Lagrangian Model Based Reinforcement Learning

Adithya Ramesh
Robert Bosch Centre for Data Science and Artificial Intelligence
Indian Institute of Technology Madras

Balaraman Ravindran
Robert Bosch Centre for Data Science and Artificial Intelligence
Department of Computer Science and Engineering
Indian Institute of Technology Madras

Abstract

We apply reinforcement learning (RL) to robotics tasks. One of the drawbacks of traditional RL algorithms has been their poor sample efficiency. In robotics, collecting large amounts of training data using actual robots is not practical. One approach to improve the sample efficiency is model-based RL. In our model-based RL algorithm, we learn a model of the environment, essentially its transition dynamics and reward function, use it to generate imaginary trajectories and backpropagate through them to update the policy, exploiting the differentiability of the model. Intuitively, learning better environment models should lead to better model-based RL performance. Recently, there has been growing interest in developing better deep neural network based dynamics models for physical systems, through better inductive biases. We focus on robotic systems undergoing rigid body motion. We utilize the structure of rigid body dynamics to learn a Lagrangian Neural Network and use it to train our model-based RL algorithm. To the best of our knowledge, we are the first to explore such an approach. While it is intuitive that such physics-informed dynamics models will improve model-based RL performance, it is not apparent in which environments we will significantly benefit from such an approach. We show that, in environments with underactuation and high inherent sensitivity to initial conditions, characterized by a large maximal Lyapunov exponent under freefall, we are likely to significantly benefit from learning physics-informed dynamics models. In such environments, our Lagrangian model-based RL approach achieves better average-return and sample efficiency than a version of our model-based RL algorithm that uses a standard deep neural network based dynamics model, as well as Soft Actor-Critic, a state-of-the-art model-free RL algorithm.

1 Introduction

We apply reinforcement learning (RL) to robotics tasks. RL can solve sequential decision making problems through trial and error. In recent years, RL has been combined with powerful function approximators such as deep neural networks to successfully solve complex robotics tasks with high-dimensional, continuous state and action spaces (Lillicrap et al., 2015; Schulman et al., 2017; Haarnoja et al., 2018a; Fujimoto et al., 2018; Haarnoja et al., 2018b; Andrychowicz et al., 2020). However, there remain some critical challenges that must be addressed, to take more RL based robotic systems to the real world (Dulac-Arnold et al., 2021). One of these challenges is sample efficiency.

One of the drawbacks of traditional RL algorithms has been their poor sample efficiency – they require a large number of interactions with the environment to learn successful policies. In robotics, collecting such large amounts of training data using actual robots is not practical. One solution is

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

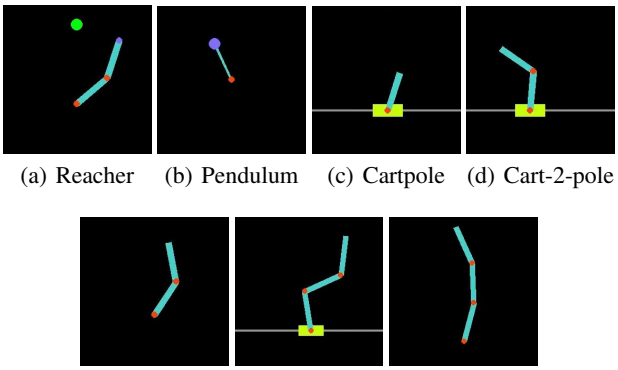
to train in simulation. However, in many robotics tasks, developing realistic simulations is hard. Another solution is to improve the sample efficiency. One approach to improve the sample efficiency is model-based RL, where we learn a model of the environment, essentially its transition dynamics and reward function, use it to generate imaginary trajectories and use them to update the policy. In our model-based RL algorithm, we exploit the differentiability of the model and backpropagate through the imaginary trajectories to update the policy, similar to Dreamer (Hafner et al., 2019, 2020). Intuitively, learning better environment models should lead to better model-based RL performance.

Recently there has been growing interest in developing better deep neural network based dynamics models for physical systems through better inductive biases (Lutter et al., 2019a; Greydanus et al., 2019; Lutter et al., 2019b; Zhong et al., 2019, 2020; Cranmer et al., 2020; Finzi et al., 2020; Zhong et al., 2021a). These physics-informed models predict the future states more accurately compared to standard deep neural networks. They also obey the underlying physical laws, such as conservation of energy better. Previous studies in this area have mostly focused on improving dynamics learning. Some studies have learnt a physics-informed dynamics model and used it for inverse dynamics control (Lutter et al., 2019a) and energy based control (Lutter et al., 2019b; Zhong et al., 2019). We learn a physics-informed dynamics model and use it to train a model-based RL algorithm. To the best of our knowledge, we are the first to explore such an approach.

We focus on robotic systems undergoing rigid body motion. We utilize the structure of rigid body dynamics to learn a Lagrangian Neural Network (Lutter et al., 2019a,b; Cranmer et al., 2020) and use it to train our model-based RL algorithm. While it is intuitive that such physics-informed dynamics models will improve model-based RL performance, it is not apparent in which environments we will significantly benefit from such an approach. We show that, in environments with underactuation and high inherent sensitivity to initial conditions, characterized by a large maximal Lyapunov exponent under freefall, we are likely to significantly benefit from learning physics-informed dynamics models. In such environments, our Lagrangian model-based RL approach achieves better average-return and sample efficiency than a version of our model-based RL algorithm that uses a standard deep neural network based dynamics model, as well as Soft Actor-Critic (Haarnoja et al., 2018b), a state-of-the-art model-free RL algorithm.

2 Environments Considered

We focus on robotic systems undergoing rigid body motion. The environments considered are shown in Figure 1. In this work, we assume that there is no friction or contacts. In future work, we plan to include these effects as well. We provide more details about the environments, such as the task to be accomplished, which joints are acutated, etc, in Appendix A. We develop our own simulations from first principles.



(a) Reacher (b) Pendulum (c) Cartpole (d) Cart-2-pole

(e) Acrobot (f) Cart-3-pole (g) Acro-3-bot

Figure 1: We consider robotic systems undergoing rigid body motion. We assume that there is no friction or contacts.

3 Model-Based RL

In model-based RL, the training essentially iterates over three steps. First is the environment interaction step, where we use the current policy to interact with the environment and gather data. Second is the model learning step, where we use the gathered data to learn the dynamics and reward models. Third is the behaviour learning step, where we use the learned model to generate imaginary trajectories and use them to update the policy. We discuss the model learning and behaviour learning steps in detail below.

