# Lagrangian Model Based Reinforcement Learning

**Anonymous Author(s)**
Affiliation
Address
email

## Abstract

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 of RL algorithms is model-based RL. Here we learn a model of the environment, essentially its transition dynamics and reward function, and use it to generate imaginary trajectories, which we then use to update the policy. Intuitively, learning better environment models should improve model-based RL. Recently there has been growing interest in developing better deep neural network based dynamics models for physical systems through better inductive biases. We investigate if such physics-informed dynamics models can also improve model-based RL. We focus on robotic systems undergoing rigid body motion. We utilize the structure of rigid body dynamics to learn Lagrangian neural networks and use them within a model-based RL algorithm. We find that our Lagrangian model-based RL approach achieves better average-return and sample efficiency compared to standard model-based RL as well as state-of-the-art model-free RL algorithms such as Soft-Actor-Critic, in complex environments.

## 1 Introduction

Reinforcement Learning (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 problems 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 to train in simulation. However, in many robotics tasks, developing realistic simulations is hard. Another solution is to improve the sample efficiency of RL algorithms so that we can learn using data from the actual robot. One approach to improve sample efficiency is model-based RL. Here we learn a model of the environment, essentially its transition dynamics and reward function, and use it to generate imaginary trajectories, which we then use to update the policy. Intuitively, learning better environment models should improve model-based RL.

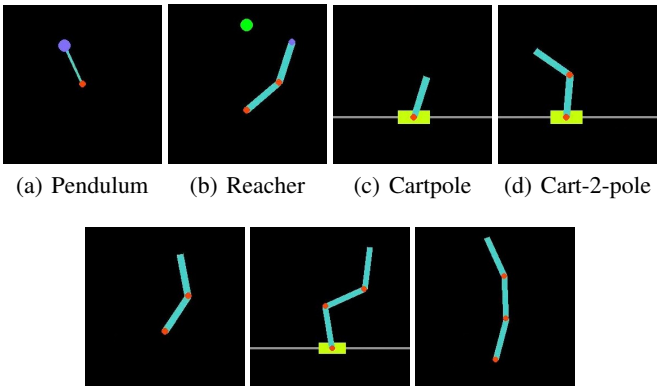
Recently there has been growing interest in developing better deep neural network based dynamics models for physical systems through better inductive biases (Lutter et al., 2019; Greydanus et al., 2019; Zhong et al., 2019, 2020; Cranmer et al., 2020; Finzi et al., 2020; Zhong et al., 2021b). These physics-informed models learn the dynamics more accurately compared to standard deep neural networks. They also obey the underlying physical laws, such as conservation of energy better.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

Previous studies however, have mostly focused on improving dynamics learning. We investigate if such physics-informed dynamics models can also improve model-based RL.

## 2 Environments Considered

We focus on robotic systems undergoing rigid body motion. We assume that there are no contacts or dissipative effects such as friction and damping. The environments considered are shown in Figure 1.



(a) Pendulum (b) Reacher (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 without contacts or dissipation.

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

- Pendulum : Swing up and balance a simple pendulum.
- Reacher : Control a fully-actuated two-link manipulator in the horizontal plane to reach a fixed target location.
- 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. The cart and the third pole are actuated.
- Acro-3-bot : One extra pole is added to Acrobot. The first and the third poles are actuated.

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 actions are the motor torques $\boldsymbol{\tau}$ to be applied at different joints. All actions are in the range $[-1, 1]$. The reward is based on proximity to goal position, whether the cart is centered (if applicable) and another term which penalizes large angular velocities. All rewards are in the range $[0, 1]$. There are no terminal states. All episodes are $1000$ time-steps each. Hence, the total reward over an episode is in the range $[0, 1000]$. We develop our own simulations from first principles.

## 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 then 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, i. e., we 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 fully connected deep neural network. We refer to this approach as DNN, short for deep neural network. 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 Lutter et al., 2019; Cranmer et al., 2020; Finzi et al., 2020; Zhong et al., 2021b. 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,

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

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,

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

Thus,

$$\frac{d}{dt}\begin{bmatrix}\mathbf{q}\\ \dot{\mathbf{q}}\end{bmatrix} = \begin{bmatrix}\dot{\mathbf{q}}\\ \mathbf{M}^{-1}(\mathbf{q})\left(\boldsymbol{\tau} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}} - \mathbf{G}(\mathbf{q})\right)\end{bmatrix} \tag{4}$$

