2018 IEEE International Conference on Robotics and Automation (ICRA)
May 21-25, 2018, Brisbane, Australia

Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning

Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, Sergey Levine
University of California, Berkeley

***Abstract*— Model-free deep reinforcement learning algorithms have been shown to be capable of learning a wide range of robotic skills, but typically require a very large number of samples to achieve good performance. Model-based algorithms, in principle, can provide for much more efficient learning, but have proven difficult to extend to expressive, high-capacity models such as deep neural networks. In this work, we demonstrate that neural network dynamics models can in fact be combined with model predictive control (MPC) to achieve excellent sample complexity in a model-based reinforcement learning algorithm, producing stable and plausible gaits that accomplish various complex locomotion tasks. We further propose using deep neural network dynamics models to initialize a model-free learner, in order to combine the sample efficiency of model-based approaches with the high task-specific performance of model-free methods. We empirically demonstrate on MuJoCo locomotion tasks that our pure model-based approach trained on just random action data can follow arbitrary trajectories with excellent sample efficiency, and that our hybrid algorithm can accelerate model-free learning on high-speed benchmark tasks, achieving sample efficiency gains of $3-5\times$ on swimmer, cheetah, hopper, and ant agents. Videos can be found at `https://sites.google.com/view/mbmf`**

I. Introduction

Model-free deep reinforcement learning algorithms have been shown to be capable of learning a wide range of tasks, ranging from playing video games from images [1], [2] to learning complex locomotion skills [3]. However, such methods suffer from very high sample complexity, often requiring millions of samples to achieve good performance [3]. Model-based reinforcement learning algorithms are generally regarded as being more efficient [4]. However, to achieve good sample efficiency, these model-based algorithms have conventionally used either simple function approximators [5] or Bayesian models that resist overfitting [6] in order to effectively learn the dynamics using few samples. This makes them difficult to apply to a wide range of complex, high-dimensional tasks. Although a number of prior works have attempted to mitigate these shortcomings by using large, expressive neural networks to model the complex dynamical systems typically used in deep reinforcement learning benchmarks [7], [8], such models often do not perform well [9] and have been limited to relatively simple, low-dimensional tasks [10].

In this work, we demonstrate that multi-layer neural network models can in fact achieve excellent sample complexity in a model-based reinforcement learning algorithm. The resulting models can then be used for model-based control, which we perform using model predictive control (MPC) with a simple random-sampling shooting method [11]. We demonstrate that this method can acquire effective locomotion gaits for a variety of MuJoCo benchmark systems [8], including the swimmer, half-cheetah, hopper, and ant. Fig. 1 shows that these models can be used at run-time to execute a variety of locomotion tasks such as trajectory following, where the agent executes a path through a given set of sparse waypoints that represent desired center-of-mass positions. Additionally, each systems uses less than four hours worth of data, indicating that the sample complexity of our model-based approach is low enough to be applied in the real world, and is dramatically lower than pure model-free learners. In particular, when comparing our model-based approach's ability to follow arbitrary desired trajectories with a model-free approach's ability to learn just a competent moving forward gait, our results show that the model-based method uses only 3%, 10%, and 14% of the data that is used by a model-free approach (for half-cheetah, swimmer, and ant, respectively). Relatedly, our model-based method can achieve qualitatively good moving forward gaits for the swimmer, cheetah, hopper, and ant using $20-80\times$ fewer data points than is required by a model-free approach.

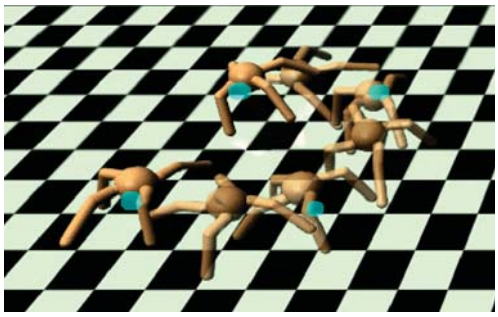


Fig. 1: Our method can learn a dynamics model that enables a simulated quadrupedal robot to autonomously follow user-defined waypoints. Training for this task used 7e5 time steps (collected without any knowledge of the test-time navigation task), and the learned model can be reused at test time to follow arbitrary desired trajectories.

Although such model-based methods are drastically more sample efficient and more flexible than task-specific policies learned with model-free reinforcement learning, their asymptotic performance is usually worse than model-free learners due to model bias. Model-free algorithms are not limited by the accuracy of the model, and therefore can achieve better final performance, though at the expense of

much higher sample complexity [4], [12]. To address this issue, we use our model-based algorithm, which can quickly achieve moderately proficient behavior, to initialize a model-free learner, which can slowly achieve near-optimal behavior. The learned model-based controller provides good rollouts, which enable supervised initialization of a policy that can then be fine-tuned with model-free algorithms, such as policy gradients. We empirically demonstrate that the resulting hybrid model-based and model-free (Mb-Mf) algorithm can accelerate model-free learning, achieving sample efficiency gains of $3-5\times$ on the swimmer, half-cheetah, hopper, and ant.

The primary contributions of our work are the following: (1) we demonstrate effective model-based reinforcement learning with neural network models for several contact-rich simulated locomotion tasks from standard deep reinforcement learning benchmarks, (2) we empirically evaluate a number of design decisions for neural network dynamics model learning, and (3) we show how a model-based learner can be used to initialize a model-free learner to achieve high rewards while significantly reducing sample complexity.