3.1 Model Learning

In the model learning step, we learn the dynamics and reward models. In dynamics learning, we want to predict the next state, given the current state and action. We model these systems using Lagrangian mechanics. Hence, the state consists of generalized coordinates $\mathbf{q}$, which describe the configuration of the system, and generalized velocities $\dot{\mathbf{q}}$, which are the time derivatives of $\mathbf{q}$. The action is the motor torque $\boldsymbol{\tau}$. So, in dynamics learning, we essentially want to learn the transformation $(\mathbf{q}_t, \dot{\mathbf{q}}_t, \boldsymbol{\tau}_t) \rightarrow (\mathbf{q}_{t+1}, \dot{\mathbf{q}}_{t+1})$. The most straightforward solution is to train a standard deep neural network. We refer to this approach as DNN. This is shown in Figure 2(a). Another approach is to utilize the structure of the underlying Lagrangian mechanics. This approach builds upon recent work such as Deep Lagrangian Networks (DeLaN) (Lutter et al., 2019a), DeLaN for energy control (Lutter et al., 2019b) and Lagrangian Neural Networks (Cranmer et al., 2020). We detail this approach below.

In Lagrangian mechanics, the Lagrangian is a scalar quantity defined as $\mathcal{L}(\mathbf{q}, \dot{\mathbf{q}}, t) = \mathcal{T}(\mathbf{q}, \dot{\mathbf{q}}) - \mathcal{V}(\mathbf{q})$, where $\mathcal{T}(\mathbf{q}, \dot{\mathbf{q}})$ is the kinetic energy and $\mathcal{V}(\mathbf{q})$ is the potential energy. The Lagrangian equations of motion are given by,

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\mathbf{q}}} - \frac{\partial \mathcal{L}}{\partial \mathbf{q}} = \boldsymbol{\tau} \tag{1}$$

For systems undergoing rigid body motion, the kinetic energy is given by $\mathcal{T}(\mathbf{q}, \dot{\mathbf{q}}) = \frac{1}{2}\,\dot{\mathbf{q}}^T\,\mathbf{M}(\mathbf{q})\,\dot{\mathbf{q}}$, where $\mathbf{M}(\mathbf{q})$ is the mass matrix, which is symmetric and positive definite. Hence, the Lagrangian is given by $\mathcal{L}(\mathbf{q}, \dot{\mathbf{q}}, t) = \frac{1}{2}\,\dot{\mathbf{q}}^T\,\mathbf{M}(\mathbf{q})\,\dot{\mathbf{q}} - \mathcal{V}(\mathbf{q})$. Substituting this expression into Equation 1, we get the rigid body equations of motion,

$$\mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \underbrace{\frac{\partial}{\partial \mathbf{q}}\left(\mathbf{M}(\mathbf{q})\,\dot{\mathbf{q}}\right)\dot{\mathbf{q}} - \frac{\partial}{\partial \mathbf{q}}\left(\frac{1}{2}\,\dot{\mathbf{q}}^T\,\mathbf{M}(\mathbf{q})\,\dot{\mathbf{q}}\right)}_{\mathbf{C}(\mathbf{q},\dot{\mathbf{q}})\,\dot{\mathbf{q}}} + \underbrace{\frac{\partial \mathcal{V}(\mathbf{q})}{\partial \mathbf{q}}}_{\mathbf{G}(\mathbf{q})} = \boldsymbol{\tau} \tag{2}$$

Here, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}}$ represents the centripetal / Coriolis forces and $\mathbf{G}(\mathbf{q})$ represents the conservative forces (e. g., gravity). Rearranging Equation 2 we get,

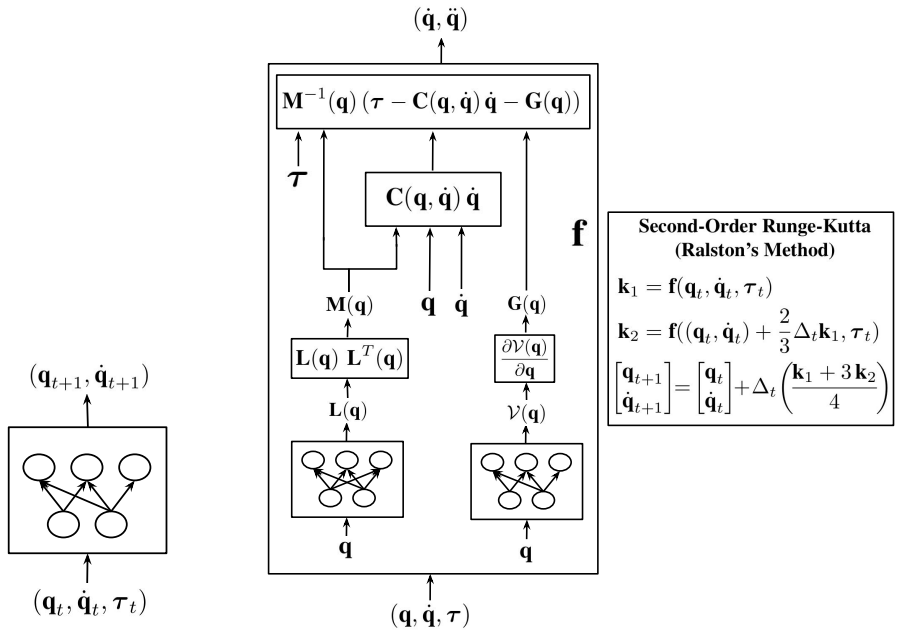
$$\ddot{\mathbf{q}} = \mathbf{M}^{-1}(\mathbf{q})\left(\boldsymbol{\tau} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q})\right) \tag{3}$$

By integrating the state derivative $(\dot{\mathbf{q}}, \ddot{\mathbf{q}})$ over one time step, we can obtain the next state,

$$\begin{bmatrix}\mathbf{q}_{t+1}\\ \dot{\mathbf{q}}_{t+1}\end{bmatrix} = \begin{bmatrix}\mathbf{q}_{t}\\ \dot{\mathbf{q}}_{t}\end{bmatrix} + \int_{t\Delta t}^{(t+1)\Delta t} \begin{bmatrix}\dot{\mathbf{q}}\\ \ddot{\mathbf{q}}\end{bmatrix} dt \tag{4}$$

We use one network to learn the potential energy function $\mathcal{V}(\mathbf{q})$ and another network to learn a lower triangular matrix $\mathbf{L}(\mathbf{q})$, using which we compute the mass matrix as $\mathbf{M}(\mathbf{q}) = \mathbf{L}(\mathbf{q})\,\mathbf{L}^T(\mathbf{q})$. We then compute $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}}$ and $\mathbf{G}(\mathbf{q})$ as per Equation 2 and $\ddot{\mathbf{q}}$ as per Equation 3. Then, as per Equation 4, we integrate $(\dot{\mathbf{q}}, \ddot{\mathbf{q}})$ over one time step, numerically, using second-order Runge-Kutta, to compute the next state. We refer to this approach as LNN, short for Lagrangian Neural Network. The entire process is shown in Figure 2(b). In both the DNN and LNN approaches to dynamics learning, we use the L1 error between the predicted state and the ground truth as the loss for training.