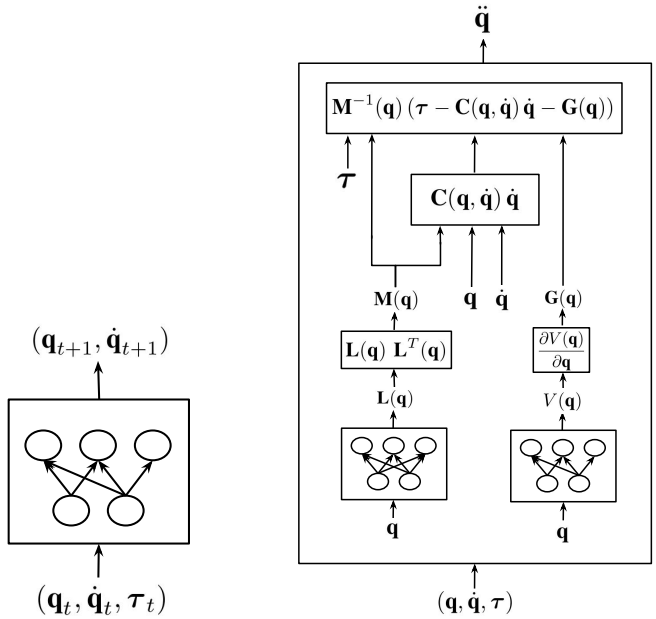
We use a fully connected network to learn the potential energy function $\mathcal{V}(\mathbf{q})$ and another fully connected network to learn a lower triangular matrix $\mathbf{L}(\mathbf{q})$, which we use to compute the mass matrix as $\mathbf{M}(\mathbf{q}) = \mathbf{L}(\mathbf{q})\,\mathbf{L}^T(\mathbf{q})$. We then compute $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\,\dot{\mathbf{q}}$ and $\mathbf{G}(\mathbf{q})$ as shown in Equation 2. We then numerically integrate Equation 4 using second-order Runge-Kutta to compute the next state $(\mathbf{q}_{t+1}, \dot{\mathbf{q}}_{t+1})$. 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, the loss used for training is the L1 error between the predicted next state and the ground truth.

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 fully connected network to map the next state to the reward. The loss used for training is the L1 error between the predicted reward and the ground truth.

### 3.2 Behaviour Learning

In the behaviour learning step, we use the learned model to generate imaginary trajectories and then use them to update the policy. Some model-based RL approaches use the model just to generate additional data. To update the policy, they use a regular model-free RL 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), MAAC (Clavera et al., 2020) and Dreamer (Hafner et al., 2019, 2020) adopt this approach. We choose 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 discounted sum of future rewards from a given state. We train the critic to regress the $\lambda$-return (Sutton and Barto, 2018; Schulman et al., 2015). We stabilize the critic




(a) DNN (b) LNN

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

training by computing the $\lambda$-return using a target network that is updated every 100 updates. The critic loss function is given by,

$$L(w) = \mathbb{E}\,[\sum_{t=0}^{T-1} \frac{1}{2}(\,V'_\lambda(s_t) - V(s_t)\,)^2] \tag{5}$$

where

$$V'_\lambda(s_t) = \begin{cases} r_t + \gamma\,(\,(1-\lambda)V'(s_{t+1}) + \lambda V'_\lambda(s_{t+1})\,) & \text{if } t < T \\ V'(s_t) & \text{if } t = T \end{cases} \tag{6}$$

We use a stochastic actor. To backpropagate through sampled actions, we use the reparameterization trick (Kingma and Welling, 2013). The actor network outputs the mean $\mu$ and standard deviation $\sigma$ of a Gaussian distribution, which we combine with a random noise $\epsilon \sim \mathcal{N}(0, \mathbb{I})$ and transform using a tanh to obtain the action,

$$a_t = \tanh(\mu_\theta(s_t) + \sigma_\theta(s_t) \cdot \epsilon) \tag{7}$$

The actor aims to output actions that lead to states that maximize the expected future reward. We train the actor to maximize the same $\lambda$-return that was computed to train the critic. We add an entropy term to the actor objective to encourage exploration. The actor loss function is given by,

$$L(\theta) = -\,\mathbb{E}\,[\sum_{t=0}^{T-1}[\,V'_\lambda(s_t) - \eta \log \pi(a_t|s_t)\,]] \tag{8}$$

We summarize our overall model-based RL algorithm in Algorithm 1.