II. Related Work

Deep reinforcement learning algorithms based on Q-learning [13], [2], [9], actor-critic methods [14], [15], [16], and policy gradients [3], [17] have been shown to learn very complex skills in high-dimensional state spaces, including simulated robotic locomotion, driving, video game playing, and navigation. However, the high sample complexity of purely model-free algorithms has made them difficult to use for learning in the real world, where sample collection is limited by the constraints of real-time operation. Model-based algorithms are known in general to outperform model-free learners in terms of sample complexity [4], and in practice have been applied successfully to control both simulated and real-world robotic systems, such as pendulums [6], legged robots [18], swimmers [19], and manipulators [20]. However, the most efficient model-based algorithms have used relatively simple function approximators, such as Gaussian processes [6], [21], [22], time-varying linear models [5], [23], [24], and mixtures of Gaussians [25]. PILCO [6], in particular, is a model-based policy search method which reports excellent sample efficiency by learning probabilistic dynamics models and incorporating model uncertainty into long-term planning. These methods have difficulties, however, in high-dimensional spaces and with nonlinear dynamics. The most high-dimensional task demonstrated with PILCO that we could find has 11 dimensions [19], while the most complex task in our work has 49 dimensions and features challenging properties such as frictional contacts. To the best of our knowledge, no prior model-based method utilizing Gaussian processes has demonstrated successful learning for locomotion tasks with complex contact physics, though several works have proposed to learn the dynamics [26].

Although neural networks have been widely used in earlier work to model plant dynamics [27], [28], more recent model-based algorithms have achieved only limited success in applying such models to the more complex benchmark tasks that are commonly used in deep reinforcement learning. Several works have proposed to use deep neural network models for building predictive models of images [29], but these methods have either required extremely large datasets for training [29] or were applied to short-horizon control tasks [30]. In contrast, we consider long-horizon simulated locomotion tasks, where the high-dimensional systems and contact-rich environment dynamics provide a considerable modeling challenge. [10] proposed a relatively complex time-convolutional model for dynamics prediction, but only demonstrated results on low-dimensional (2D) manipulation tasks. [31] extended PILCO [6] using Bayesian neural networks, but only presented results on a low-dimensional cart-pole swingup task, which does not include contacts.

Aside from training neural network dynamics models for model-based reinforcement learning, we also explore how such models can be used to accelerate a model-free learner. Prior work on model-based acceleration has explored a variety of avenues. The classic Dyna [32] algorithm proposed to use a model to generate simulated experience that could be included in a model-free algorithm. This method was extended [33], [34] to work with deep neural network policies, but performed best with models that were not neural networks [9]. Model learning has also been used to accelerate model-free Bellman backups [35], but the gains in performance from including the model were relatively modest. Prior work has also used model-based learners to guide policy optimization through supervised learning [36], but the models that were used were typically local linear models. In a similar way, we also use supervised learning to initialize the policy, but we then fine-tune this policy with model-free learning to achieve the highest returns. Our model-based method is more flexible than local linear models, and it does not require multiple samples from the same initial state for local linearization.

III. Preliminaries

The goal of reinforcement learning is to learn a policy that maximizes the sum of future rewards. At each time step t, the agent is in state $\mathbf{s}_t \in \mathcal{S}$, executes some action $\mathbf{a}_t \in \mathcal{A}$, receives reward $r_t = r(\mathbf{s}_t, \mathbf{a}_t)$, and transitions to the next state $\mathbf{s}_{t+1}$ according to some unknown dynamics function $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$. The goal at each time step is to take the action that maximizes the discounted sum of future rewards, given by $\sum_{t'=t}^{\infty} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$, where $\gamma \in [0, 1]$ is a discount factor that prioritizes near-term rewards. Note that performing this policy extraction requires either knowing the underlying reward function $r(\mathbf{s}_t, \mathbf{a}_t)$ or estimating the reward function from samples [37]. In this work, we assume access to the underlying reward function, which we use for planning actions under the learned model.

In model-based reinforcement learning, a model of the dynamics is used to make predictions, which are then used for action selection. Let $\hat{f}_\theta(\mathbf{s}_t, \mathbf{a}_t)$ denote a learned discrete-time dynamics function, parameterized by θ, that takes the current state $\mathbf{s}_t$ and action $\mathbf{a}_t$ and outputs an estimate of the next state at time $t + \Delta t$. This model can then be used to

predict the outcomes of various action sequences, and then actions can be selected by choosing the sequence that results in the highest predicted total reward. In practice, it is often desirable to solve this optimization at each time step, execute only the first action $\mathbf{a}_t$ from the sequence, and then replan at the next time step with updated state information. Such a control scheme is often referred to as model predictive control (MPC), and it is known to compensate well for errors in the model.

IV. Model-Based Deep Reinforcement Learning

We now present our model-based deep reinforcement learning algorithm. We detail our learned dynamics function $\hat{f}_\theta(\mathbf{s}_t, \mathbf{a}_t)$ in Sec. IV-A, how to train the learned dynamics function in Sec. IV-B, how to extract a policy using our learned dynamics function in Sec. IV-C, and how to use reinforcement learning to further improve our learned dynamics function in Sec. IV-D.

A. Neural Network Dynamics Function

We parameterize our learned dynamics function $\hat{f}_\theta(\mathbf{s}_t, \mathbf{a}_t)$ as a deep neural network, where the parameter vector θ represents the weights of the network. A straightforward parameterization for $\hat{f}_\theta(\mathbf{s}_t, \mathbf{a}_t)$ would take as input the current state $\mathbf{s}_t$ and action $\mathbf{a}_t$, and output the predicted next state $\hat{\mathbf{s}}_{t+1}$. However, this function can be difficult to learn when the states $\mathbf{s}_t$ and $\mathbf{s}_{t+1}$ are too similar and the action has seemingly little effect on the output; this difficulty becomes more pronounced as the time between states Δt becomes smaller and the state differences do not indicate the underlying dynamics well.