In reward learning, we want to learn the reward function. In general, the reward is a function of the current state, action and the next state. In our case, the reward only depends on the next state. Hence, we train a network to map the next state to the reward. We use the L1 error between the predicted reward and the ground truth as the loss for training.



(a) DNN (b) LNN

Figure 2: Two approaches to dynamics learning, i. e., learning the transformation $(\mathbf{q}_t, \dot{\mathbf{q}}_t, \boldsymbol{\tau}_t) \rightarrow (\mathbf{q}_{t+1}, \dot{\mathbf{q}}_{t+1})$: In DNN, we use a standard deep neural network. In LNN, we utilize the structure of the underlying Lagrangian mechanics to estimate $\ddot{\mathbf{q}}$, given $(\mathbf{q}, \dot{\mathbf{q}}, \boldsymbol{\tau})$. We then numerically integrate $(\dot{\mathbf{q}}, \ddot{\mathbf{q}})$ over one time step using second-order Runge-Kutta, to compute the next state $(\mathbf{q}_{t+1}, \dot{\mathbf{q}}_{t+1})$.

3.2 Behaviour Learning

In the behaviour learning step, we use the learned model to generate imaginary trajectories and use them to update the policy. Some model-based RL algorithms use the model just to generate additional data. To update the policy, they use a model-free algorithm. Algorithms such as Dyna (Sutton, 1991) and MBPO (Janner et al., 2019) adopt this approach. Other model-based RL algorithms exploit the differentiability of the model and backpropagate through the imaginary trajectories to update the policy. Algorithms such as PILCO (Deisenroth and Rasmussen, 2011), SVG (Heess et al., 2015), Model-Augmented Actor-Critic (Clavera et al., 2020) and Dreamer (Hafner et al., 2019, 2020) adopt this approach. We follow the latter approach.

We build upon the Dreamer algorithm (Hafner et al., 2019, 2020). We adopt an actor-critic approach. The critic aims to predict the expected discounted return from a given state. We train the critic to regress the λ-return (Sutton and Barto, 2018; Schulman et al., 2015). We stabilize the critic training by computing the λ-return using a target network that is updated every 100 critic updates. The critic loss function is given by $L(w) = \mathbb{E}\,[\sum_{t=0}^{T-1} \frac{1}{2}(\,V(s_t; w) - \text{sg}\,(V'_\lambda(s_t))\,)^2]$, where,

$$V'_\lambda(s_t) = \begin{cases} r_t + \gamma\,(\,(1-\lambda)V'(s_{t+1}; w') + \lambda V'_\lambda(s_{t+1})\,) & \text{if } t < T \\ V'(s_t; w') & \text{if } t = T \end{cases} \tag{5}$$

We stop the gradients around the target (denoted by the sg$(.)$ function), as is typical in the literature.

We use a stochastic actor. The actor aims to output actions that lead to states that maximize the expected discounted return. We train the actor to maximize the same λ-return that was computed to train the critic. We add an entropy term to the actor objective to encourage exploration. The overall actor loss function is given by $L(\theta) = -\,\mathbb{E}\,[\sum_{t=0}^{T-1}[\,V'_\lambda(s_t) - \eta \log \pi(a_t|s_t; \theta)\,]]$. To backpropagate through sampled actions, we use the reparameterization trick (Kingma and Welling, 2013). The actor network outputs the mean μ and standard deviation σ of a Gaussian distribution, from which we obtain the action as $a_t = \tanh(\mu_\theta(s_t) + \sigma_\theta(s_t) \cdot \epsilon)$, where $\epsilon \sim \mathcal{N}(0, \mathbb{I})$. We summarize our overall model-based RL algorithm in Algorithm 1.

Algorithm 1 MBRL Algorithm

```
Initialize networks with random weights.
Execute random actions for K episodes to initialize replay buffer.
for each episode do
    // Model Learning
    for N_1 times do
        Draw mini batch of transitions from replay buffer.
        Fit dynamics and reward models.
    end for
    // Behaviour Learning
    for N_2 times do
        Draw mini batch of states from replay buffer.
        From each state, imagine a trajectory of length T.
        Backpropagate through imaginary trajectories to update actor, critic.
        Every 100 updates, copy critic weights into critic target.
    end for
    // Environment Interaction
    Interact with environment for one episode using current policy.
    Add transitions to replay buffer.
end for
```

3.3 Experiments

We train two versions of our model-based RL algorithm, one which uses the DNN approach for dynamics learning and the other which uses the LNN approach. We refer to them as MBRL-DNN and MBRL-LNN respectively. In both versions, we use an imagination horizon of 16 time steps. In addition, we train a state-of-the-art model-free RL algorithm, Soft Actor-Critic (SAC) (Haarnoja et al., 2018b), to serve as a baseline. We train each algorithm on five random seeds.

4 Results

We record the results from the experiments in Table 1. The training curves are shown in Figure 3. We summarize the results below.

- Across environments, MBRL-LNN achieves significantly lower dynamics error than MBRL-DNN. The reward error for both methods are similar.
- In Reacher, Pendulum and Cartpole, all the methods, i. e., MBRL-DNN, MBRL-LNN and SAC, successfully solve the task and achieve similar average-return.
- In Cart-2-pole, again, all the methods successfully solve the task. However, in terms of average-return, MBRL-LNN $>$ SAC $>$ MBRL-DNN.
- In Acrobot, Cart-3-pole and Acro-3-bot, only MBRL-LNN and SAC successfully solve the task, while MBRL-DNN is unsuccessful. Again, in terms of average-return, MBRL-LNN $>$ SAC $>$ MBRL-DNN.
- Across environments, MBRL-LNN achieves similar or better average return than MBRL-DNN and SAC, while requiring fewer samples. In other words, MBRL-LNN achieves better sample efficiency.