### 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

**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, V' ← V.
    end for
    // Environment Interaction
    Interact with environment for one episode using current policy.
    Add transitions to replay buffer.
end for
```

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 MBRL-DNN, MBRL-LNN and SAC 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 loss compared to MBRL-DNN. The reward loss for both methods are similar.

In Pendulum, Reacher and Cartpole, all the methods, i. e., MBRL-DNN, MBRL-LNN and SAC are able to successfully solve the task. In terms of average-return, MBRL-DNN $\approx$ MBRL-LNN $\approx$ SAC.

In Cart-2-pole, again, all the methods are able to 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 are able to 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 compared to 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. Pendulum, Reacher 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 complex environments where there is a noticeable difference in the performance of different methods. We try to understand what makes these latter environments more complex.

## 5.1 Environment Dynamics

As a first step, we try to characterize the underlying dynamics of each environment by calculating their Lyapunov exponents. Lyapunov exponents measure the exponential 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

| Environment | Method | Steps | Dynamics Loss | Reward Loss | Average Return | Solved (Y/N) | Global MLE | Local MLE | Trajectory Error |
|---|---|---|---|---|---|---|---|---|---|
| Pendulum | MBRL-DNN | 0.25 M | 108.93e-5 | **8.198e-4** | **787.3** | **Y** | 0.1287 | 1.4481 | 0.6933 |
| | MBRL-LNN | **0.125 M** | **3.4559e-5** | 8.3962e-4 | 786.3 | **Y** | | **0.4043** | **0.1099** |
| | SAC | 1.25 M | – | – | 786.4 | **Y** | | – | – |
| Reacher | MBRL-DNN | 0.25 M | 126.57e-5 | 7.2217e-4 | 942.7 | **Y** | 0.0241 | -1.4003 | 1.0984 |
| | MBRL-LNN | **0.125 M** | **4.7557e-5** | **8.4078e-4** | **943.4** | **Y** | | **-2.117** | **0.2036** |
| | SAC | 1.25 M | – | – | 943.2 | **Y** | | – | – |
| Cartpole | MBRL-DNN | 0.25 M | 149.31e-5 | **7.5226e-4** | 899.4 | **Y** | 0.2688 | 1.8693 | 0.5141 |
| | MBRL-LNN | **0.125 M** | **6.014e-5** | 8.6215e-4 | **900.0** | **Y** | | **1.1955** | **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** | 1.3122 | 4.9786 | 2.6164 |
| | MBRL-LNN | **0.5 M** | **70.389e-5** | **8.8135e-4** | **828.3** | **Y** | | **1.6674** | **0.9744** |
| | SAC | 10 M | – | – | 801.7 | **Y** | | – | – |
| Acrobot | MBRL-DNN | 2 M | 799.90e-5 | 8.848e-4 | 764.5 | N | 1.1800 | 10.8583 | 4.8975 |
| | MBRL-LNN | **0.5 M** | **48.046e-5** | **7.9032e-4** | **911.8** | **Y** | | **2.0932** | **2.2121** |
| | SAC | 10 M | – | – | 817.8 | **Y** | | – | – |
| Cart-3-pole | MBRL-DNN | 2 M | 1196.0e-5 | 17.044e-4 | 757.0 | N | 3.1164 | 8.3368 | 18.2503 |
| | MBRL-LNN | **0.5 M** | **61.3e-5** | **6.5141e-4** | **916.6** | **Y** | | **3.5838** | **1.3601** |
| | SAC | 10 M | – | – | 759.4 | **Y** | | – | – |
| Acro-3-bot | MBRL-DNN | 2 M | 1322.0e-5 | 16.6256e-4 | 739.7 | N | 3.1240 | 11.1282 | 16.0338 |
| | MBRL-LNN | **0.5 M** | **156.34e-5** | **10.308e-4** | **929.4** | **Y** | | **3.6980** | **2.8153** |