We overcome this issue by instead learning a dynamics function that predicts the change in state $\mathbf{s}_t$ over the time step duration of Δt. Thus, the predicted next state is as follows: $\hat{\mathbf{s}}_{t+1} = \mathbf{s}_t + \hat{f}_\theta(\mathbf{s}_t, \mathbf{a}_t)$. Note that increasing this Δt increases the information available from each data point, and can help with not only dynamics learning but also with planning using the learned dynamics model (Sec. IV-C). However, increasing Δt also increases the discretization and complexity of the underlying continuous-time dynamics, which can make the learning process more difficult.

B. Training the Learned Dynamics Function

Collecting training data: We collect training data by sampling starting configurations $\mathbf{s}_0 \sim p(\mathbf{s}_0)$, executing random actions at each timestep, and recording the resulting trajectories $\tau = (\mathbf{s}_0, \mathbf{a}_0, \cdots, \mathbf{s}_{T-2}, \mathbf{a}_{T-2}, \mathbf{s}_{T-1})$ of length T. We note that these trajectories are very different from the trajectories the agents will end up executing when planning with this learned dynamics model and a given reward function $r(\mathbf{s}_t, \mathbf{a}_t)$ (Sec. IV-C), showing the ability of model-based methods to learn from off-policy data.

Data preprocessing: We slice the trajectories $\{\tau\}$ into training data inputs $(\mathbf{s}_t, \mathbf{a}_t)$ and corresponding output labels $\mathbf{s}_{t+1} - \mathbf{s}_t$. We then subtract the mean of the data and divide by the standard deviation of the data to ensure the loss function weights the different parts of the state (e.g., positions and velocities) equally. We also add zero mean Gaussian noise to the training data (inputs and outputs) to increase model robustness. The training data is then stored in the dataset $\mathcal{D}$.

Training the model: We train the dynamics model $\hat{f}_\theta(\mathbf{s}_t, \mathbf{a}_t)$ by minimizing the error

$$\mathcal{E}(\theta) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \in \mathcal{D}} \frac{1}{2} \| (\mathbf{s}_{t+1} - \mathbf{s}_t) - \hat{f}_\theta(\mathbf{s}_t, \mathbf{a}_t) \|^2 \quad (1)$$

using stochastic gradient descent. While training on the training dataset $\mathcal{D}$, we also calculate the mean squared error in Eqn. 1 on a validation set $\mathcal{D}_{val}$, composed of trajectories not stored in the training dataset.

Although this error provides an estimate of how well our learned dynamics function is at predicting next state, we would in fact like to know how well our model can predict further into the future because we will ultimately use this model for longer-horizon control (Sec. IV-C). We therefore calculate H-step validation errors by propagating the learned dynamics function forward H times to make multi-step open-loop predictions. For each given sequence of true actions $(\mathbf{a}_t, \dots \mathbf{a}_{t+H-1})$ from $\mathcal{D}_{val}$, we compare the corresponding ground-truth states $(\mathbf{s}_{t+1}, \dots \mathbf{s}_{t+H})$ to the dynamics model's multi-step state predictions $(\hat{\mathbf{s}}_{t+1}, \dots \hat{\mathbf{s}}_{t+H})$, calculated as

$$\mathcal{E}_{val}^{(H)} = \frac{1}{\mathcal{D}_{val}} \sum_{\mathcal{D}_{val}} \frac{1}{H} \sum_{h=1}^{H} \frac{1}{2} \| \mathbf{s}_{t+h} - \hat{\mathbf{s}}_{t+h} \|^2 \quad :$$

$$\hat{\mathbf{s}}_{t+h} = \begin{cases} \mathbf{s}_t & h = 0 \\ \hat{\mathbf{s}}_{t+h-1} + \hat{f}_\theta(\hat{\mathbf{s}}_{t+h-1}, \mathbf{a}_{t+h-1}) & h > 0 \end{cases} \quad (2)$$

This H-step validation is used to analyze our experimental results, but otherwise not used during training.

C. Model-Based Control

In order to use the learned model $\hat{f}_\theta(\mathbf{s}_t, \mathbf{a}_t)$, together with a reward function $r(\mathbf{s}_t, \mathbf{a}_t)$ that encodes some task, we formulate a model-based controller that is both computationally tractable and robust to inaccuracies in the learned dynamics model. Expanding on the discussion in Sec. III, we first optimize the sequence of actions $\mathbf{A}_t^{(H)} = (\mathbf{a}_t, \cdots, \mathbf{a}_{t+H-1})$ over a finite horizon H, using the learned dynamics model to predict future states:

$$\mathbf{A}_t^{(H)} = \arg\max_{\mathbf{A}_t^{(H)}} \sum_{t'=t}^{t+H-1} r(\hat{\mathbf{s}}_{t'}, \mathbf{a}_{t'}) \quad :$$

$$\hat{\mathbf{s}}_t = \mathbf{s}_t, \hat{\mathbf{s}}_{t'+1} = \hat{\mathbf{s}}_{t'} + \hat{f}_\theta(\hat{\mathbf{s}}_{t'}, \mathbf{a}_{t'}). \quad (3)$$

Calculating the exact optimum of Eqn. 3 is difficult due to the dynamics and reward functions being nonlinear, but many techniques exist for obtaining approximate solutions to finite-horizon control problems that are sufficient for succeeding at the desired task. In this work, we use a simple random-sampling shooting method [38] in which K candidate action sequences are randomly generated, the corresponding state sequences are predicted using the learned dynamics model, the rewards for all sequences are calculated, and the candidate action sequence with the highest expected cumulative reward is chosen. Rather than have the policy execute this action