5 Analysis

We try to understand the results better. Reacher, Pendulum and Cartpole appear to be simple environments where all methods perform well. Cart-2-pole, Acrobot, Cart-3-pole and Acro-3-bot appear to be more challenging environments, where there is a noticeable difference in the performance of different methods. We try to understand what makes these latter environments more challenging and why certain methods perform better in these environments than others.

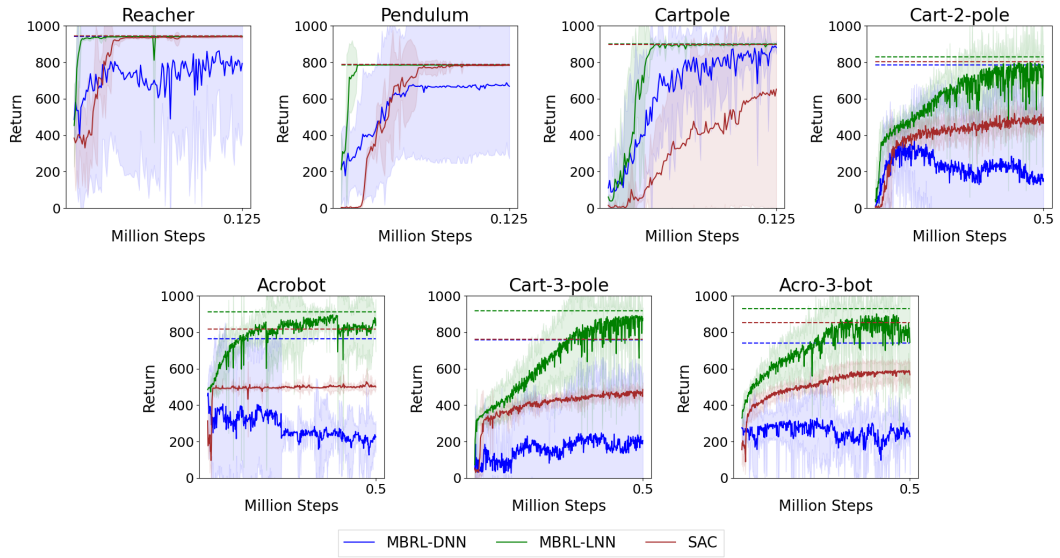


Figure 3: Training curves for MBRL-DNN, MBRL-LNN and SAC experiments. The solid curves represent the mean performance over all the seeds. The shaded region represents the 95 % confidence interval. The dashed lines indicate the best average-return after convergence.

5.1 State Space and Action Space

In Appendix A, we list the dimensions of the state space and action space of all the environments. Broadly speaking, higher-dimensional state spaces are more challenging. Actuation matters too. Reacher and Pendulum are fully actuated. The rest of the environments are underactuated. The level of underactuation varies. Cartpole and Cart-2-pole are somewhat less underactuated, while Acrobot, Cart-3-pole and Acro-3-bot are somewhat more underactuated. Underactuation makes it harder to learn successful policies, hence we need more samples.

5.2 Environment Dynamics

We try to characterize the underlying dynamics of each environment by calculating their Lyapunov exponents. Lyapunov exponents measure the rate of separation of trajectories which start from nearby initial states. If the initial separation vector is $\mathbf{u}_0$ and the separation vector at time t is $\mathbf{u}_t$, then the Lyapunov exponent is defined as $\lambda = \frac{1}{t} \log \frac{\|\mathbf{u}_t\|}{\|\mathbf{u}_0\|}$. The rate of separation varies based on the orientation of the initial separation vector. In general, there is a spectrum of Lyapunov exponents, equal in number to the dimensionality of the state space, each associated with a direction. It is common to refer to the largest Lyapunov exponent as the maximal Lyapunov exponent (MLE). The rate of separation is maximum along the direction associated with the MLE. An arbitrary initial separation vector will typically contain some component in this direction, and because of the exponential growth rate, this component will dominate the evolution. The MLE is a measure of the system's sensitivity to initial conditions. It also represents how quickly numerical errors will accumulate. Thus, it is also a measure of the predictability of a system.

The standard version of the MLE is concerned with the maximal growth rate in the long-term, i. e., $t \to \infty$. We refer to this as the long-term MLE. Under a trained RL policy, the separation between two trajectories starting from nearby initial states will become zero in the long-term, as they will be driven to the same goal state. Hence, the long-term MLE under a trained RL policy is simply $-\infty$.

Freefall MLE : To get an idea of the system's inherent sensitivity to initial conditions, independent of any control policy, we set the control inputs to zero and compute the long-term MLE. We refer to this as the freefall MLE. All environments except Reacher are set in the vertical plane. In these environments, we start near the goal state (vertically upright) and let the system fall freely under just

Environment	Method	Steps	Dynamics Error	Reward Error	Average Return	Solved (Y/N)	Freefall MLE	Transient MLE	Short Term MLE	Trajectory Error
Reacher	MBRL-DNN	0.25 M	126.57e-5	7.2217e-4	942.7	**Y**	0.0247	-0.0051	-1.3567	1.0984
	MBRL-LNN	**0.125 M**	**4.7557e-5**	**8.4078e-4**	**943.4**	**Y**			**-2.1170**	**0.2036**
	SAC	1.25 M	–	–	943.2	**Y**			–	–
Pendulum	MBRL-DNN	0.25 M	108.93e-5	**8.198e-4**	**787.3**	**Y**	0.0346	-0.0811	1.4055	0.6933
	MBRL-LNN	**0.125 M**	**3.4559e-5**	8.3962e-4	786.7	**Y**			**0.3607**	**0.1099**
	SAC	1.25 M	–	–	786.8	**Y**			–	–
Cartpole	MBRL-DNN	0.25 M	149.31e-5	**7.5226e-4**	899.4	**Y**	0.0718	-0.4626	2.0445	0.5141
	MBRL-LNN	**0.125 M**	**6.014e-5**	8.6215e-4	**900.0**	**Y**			**1.1985**	**0.1645**
	SAC	1.25 M	–	–	897.4	**Y**			–	–
Cart-2-pole	MBRL-DNN	2 M	499.94e-5	16.049e-4	786.4	**Y**	1.1330	0.3241	5.0024	2.6164
	MBRL-LNN	**0.5 M**	**70.389e-5**	**8.8135e-4**	**828.3**	**Y**			**1.6821**	**0.9744**
	SAC	10 M	–	–	801.7	**Y**			–	–
Acrobot	MBRL-DNN	2 M	799.90e-5	8.848e-4	764.5	N	1.0340	0.4265	11.7236	4.8975
	MBRL-LNN	**0.5 M**	**48.046e-5**	**7.9032e-4**	**911.8**	**Y**			**0.9475**	**2.2121**
	SAC	10 M	–	–	817.8	**Y**			–	–
Cart-3-pole	MBRL-DNN	2 M	1196.0e-5	17.044e-4	757.0	N	3.0904	0.8099	8.5330	18.2503
	MBRL-LNN	**0.5 M**	**61.3e-5**	**6.5141e-4**	**916.6**	**Y**			**3.7834**	**1.3601**
	SAC	10 M	–	–	759.4	**Y**			–	–
Acro-3-bot	MBRL-DNN	2 M	1322.0e-5	16.6256e-4	739.7	N	3.2265	0.6230	9.7397	16.0338
	MBRL-LNN	**0.5 M**	**156.34e-5**	**10.308e-4**	**929.4**	**Y**			**3.1205**	**2.8153**
	SAC	10 M	–	–	853.5	**Y**			–	–

