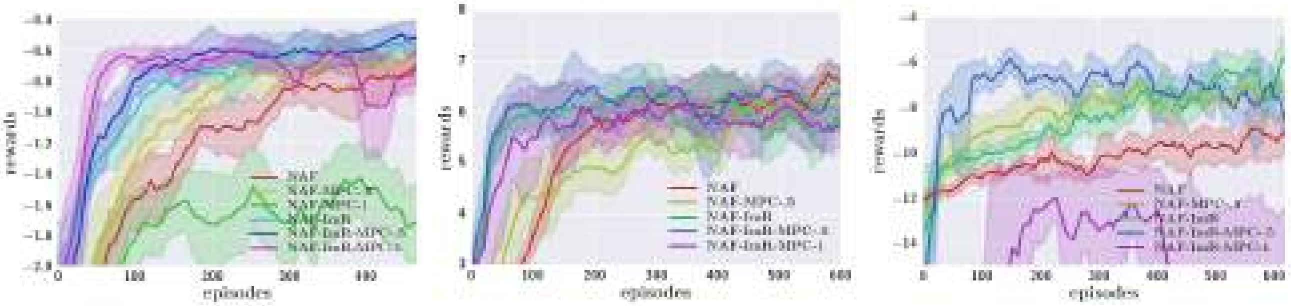


(a) NAF significantly outperforms DDPG on moving gripper.

(b) NAF converges faster than DDPG on swimmer6.

(c) DDPG converges faster than NAF on cheetah.

Figure 4. NAF vs DDPG on three domains.



(a) NAF on multi-target reacher. Insignificant gain from mixing MPC experience. Significant gain from imagination rollouts.

(b) NAF on cheetah. Great speeds up with imagination rollouts, no gain from mixing MPC experiences.

(c) NAF on canada2d. Very significant speed-ups from mixing MPC experiences, both with or without the rollouts.

Figure 5. NAF on multi-target reacher, cheetah, and canada2d, with model-based acceleration using true dynamics: "ImR" denotes using the imagination rollout, $l = 10$ steps. "MPC-x" indicates mixing x fraction of MPC episodes.

Domain	Description	Domain	Description
Cartpole	The classic cart-pole swing-up task. Agent must balance a pole attached to a cart by applying forces to the cart alone. The pole starts each episode hanging upside-down.	Reacher	Agent is required to move a 3-DOF arm from random starting locations to random target positions.
Peg	Agent is required to insert the tip of a 3-DOF arm from locally-perturbed starting locations to a fixed hole.	Gripper	Agent must use an arm with gripper appendage to grasp an object and manuver the object to a fixed target.
GripperM	Agent must use an arm with gripper attached to a moveable platform to grasp an object and move it to a fixed target.	Canada2d	Agent is required to use an arm with hockey-stick like appendage to hit a ball initialzed to a random start location to a random target location.
Cheetah	Agent should move forward as quickly as possible with a cheetah- like body that is constrained to the plane.	Swimmer6	Agent should swim in snake-like manner toward the fixed target using six joints, starting from random poses.
Ant	The four-legged ant should move toward the fixed target from a fixed starting position and posture.	Walker2d	Agent should move forward as quickly as possible with a bipedal walker constrained to the plane without falling down or pitching the torso too far forward or backward.

Table 3. List of domains. All the domains except ant are 2D.