Algorithm 1 Model-based Reinforcement Learning

gather dataset $\mathcal{D}_{\text{RAND}}$ of random trajectories
initialize empty dataset $\mathcal{D}_{\text{RL}}$, and randomly initialize $\hat{f}_\theta$
for iter=1 **to** max_iter **do**
  train $\hat{f}_\theta(\mathbf{s}, \mathbf{a})$ by performing gradient descent on Eqn. 1, using $\mathcal{D}_{\text{RAND}}$ and $\mathcal{D}_{\text{RL}}$
  for $t = 1$ **to** T **do**
    get agent's current state $\mathbf{s}_t$
    use $\hat{f}_\theta$ to estimate optimal action sequence $\mathbf{A}_t^{(H)}$ (Eqn. 3)
    execute first action $\mathbf{a}_t$ from selected action sequence $\mathbf{A}_t^{(H)}$
    add $(\mathbf{s}_t, \mathbf{a}_t)$ to $\mathcal{D}_{\text{RL}}$
  end for
end for

sequence in open-loop, we use model predictive control (MPC): the policy executes only the first action $\mathbf{a}_t$, receives updated state information $\mathbf{s}_{t+1}$, and recalculates the optimal action sequence at the next time step. Note that for higher-dimensional action spaces and longer horizons, random sampling with MPC may be insufficient, and investigating other methods [39] in future work could improve performance.

Note that this combination of predictive dynamics model plus controller is beneficial in that the model is trained only once, but by simply changing the reward function, we can accomplish a variety of goals at run-time, without a need for live task-specific retraining.

D. Improving Model-Based Control with Reinforcement Learning

To improve the performance of our model-based learning algorithm, we gather additional on-policy data by alternating between gathering data with our current model and retraining our model using the aggregated data. This on-policy data aggregation (i.e., reinforcement learning) improves performance by mitigating the mismatch between the data's state-action distribution and the model-based controller's distribution [40]. Alg. 1 and Fig. 2 provide an overview of our model-based reinforcement learning algorithm.

First, random trajectories are collected and added to dataset $\mathcal{D}_{\text{RAND}}$, which is used to train $\hat{f}_\theta$ by performing gradient descent on Eqn. 1. Then, the model-based MPC controller (Sec. IV-C) gathers T new on-policy datapoints and adds these datapoints to a separate dataset $\mathcal{D}_{\text{RL}}$. The dynamics function $\hat{f}_\theta$ is then retrained using data from both $\mathcal{D}_{\text{RAND}}$ and $\mathcal{D}_{\text{RL}}$. Note that during retraining, the neural network dynamics function's weights are warm-started with the weights from the previous iteration. The algorithm continues alternating between training the model and gathering additional data until a predefined maximum iteration is reached. We evaluate design decisions related to data aggregation in our experiments (Sec. VI-A).

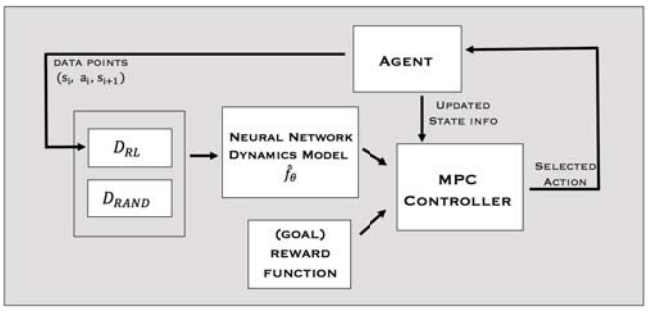


Fig. 2: Illustration of Algorithm 1. On the first iteration, random actions are performed and used to initialize $\mathcal{D}_{\text{RAND}}$. On all following iterations, this iterative procedure is used to train the dynamics model, run the MPC controller for action selection, aggregate data, and retrain the model.

V. MB-MF: Model-Based Initialization of Model-Free Reinforcement Learning Algorithm

The model-based reinforcement learning algorithm described above can learn complex gaits using very small numbers of samples, when compared to purely model-free learners. However, on benchmark tasks, its final performance still lags behind purely model-free algorithms. To achieve the best final results, we can combine the benefits of model-based and model-free learning by using the model-based learner to initialize a model-free learner. We propose a simple but highly effective method for combining our model-based approach with off-the-shelf, model-free methods by training a policy to mimic our learned model-based controller, and then using the resulting imitation policy as the initialization for a model-free reinforcement learning algorithm.

A. Initializing the Model-Free Learner

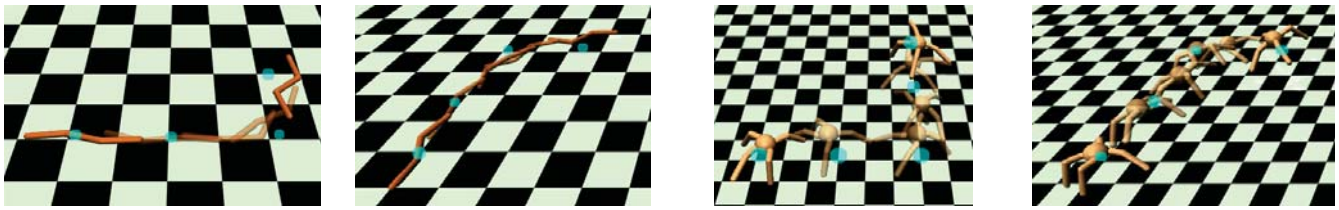
We first gather example trajectories with the MPC controller detailed in Sec. IV-C, which uses the learned dynamics function $\hat{f}_\theta$ that was trained using our model-based reinforcement learning algorithm (Alg. 1). We collect the trajectories into a dataset $\mathcal{D}^*$, and we then train a neural network policy $\pi_\phi(\mathbf{a}|\mathbf{s})$ to match these "expert" trajectories in $\mathcal{D}^*$. We parameterize π_ϕ as a conditionally Gaussian policy $\pi_\phi(\mathbf{a}|\mathbf{s}) \sim \mathcal{N}(\mu_\phi(\mathbf{s}), \Sigma_{\pi_\phi})$, in which the mean is parameterized by a neural network $\mu_\phi(\mathbf{s})$, and the covariance Σ_{π_ϕ} is a fixed matrix. This policy's parameters are trained using the behavioral cloning objective