Table 1: Overall results from MBRL-DNN, MBRL-LNN and SAC experiments. The data shown corresponds to the best performing checkpoint (in terms of average-return).

the influence of gravity. In Reacher, which is set in the horizontal plane, we start at a random state and let the system freely evolve. In all cases, we use the true environment dynamics.

The simplest method to compute the freefall MLE is to evolve two trajectories starting from nearby initial states for a sufficiently long time, monitor their separation and use the Lyapunov exponent definition. A more efficient and reliable method is to use the so-called variational equation (Skokos, 2010), which linearizes the dynamics to explicitly compute how separation vectors evolve with time. Let the true dynamics in the absence of control inputs be described by $\dot{\mathbf{s}} = \mathbf{f}(\mathbf{s})$. Then, the variational equation is given by $\dot{\mathbf{u}} = \frac{d\mathbf{f}}{d\mathbf{s}}\mathbf{u}$. We follow the procedure outlined in Skokos, 2010 to compute the freefall MLE. We describe the procedure here. We consider a random initial separation vector of unit length. We then jointly integrate the system dynamics along with the variational equation to compute how the separation vector evolves with time. One issue is that, the variational equation was derived by linearizing the dynamics, hence it is valid only for small separations. However, in practice, the separation vector can grow large. Hence, we renormalize the separation vector periodically, without changing its direction. Each time, just before renormalizing, we use Equation **??** to compute a fresh estimate of the freefall MLE and update its running average. We continue this process for a sufficiently long time, until the running average converges. We record the result in Table 1. We find that Reacher, Pendulum and Cartpole have a relatively small, positive, freefall MLE, implying low inherent sensitivity to initial conditions. Whereas, Cart-2-pole, Acrobot, Cart-3-pole and Acro-3-bot have a relatively large, positive, freefall MLE, implying high inherent sensitivity to initial conditions.

We turn our focus back to computing the MLE under a trained RL policy, as this is our actual use case. We now compute the finite-time MLE, i. e., the maximal growth rate over finite time. We consider two finite time periods, (i) "Transient" : Here, we consider the period from the start of an episode, till the agent reaches the goal state, which typically takes several hundred time steps, and (ii)

"Short-term" : Here, we consider a period of 16 time steps, which is the imagination horizon used in our model-based RL experiments.

Transient MLE : We compute the finite-time MLE of the true environment dynamics over the transient period, under a trained RL policy (MBRL-DNN / MBRL-LNN / SAC). We refer to this as the transient MLE. The transient MLE is a measure of the system's sensitivity to initial conditions and predictability, over the transient period, under the given policy. It is a measure of the stability of the system's transient dynamics, under the given policy. We follow the same procedure we used to compute the freefall MLE, with some minor changes. One change is that, we tweak the standard variational equation to accomodate a control policy. Let the true dynamics in the presence of control inputs be described by $\dot{\mathbf{s}} = \mathbf{f}(\mathbf{s}, \mathbf{a})$, where $\mathbf{a} = \boldsymbol{\pi}(\mathbf{s})$. Now, the variational equation is given by, $\dot{\mathbf{u}} = \frac{d\,\mathbf{f}}{d\,\mathbf{s}}\,\mathbf{u} = \left(\frac{\partial\,\mathbf{f}}{\partial\,\mathbf{s}} + \frac{\partial\,\mathbf{f}}{\partial\,\mathbf{a}}\,\frac{d\,\mathbf{a}}{d\,\mathbf{s}}\right)\mathbf{u}$. The other change is that, we start from the default initial state and stop the MLE computation once the agent reaches the goal state. We compute the transient MLE of the true environment dynamics, under trained policies of MBRL-DNN, MBRL-LNN and SAC, and average the results. We record the average transient MLE in Table 1.

We find that Reacher, Pendulum and Cartpole have a relatively small, negative, transient MLE. This implies that they have relatively stable transient dynamics, i. e., they are less sensitive to initial conditions and more predictable, over the transient period. Whereas, Cart-2-pole, Acrobot, Cart-3-pole and Acro-3-bot have a relatively large, positive, transient MLE. This implies that they have relatively unstable transient dynamics, i. e., they are more sensitive to initial conditions and less predictable, over the transient period. These systems exhibit unstable transient dynamics due to their high inherent sensitivity to initial conditions and also due to underactuation, which leads to relatively unrestrained motion.

Short-term MLE : In model-based RL, we generate imaginary trajectories that predict the near future. Hence, the short-term predictability becomes important. We compute the finite-time MLE of the learnt dynamics model (MBRL-DNN / MBRL-LNN) over a period of 16 time steps, while following the corresponding policy. We refer to this as the short-term MLE. The short-term MLE along with the dynamics error determine the short-term predictability. We adapt the procedure outlined in Skokos, 2010 to compute the short-term MLE. We describe the procedure here. Let the learnt dynamics model be described by $\mathbf{s}_t = \mathbf{F}(\mathbf{s}_{t-1}, \mathbf{a}_{t-1})$, where $\mathbf{a}_{t-1} = \boldsymbol{\pi}(\mathbf{s}_{t-1})$. How $\mathbf{s}_t$ will change for a small change in $\mathbf{s}_0$ is given by $\frac{d\,\mathbf{s}_t}{d\,\mathbf{s}_0} = \frac{d\,\mathbf{s}_t}{d\,\mathbf{s}_{t-1}}\,\frac{d\,\mathbf{s}_{t-1}}{d\,\mathbf{s}_{t-2}} \cdots \frac{d\,\mathbf{s}_1}{d\,\mathbf{s}_0} = \mathbf{J}_t$. We define the short-term MLE as $\frac{1}{T}\sum_{t=1}^{t=T} \frac{1}{t\,\Delta t} \log \frac{\|\mathbf{J}_t\,\mathbf{u}\|}{\|\mathbf{u}\|}$. Here, $T = 16$ is the length of the trajectory, Δt is the sampling time and $\mathbf{u}$ is a random vector.

We generate 10,000 imaginary trajectories of length 16 time steps and compute the average short-term MLE of MBRL-DNN and MBRL-LNN. We record the results in Table 1. We find that in Reacher, Pendulum and Cartpole, MBRL-DNN has a relatively small short-term MLE, implying numerical errors will accumulate slowly in the short-term. Cart-2-pole is a borderline case. Here, MBRL-DNN has a moderate short-term MLE, implying numerical errors will accumulate moderately in the short-term. In Acrobot, Cart-3-pole and Acro-3-bot, MBRL-DNN has a relatively large short-term MLE, implying numerical errors will accumulate fast even in the short-term. In contrast, MBRL-LNN has a relatively small short-term MLE across environments, implying numerical errors will accumulate slowly in the short-term always.

5.3 Imaginary Trajectories

Next, we assess the quality of the imaginary trajectories produced by MBRL-DNN and MBRL-LNN. We use the 10,000 imaginary trajectories of length 16 time steps we generated earlier to compute the short-term MLE. We generate the corresponding ground truth trajectories.

First, we assess the accuracy of the imaginary trajectories. For each imaginary trajectory, we compute the state error (L1 error between the predicted state and the true state) at each time step. We also compute the trajectory error, which is the sum of the state errors along the trajectory. We plot the average state error as a function of time in Figure 4. We record the average trajectory error in Table 1. We find that, MBRL-LNN has a low trajectory error across environments, while MBRL-DNN's trajectory error varies. In Reacher, Pendulum and Cartpole, MBRL-DNN has a low trajectory error.

In Cart-2-pole, MBRL-DNN has a moderate trajectory error. In Acrobot, Cart-3-pole and Acro-3-bot, MBRL-DNN has a high trajectory error. Across environments, MBRL-LNN has lower trajectory error than MBRL-DNN, i. e, MBRL-LNN's imaginary trajectories are more accurate.

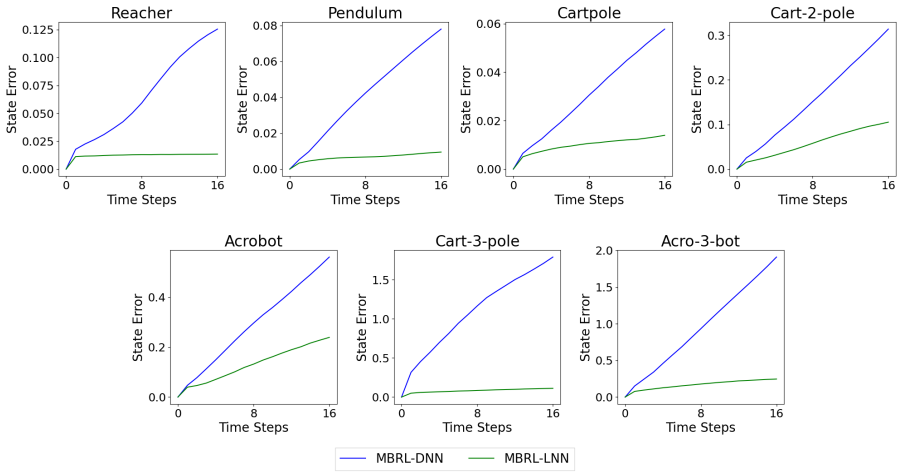


Figure 4: Accuracy of imaginary trajectories : State Error vs Time.

Next, we assess how well the imaginary trajectories respect the underlying physical laws, such as conservation of energy. In our environments, the work done by external non-conservative forces, i. e, the actuators, is non-zero. Hence, the total energy of the system will change. The change in total energy must equal the work done by external non-conservative forces, for energy to be conserved. We define the energy error as the L1 error between the change in total energy and the work done by external non-conservative forces. For each imaginary trajectory, we compute the energy error at each time step. We plot the average energy error as a function of time in Figure 5. We find that, across environments, MBRL-LNN has lower energy error than MBRL-DNN, i. e, MBRL-LNN's imaginary trajectories conserve energy better.

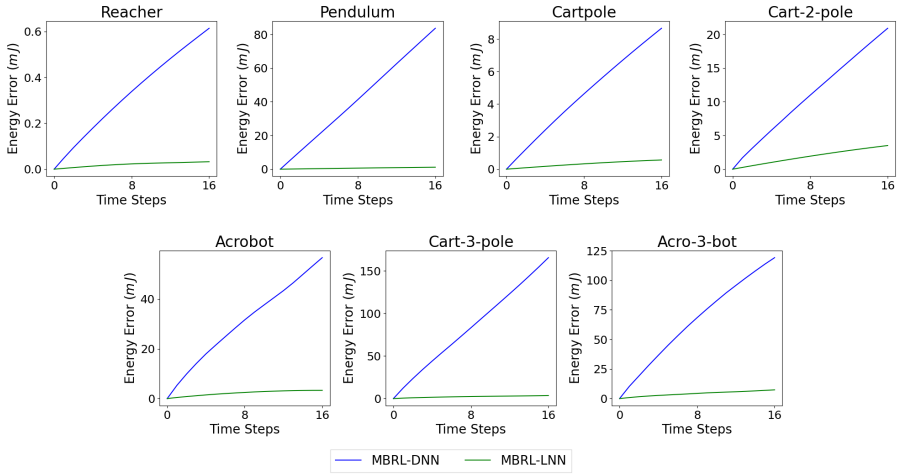


Figure 5: Conservation of energy in imaginary trajectories : Energy Error vs Time.

5.4 Discussion

Underactuation and high inherent sensitivity to initial conditions, characterized by a large freefall MLE, are the main factors that make Cart-2-pole, Acrobot, Cart-3-pole and Acro-3-bot challenging. Underactuation, in itself, makes it harder to learn successful policies, hence we need more samples. Underactuation also leads to relatively unrestrained motion. This condition combined with high

inherent sensitivity to initial conditions leads to unstable transient dynamics, which is characterized by a large transient MLE. Unstable transient dynamics leads to sensitivity to initial conditions over the transient period, which leads to high variance in the agent trajectories. Unstable transient dynamics also leads to poor predictability over the transient period, which affects critic learning, as it is concerned with estimating the future return, and hence also affects actor learning, as our methods follow an actor-critic approach. Under these conditions, we need a lot of samples to learn. Unstable transient dynamics also makes it challenging for standard deep neural networks to learn accurate dynamics models.