$$\min_\phi \frac{1}{2} \sum_{(\mathbf{s}_t, \mathbf{a}_t) \in \mathcal{D}^*} ||\mathbf{a}_t - \mu_\phi(\mathbf{s}_t)||_2^2, \tag{4}$$

which we optimize using stochastic gradient descent. To achieve desired performance and address the data distribution problem, we applied DAGGER [40]: This consisted of iterations of training the policy, performing on-policy rollouts, querying the "expert" MPC controller for "true" action labels for those visited states, and then retraining the policy.

B. Model-Free Reinforcement Learning

After initialization, we can use the policy π_ϕ, which was trained on data generated by our learned model-based controller, as an initial policy for a model-free reinforcement learning algorithm. Specifically, we use trust region policy



(a) Swimmer left turn (b) Swimmer right turn (c) Ant left turn (d) Ant right turn

Fig. 3: Trajectory following samples showing turns with swimmer and ant, with blue dots representing the center-of-mass positions that were specified as the desired trajectory. For each agent, we train the dynamics model only once on random trajectories, but use it at run-time to execute various desired trajectories.

optimization (TRPO) [3]; such policy gradient algorithms are a good choice for model-free fine-tuning since they do not require any critic or value function for initialization [41], though our method could also be combined with other model-free RL algorithms.

TRPO is also a common choice for the benchmark tasks we consider, and it provides us with a natural way to compare purely model-free learning with our model-based pre-initialization approach. Initializing TRPO with our learned expert policy π_ϕ is as simple as using π_ϕ as the initial policy for TRPO, instead of a standard randomly initialized policy. Although this approach of combining model-based and model-free methods is extremely simple, we demonstrate the efficacy of this approach in our experiments.

VI. Experimental Results

We evaluated our model-based reinforcement learning approach (Alg. 1) on agents in the MuJoCo [8] physics engine. The agents we used were swimmer ($\mathcal{S} \in \mathbb{R}^{16}, \mathcal{A} \in \mathbb{R}^2$), hopper ($\mathcal{S} \in \mathbb{R}^{17}, \mathcal{A} \in \mathbb{R}^3$), half-cheetah ($\mathcal{S} \in \mathbb{R}^{23}, \mathcal{A} \in \mathbb{R}^6$), and ant ($\mathcal{S} \in \mathbb{R}^{41}, \mathcal{A} \in \mathbb{R}^8$). Relevant parameter values and implementation details are listed in the Appendix, and videos of all our experiments are provided online[1].

(a) Swimmer (b) Cheetah (c) Ant (d) Hopper

Fig. 4: Benchmark systems used in this paper. Agents on which we efficiently learn locomotion gaits, as well as combine our model-based approach with a model-free one to demonstrate fine-tuning performance.

A. Evaluating Design Decisions for Model-Based Reinforcement Learning

We first evaluate various design decisions for model-based reinforcement learning with neural networks using empirical evaluations with our model-based approach (Sec. IV). We explored these design decisions for the task of running forward as quickly as possible with the swimmer and half-cheetah agents; the other agents exhibited similar trends, and are therefore omitted for brevity. After each design decision was evaluated, we used the best outcome of that evaluation for the remainder of the evaluations.

(A) Training steps. Fig. 5a shows varying numbers of gradient descent steps taken during the training of the learned dynamics function. As expected, training for too few epochs negatively affects learning performance, with 20 epochs causing swimmer to reach only half of the other experiments' performance.

(B) Dataset aggregation. Fig. 5b shows varying amounts of (initial) random data versus (aggregated) on-policy data used within each mini-batch of stochastic gradient descent when training the learned dynamics function. We see that training with at least some aggregated on-policy rollouts significantly improves performance, revealing the benefits of improving learned models with reinforcement learning. However, our method still works well with even just 30% of each mini-batch coming from on-policy rollouts, showing the advantage of model-based reinforcement learning being off-policy.

(C) Controller. Fig. 5c shows the effect of varying the horizon H and the number of random samples K used at each time step by the model-based controller. We see that too short of a horizon is harmful for performance, perhaps due to greedy behavior and entry into unrecoverable states. Additionally, the model-based controller for half-cheetah shows worse performance for longer horizons. This is further revealed below in Fig. 6, which illustrates a single 100-step validation rollout (as explained in Eqn. 2). We see here that the open-loop predictions for certain state elements, such as the center of mass x position, diverge from ground truth. Thus, a large H leads to the use of an inaccurate model for making predictions, which is detrimental to task performance. Finally, with regards to the number of randomly sampled trajectories evaluated, we expect this value needing to be higher for systems with higher-dimensional action spaces.

(D) Number of initial random trajectories. Fig. 5d shows varying numbers of random trajectories used to initialize our model-based approach. We see that although a higher amount of initial training data leads to higher initial performance, data aggregation allows low-data initialization runs to reach a high final performance level, highlighting how on-policy data from reinforcement learning improves sample efficiency.