SAC is successful in all four of these environments. However, it is unable to overcome the issue of a lot of samples being needed to learn. MBRL-DNN and MBRL-LNN overcome this issue by generating a lot of imaginary data, which allows them to perform many more policy updates. This is the main reason why SAC achieves lower average-return than MBRL-LNN in these environments.

In Acrobot, Cart-3-pole and Acro-3-bot, due to unstable transient dynamics, MBRL-DNN has a large dynamics error and short-term MLE, due to which its imaginary trajectories have high trajectory error. Hence, it is unable to learn good policies and is unsuccessful in these environments. Cart-2-pole is a borderline case. Here, MBRL-DNN achieves a moderate dynamics error and short-term MLE, despite the unstable transient dynamics. Hence its imaginary trajectories only have moderate trajectory error and as a result, it is able to learn a reasonably good policy and is successful.

In all four of these environments, MBRL-LNN learns accurate dynamics models despite the unstable transient dynamics due to its inductive biases. Hence, it has a small dynamics error and short-term MLE, due to which its imaginary trajectories have low trajectory error. Hence, it is able to learn good policies and is successful.

Thus, MBRL-LNN achieves better average-return and sample efficiency than both MBRL-DNN and SAC in these environments, because it is able to generate a lot of high quality imaginary data and perform many policy updates using it.

6 Conclusion

We apply model-based RL to robotic systems undergoing rigid body motion. We utilize the structure of rigid body dynamics to learn a Lagrangian Neural Network and use it to train our model-based RL algorithm. While it is intuitive that such physics-informed dynamics models will improve model-based RL performance, it is not apparent in which environments we will significantly benefit from such an approach. We show that, in environments with underactuation and high inherent sensitivity to initial conditions, characterized by a large maximal Lyapunov exponent under freefall, we are likely to significantly benefit from learning physics-informed dynamics models. In such environments, our Lagrangian model-based RL approach achieves better average-return and sample efficiency than a version of our model-based RL algorithm that uses a standard deep neural network based dynamics model, as well as Soft Actor-Critic, a state-of-the-art model-free RL algorithm.

7 Future Work

In future work, we plan to consider friction, as it is present in most real world robots. As far as learning friction, there are two approaches. In the black box approach, the friction is learnt directly from data using a neural network (Zhong et al., 2020). In the white box approach, an analytic friction model is used and its parameters alone are learnt from data (Lutter et al., 2019b). For e. g, in robotic arms, the actuator friction dominates and it can be modelled as a combination of static friction, viscous friction and stiction, which depend on the generalized velocity $\dot{\mathbf{q}}$ and a few coefficients of friction. These coefficients alone are learnt from data. We plan to pursue the white box approach.

In future work, we also plan to consider contact dynamics, as it is present in many real world robotics tasks like robotic manipulation and legged locomotion. Simulation of contact dynamics is an area of active research, unlike simulation of smooth multi-joint dynamics, which is well established. There are predominantly two approaches to simulate contact dynamics, the linear complementarity approach (LCP) (Stewart and Trinkle, 1996; Anitescu and Potra, 1997) and the convex optimization approach (Todorov, 2011; Todorov et al., 2012; Todorov, 2014). LCP was the standard approach for a long time. Convex optimization improves upon LCP in terms of ease of

computation, while maintaining similar accuracy. We plan to pursue the convex optimization approach and use differentiable convex optimization layers (Agrawal et al., 2019) to learn differentiable contact models which can extend LNNs. There has been some recent work in this direction by Zhong et al., 2021b. This approach however assumes that the geometry of the robot and its surroundings are known and uses them to compute the active contacts and their jacobians. It remains to be seen, whether we can do away with this assumption.

Upon learning a physics-informed dynamics model that captures rigid body dynamics, friction and contact dynamics, we plan to use it to train a model-based RL algorithm.

Appendix

A Environment Details

The task to be accomplished in each environment is as follows,

- Reacher : Control a fully-actuated two-link manipulator in the horizontal plane to reach a fixed target location.
- Pendulum : Swing up and balance a simple pendulum.
- Cartpole : Swing up and balance an unactuated pole by applying forces to a cart at its base.
- Cart-2-pole : One extra pole is added to Cartpole. Again, only the cart is actuated.
- Acrobot : Control a two-link manipulator to swing up and balance. Only the second pole is actuated.
- Cart-3-pole : One extra pole is added to Cart-2-pole. Only the cart and the third pole are actuated.
- Acro-3-bot : One extra pole is added to Acrobot. Only the first and the third poles are actuated.

Environment	Reacher	Pendulum	Cartpole	Cart-2-pole	Acrobot	Cart-3-pole	Acro-3-bot
$\dim(\mathcal{S})$	4	2	4	6	4	8	6
$\dim(\mathcal{A})$	2	1	1	1	1	2	2

Table 2: State space and action space dimensions of the environments.

All actions are in the range $[-1, 1]$. The reward is based on proximity to goal position, magnitude of angular velocity and whether the cart is centered (if applicable). All rewards are in the range $[0, 1]$. There are no terminal states. Each episode consists of 1000 time steps. Hence, the total reward over an episode is in the range $[0, 1000]$.

B Implementation Details

Table 3 lists the main hyperparameters used in our model-based RL experiments.

We use the following network architecture in our model-based RL experiments,

- Critic : $\dim(\mathcal{S}) \rightarrow 256 \rightarrow 256 \rightarrow 1$
- Actor : $\dim(\mathcal{S}) \rightarrow 256 \rightarrow 256 \rightarrow 2 \times \dim(\mathcal{A})$
- DNN : $\dim(\mathcal{S}) + \dim(\mathcal{A}) \rightarrow 64 \rightarrow 64 \rightarrow \dim(\mathcal{S})$

- LNN
  - L : $\dfrac{\dim(\mathcal{S})}{2} \rightarrow 64 \rightarrow 64 \rightarrow \dfrac{\dim(\mathcal{S}) \times (\dim(\mathcal{S}) + 2)}{8}$
  - V : $\dfrac{\dim(\mathcal{S})}{2} \rightarrow 64 \rightarrow 64 \rightarrow 1$
- Reward model : $\dim(\mathcal{S}) \rightarrow 64 \rightarrow 64 \rightarrow 1$

Parameter	Value
Random episodes at start of training (K)	10
Replay buffer size	10^5
Batch size for model learning	64
Model learning batches per episode (N_1)	10^4
Batch size for behaviour learning	64
Behaviour learning batches per episode (N_2)	10^3
Imagination horizon (T)	16
Discount factor (γ)	0.99
Lambda (λ)	0.95
Entropy weightage (η)	10^{-4}
Gradient clipping norm	100
Optimizer	AdamW
Learning rate	3×10^{-4}

Table 3: Model-Based RL Hyperparameters.

For SAC, we use the same hyperparameters and network architecture as the original paper (Haarnoja et al., 2018b).

C Longer-Horizon Imaginary Trajectories

Although during behaviour learning we use an imagination horizon of only 16 time steps, in Figures 6 and 7, we plot the average state error and energy error as a function of time for imaginary trajectories of horizon 100 time steps, to see how far into the future our models can predict accurately.

We find that, across environments, MBRL-LNN is able to accurately predict quite far into the future, while MBRL-DNN is unable to.

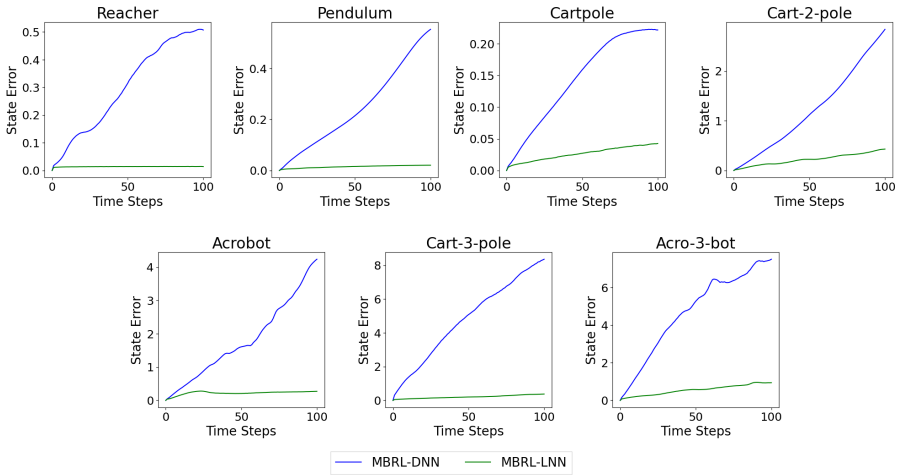


Figure 6: Accuracy of longer-horizon imaginary trajectories : State Error vs Time.

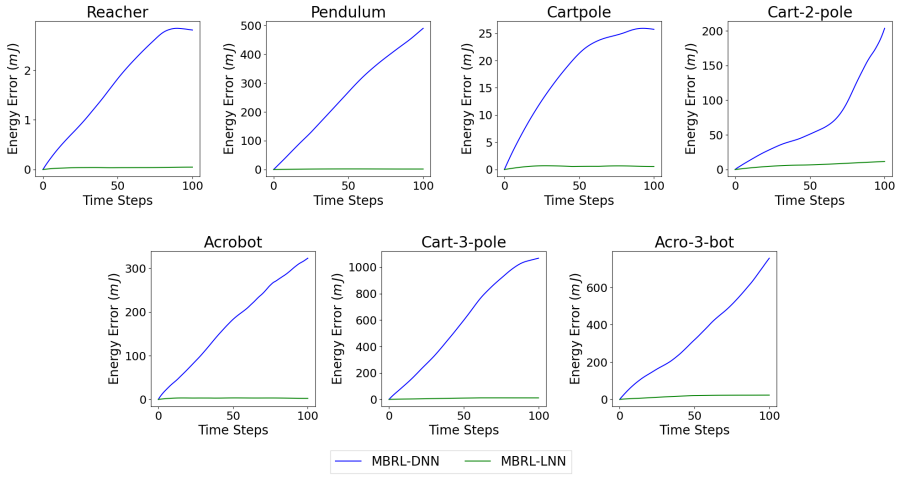


Figure 7: Conservation of energy in longer-horizon imaginary trajectories : Energy Error vs Time.

References

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018a.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b.

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468, 2021.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

M. Lutter, C. Ritter, and J. Peters. Deep lagrangian networks: Using physics as model prior for deep learning. In *International Conference on Learning Representations (ICLR)*, 2019a.

Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. *Advances in neural information processing systems*, 32, 2019.

Michael Lutter, Kim Listmann, and Jan Peters. Deep lagrangian networks for end-to-end learning of energy-based control for under-actuated systems. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7718–7725. IEEE, 2019b.

Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ode-net: Learning hamiltonian dynamics with control. In *International Conference on Learning Representations*, 2019.

Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Dissipative symoden: Encoding hamiltonian dynamics with dissipation and control into deep learning. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

Marc Finzi, Ke Alexander Wang, and Andrew G Wilson. Simplifying hamiltonian and lagrangian neural networks via explicit constraints. *Advances in neural information processing systems*, 33: 13880–13889, 2020.

Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Benchmarking energy-conserving neural networks for learning dynamics from data. In *Learning for Dynamics and Control*, pages 1218–1229. PMLR, 2021a.

Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472. Citeseer, 2011.

Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. *Advances in neural information processing systems*, 28, 2015.

Ignasi Clavera, Violet Fu, and Pieter Abbeel. Model-augmented actor-critic: Backpropagating through paths. *arXiv preprint arXiv:2005.08068*, 2020.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Ch Skokos. The lyapunov characteristic exponents and their computation. In *Dynamics of Small Solar System Bodies and Exoplanets*, pages 63–135. Springer, 2010.

David E Stewart and Jeffrey C Trinkle. An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction. *International Journal for Numerical Methods in Engineering*, 39(15):2673–2691, 1996.

Mihai Anitescu and Florian A Potra. Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. *Nonlinear Dynamics*, 14(3):231–247, 1997.

Emanuel Todorov. A convex, smooth and invertible contact model for trajectory optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 1071–1076. IEEE, 2011.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

Emanuel Todorov. Convex and analytically-invertible dynamics with contacts and constraints: Theory and implementation in mujoco. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6054–6061. IEEE, 2014.

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019.

Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Extending lagrangian and hamiltonian neural networks with differentiable contact models. *Advances in Neural Information Processing Systems*, 34:21910–21922, 2021b.