B. Trajectory Following with the Model-Based Controller

For the task of trajectory following, we evaluated our model-based reinforcement learning approach on the swimmer, ant, and half-cheetah environments (Fig. 3). Note that for these tasks, the dynamics model was trained using only random initial trajectories and was trained only once per agent, but the learned model was then used at run-time to accomplish different tasks. These results show that the models learned

[1] https://sites.google.com/view/mbmf

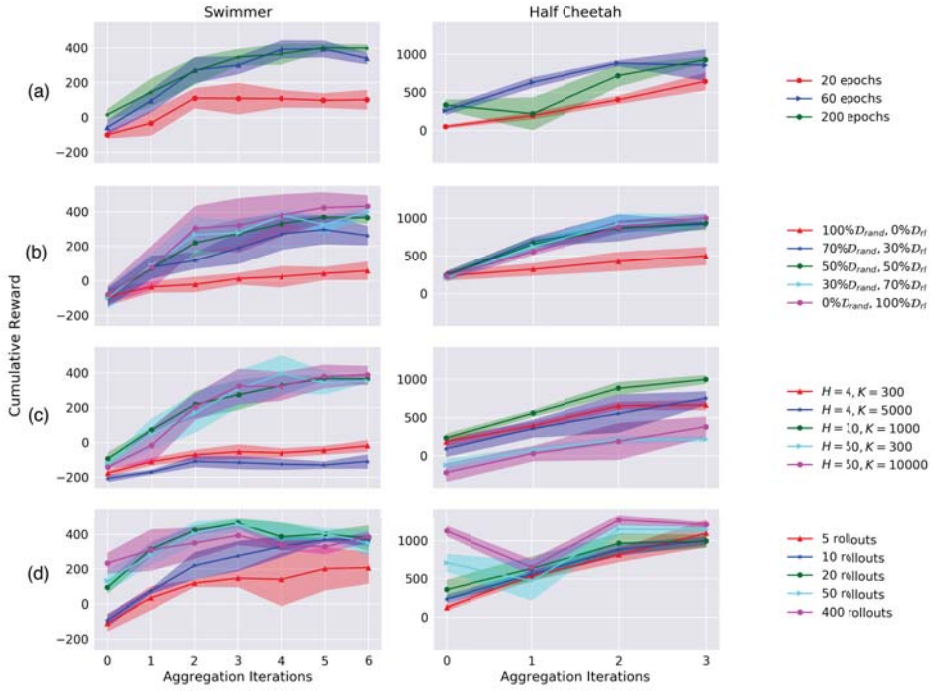


Fig. 5: Analysis of design decisions for our model-based reinforcement learning approach. (a) Training steps, (b) dataset training split, (c) horizon and number of actions sampled, (d) initial random trajectories. Training for more epochs, leveraging on-policy data, planning with medium-length horizons and many action samples were the best design choices, while data aggregation caused the number of initial trajectories that have little effect.

using our method are general enough to accommodate new tasks at test time, including tasks that are substantially more complex than anything that the robot did during training, such as following a curved path or making a U-turn. Furthermore, we show that even with the use of such a naïve random-sampling controller, the learned dynamics model is powerful enough to perform a variety of tasks.

The reward function we use requires the robot to track the desired x/y center of mass positions. This reward consists of one term to penalize the perpendicular distance away from the desired trajectory, and a second term to encourage forward movement in the direction of the desired trajectory. The reward function does not tell the robot anything about how the limbs should be moved to accomplish the desired center of mass trajectory. The model-based algorithm must discover a suitable gait entirely on its own. Further details about this reward are included in the appendix.

C. Mb-Mf Approach on Benchmark Tasks

We now compare our pure model-based approach with a pure model-free method on standard benchmark locomotion tasks, which require a simulated robot (swimmer, half-cheetah, hopper, or ant) to learn the fastest forward-moving gait possible. The model-free approach we compare with is the rllab [42] implementation of trust region policy optimization (TRPO) [3], which has obtained state-of-the-art results on these tasks.

We note that due to short-horizon planning as well as having finite sample sizes, this optimization for performing action selection will be suboptimal even with a perfect dynamics model. However, as shown in Fig. 5c, shorter horizons tend to perform better with learned models, since they minimize the ability of the optimizer to exploit an imperfect learned model.

For our model-based approach, we used the OpenAI gym [7] standard reward functions (described in the appendix) for action selection in order to allow us to compare performance to model-free benchmarks. These reward functions primarily reward speed, and are especially difficult for our model-based method due to the myopic nature of the short-horizon MPC that we employ for action selection; therefore, the results of our model-based algorithm on all following plots are lower than would be if we designed our own reward function (for instance, a straight-line trajectory-following reward function).

Even with the extremely simplistic standard reward functions, our method can very quickly learn a gait that makes forward progress. The swimmer, for example, can quickly achieve qualitatively good moving forward behavior at $20\times$ faster than the model-free method. However, the final achieved reward attained by the pure model-based variant of our approach does not match the final performance of state-of-the-art model-free learners, due to an imperfect learned model and the previously discussed sources of suboptimality. When we integrate model-free finetuning (Fig. 7), however, the asymptotic performance improves to the level of purely model-free learning. In the case of the hopper, our pure model-based approach learns to perform a double or triple hop very quickly in 1×10^4 steps, but performance plateaus as the reward signal of just forward velocity is not enough for the limited-horizon controller to keep the hopper upright for longer periods of time. Our hybrid Mb-Mf approach

takes these quickly-learned gaits and performs model-free fine-tuning in order to achieve high task rewards, achieving $3-5\times$ sample efficiency gains over pure model-free methods for all agents.

VII. Discussion

We presented a model-based reinforcement learning algorithm that is able to learn neural network dynamics functions for complex simulated locomotion tasks using a small number of samples. Although a number of prior works have explored model-based learning with neural network dynamics models, our method achieves excellent performance on a number of challenging locomotion problems that exceed the complexity demonstrated in prior methods.

We described a number of important design decisions for effectively and efficiently training neural network dynamics models, and we presented detailed experiments that evaluated these design parameters. Our method quickly discovered a dynamics model that led to an effective gait; that model could be applied to different trajectory following tasks at run-time, or the initial gait could then be fine-tuned with model-free learning to achieve high task rewards on benchmark Mujoco agents.

In addition to looking at the difference in sample complexity between our hybrid Mb-Mf approach and a pure model-free approach, there are also takeaways from the model-based approach alone. Our model-based algorithm cannot always reach extremely high rewards on its own, but it offers practical use by allowing quick and successful discovery of complex and realistic gaits. In general, our model-based approach can very quickly become competent at a task, whereas model-free approaches can very slowly become experts. For example, when we have a small legged robot with unknown dynamics and we want it to accomplish tasks in the real-world (such as exploration, construction, search and rescue, etc.), achieving reliable walking gaits that can follow any desired trajectory is a superior skill to that of just running straight forward as fast as possible. Additionally, consider the ant: A model-free approach requires 5×10^6 points to achieve a steady walking forward gait, but using just 14% of those data points, our model-based approach can allow for travel in any direction and along arbitrary

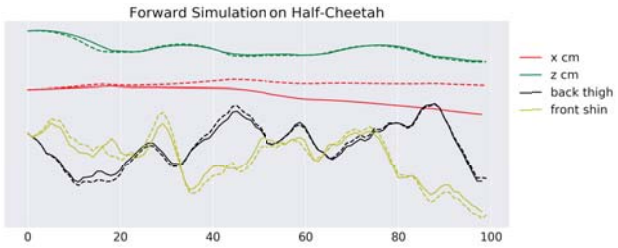


Fig. 6: Given a fixed sequence of controls, we show the resulting true rollout (solid line) vs. the multi-step prediction from the learned dynamics model (dotted line) on the half-cheetah agent. Although we learn to predict certain elements of the state space well, note the eventual divergence of the learned model on some state elements when it is used to make multi-step open-loop predictions. However, our MPC-based controller with a short horizon can succeed in using the model to control an agent.

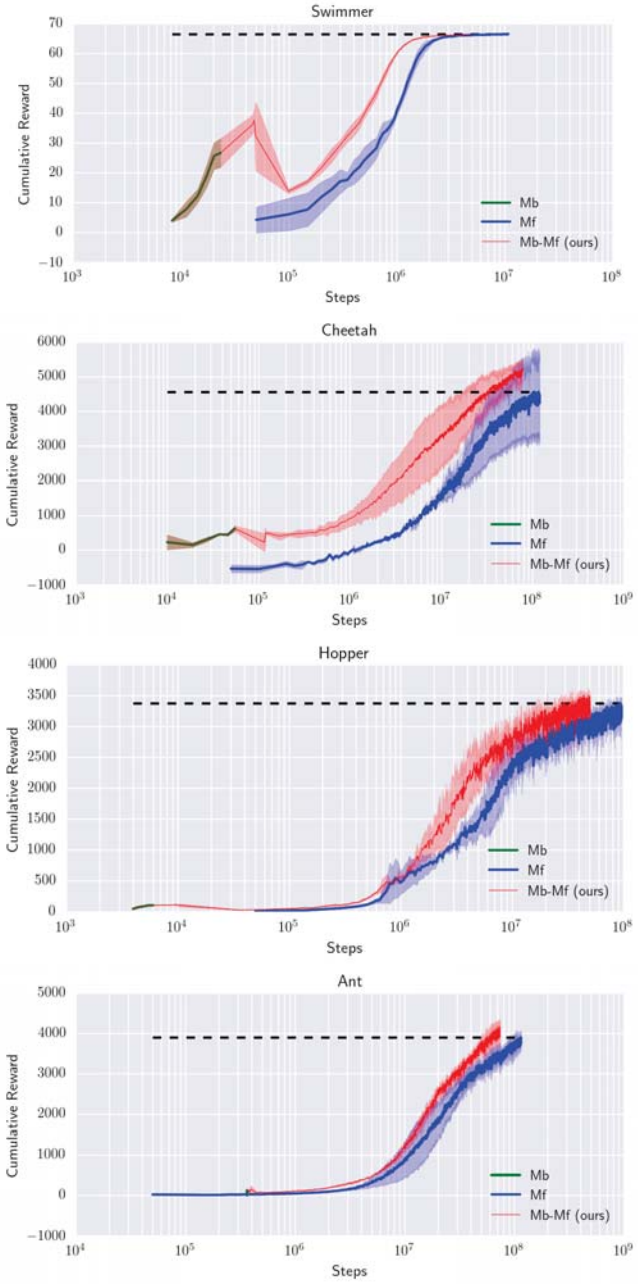


Fig. 7: Plots show the mean and standard deviation over multiple runs and compare our model-based approach, a model-free approach (TRPO [3]), and our hybrid model-based plus model-free approach. Our combined approach shows a $3-5\times$ improvement in sample efficiency for all shown agents. Note that the x-axis uses a logarithmic scale.

desired trajectories. Training such a dynamics model only once and applying it to various tasks is compelling; especially when looking toward application to real robots, this sample efficiency can bring these methods out of the simulation world and into the realm of feasibility.

While the simplicity and effectiveness of our Mb-Mf approach is enticing for ease of practical application, an interesting avenue for future work is to integrate our model-based approach more tightly and elegantly with model-free learners (Q-learning, actor-critic methods), in order to provide further sample efficiency gains. Another exciting direction for

future work is to deploy this method on real-world robotic systems, where the improved sample efficiency would make it practical to use even under the constraints of real-time sample collection in the real world. In addition to taking communication delays and computational limitations into account, another line of future work includes improving the MPC controller. In this paper, we chose to use a naïve random-sampling controller to further emphasize the power of the learned dynamics models; however, this may not be feasible on real systems with limited computational power, or on systems with high-dimensional actions spaces that would require a large number of actions to sampled. Thus, further development of a real-time controller via optimization techniques is compelling future work.

VIII. Acknowledgements

We thank Vitchyr Pong for running some baseline experiments. This work is supported by the National Science Foundation under the National Robotics Initiative, Award CMMI-1427096, as well as IIS-1637443, IIS-1614653, and a NSF Graduate Research Fellowship.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," in *Workshop on Deep Learning, NIPS*, 2013.

[2] J. Oh, V. Chockalingam, S. Singh, and H. Lee, "Control of memory, active perception, and action in minecraft," in *ICML*, 2016.

[3] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust region policy optimization," in *ICML*, 2015.

[4] M. P. Deisenroth, G. Neumann, J. Peters, *et al.*, "A survey on policy search for robotics," in *Foundations and Trends in Robotics*, 2013.

[5] R. Lioutikov, A. Paraschos, J. Peters, and G. Neumann, "Sample-based information-theoretic stochastic optimal control," in *ICRA*, 2014.

[6] M. Deisenroth and C. Rasmussen, "A model-based and data-efficient approach to policy search," in *ICML*, 2011.

[7] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," 2016.

[8] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IROS*, 2012.

[9] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep Q-learning with model-based acceleration," in *ICML*, 2016.

[10] N. Mishra, P. Abbeel, and I. Mordatch, "Prediction and control with temporal segment models," in *ICML*, 2017.

[11] A. Richards, "Robust constrained model predictive control," Ph.D. dissertation, MIT, 2004.

[12] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *IJRR*, 2013.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," in *Nature*, 2015.

[14] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *ICRL*, 2016.

[15] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *ICML*, 2016.

[16] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *ICLR*, 2016.

[17] S. Gu, T. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine, "Q-Prop: sample-efficient policy gradient with an off-policy critic," in *ICLR*, 2017.

[18] J. Morimoto and C. G. Atkeson, "Minimax differential dynamic programming: An application to robust biped walking," in *NIPS*, 2003.

[19] D. Meger, J. C. G. Higuera, A. Xu, P. Giguere, and G. Dudek, "Learning legged swimming gaits from experience," in *ICRA*, 2015.

[20] M. P. Deisenroth, C. E. Rasmussen, and D. Fox, "Learning to control a low-cost manipulator using data-efficient reinforcement learning," 2011.

[21] J. Boedecker, J. T. Springenberg, J. Wülfing, and M. Riedmiller, "Approximate real-time optimal control based on sparse gaussian process models," in *ADPRL*, 2014.

[22] J. Ko and D. Fox, "GP-BayesFilters: Bayesian filtering using gaussian process prediction and observation models," in *IROS*, 2008.

[23] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," in *NIPS*, 2014.

[24] M. C. Yip and D. B. Camarillo, "Model-less feedback control of continuum manipulators in constrained environments," in *IEEE Transactions on Robotics*, 2014.

[25] S. M. Khansari-Zadeh and A. Billard, "Learning stable nonlinear dynamical systems with gaussian mixture models," in *IEEE Transactions on Robotics*, 2011.

[26] M. P. Deisenroth, R. Calandra, A. Seyfarth, and J. Peters, "Toward fast policy search for learning legged locomotion," in *IROS*, 2012.

[27] K. J. Hunt, D. Sbarbaro, R. Żbikowski, and P. J. Gawthrop, "Neural networks for control systems—a survey," in *Automatica*, 1992.

[28] G. Bekey and K. Y. Goldberg, *Neural networks in robotics*. Springer US, 1992.

[29] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller, "Embed to control: a locally linear latent dynamics model for control from raw images," in *NIPS*, 2015.

[30] N. Wahlström, T. B. Schön, and M. P. Deisenroth, "From pixels to torques: policy learning with deep dynamical models," *Deep learning workshop at ICML*, 2015.

[31] Y. Gal, R. T. McAllister, and C. E. Rasmussen, "Improving PILCO with bayesian neural network dynamics models," in *Data-Efficient Machine Learning workshop*, 2016.

[32] R. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," in *AAAI*, 1991.

[33] D. Silver, R. S. Sutton, and M. Müller, "Sample-based learning and search with permanent and transient memories," in *ICML*, 2008.

[34] K. Asadi, "Strengths, weaknesses, and combinations of model-based and model-free reinforcement learning," Ph.D. dissertation, University of Alberta, 2015.

[35] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa, "Learning continuous control policies by stochastic value gradients," in *NIPS*, 2015.

[36] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," in *JMLR*, 2017.

[37] A. Y. Ng, S. J. Russell, *et al.*, "Algorithms for inverse reinforcement learning," in *ICML*, 2000.

[38] A. Rao, "A survey of numerical methods for optimal control," in *Advances in the Astronautical Sciences*, 2009.

[39] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems," in *ICINCO*, 2004.

[40] S. Ross, G. J. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *AISTATS*, 2011.

[41] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: standard and natural policy gradients," in *IEEE Transactions on Systems, Man, and Cybernetics*, 2012.

[42] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *ICML*, 2016.

[43] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2014.