| | SAC | 10 M | – | – | 853.5 | **Y** | | – | – |

Table 1: Overall results from MBRL-DNN, MBRL-LNN and SAC experiments.

time $t$ is $\mathbf{u}_t$, then the Lyapunov exponent is defined as,

$$\lambda = \frac{1}{t} \log \frac{\|\mathbf{u}_t\|}{\|\mathbf{u}_0\|} \tag{9}$$

In general, there is a spectrum of Lyapunov exponents – equal in number to the dimensionality of the state space. It is common to refer to the largest Lyapunov exponent as the maximal Lyapunov exponent (MLE). The MLE represents how quickly numerical errors will accumulate in a system. Thus, it is a measure of predictability and non-linearity.

Global Lyapunov exponents are concerned with long-term growth rates, i. e., $t \to \infty$. Our trained policies (MBRL-DNN / MBRL-LNN / SAC) are not suitable for computing Global Lyapunov exponents because, the separation between two trajectories starting from nearby initial states will become negligible in some time. So, we execute random actions. Our intention here is to understand the long-term predictability and non-linearity of the different environments. We estimate the global Lyapunov exponents by evolving a set of linearly independent perturbations for a sufficiently long time, while executing random actions. One issue that arises is that, over time, all the perturbations tend to align with the direction of maximal expansion. To overcome this, the perturbations are periodically orthonormalized using the Gram-Schmidt procedure. We record the global MLE in Table 1. We find that Cart-2-pole, Acrobot, Cart-3-pole and Acro-3-bot have a relatively large global MLE. This implies that they are quite non-linear and unpredictable over longer time scales. In contrast, Pendulum, Reacher and Cartpole have a relatively small global MLE. This implies that they are less non-linear and more predictable over longer time scales.

From a model-based RL perspective, we are also interested in the short-term predictability. The local MLE measures the short-term predictability around a particular state. We describe how it is computed. Let the dynamics model (DNN / LNN) be described by $\mathbf{s}_t = \mathbf{g}(\mathbf{s}_{t-1}, \mathbf{a}_{t-1})$. As $\mathbf{a}_{t-1} = \pi(\mathbf{s}_{t-1})$, we can simply denote the dynamics model by $\mathbf{s}_t = \mathbf{f}(\mathbf{s}_{t-1})$, for some $\mathbf{f}$. The jacobian matrix $\mathbf{J}_{t,t-1} = \dfrac{\partial \mathbf{s}_t}{\partial \mathbf{s}_{t-1}}$ measures how $\mathbf{s}_t$ will change for a small change in $\mathbf{s}_{t-1}$. Using the

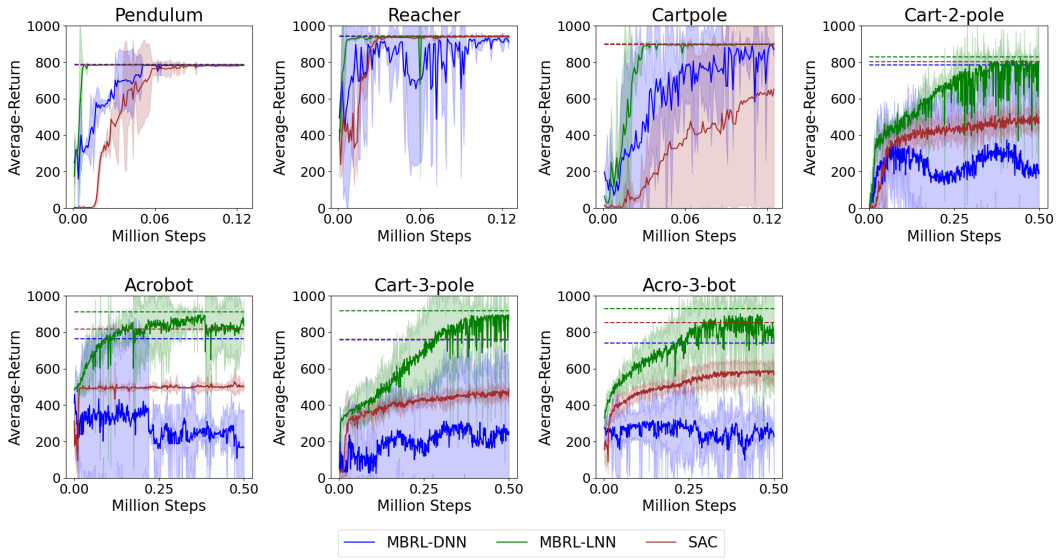



Figure 3: Training curves for MBRL-DNN, MBRL-LNN and SAC experiments. The dashed lines indicate the best average-return after convergence.

chain rule, we can extend this to compute how $\mathbf{s}_t$ will change for a small change in $\mathbf{s}_0$,

$$\mathbf{J}_{t,0} = \frac{\partial \mathbf{s}_t}{\partial \mathbf{s}_0} = \frac{\partial \mathbf{s}_t}{\partial \mathbf{s}_{t-1}} \frac{\partial \mathbf{s}_{t-1}}{\partial \mathbf{s}_{t-2}} \cdots \frac{\partial \mathbf{s}_1}{\partial \mathbf{s}_0} = \mathbf{J}_{t,t-1}\, \mathbf{J}_{t-1,t-2} \cdots \mathbf{J}_{1,0} \tag{10}$$

The local MLE is given by,

$$\text{local MLE} = \frac{1}{T} \sum_{t=1}^{t=T} \left( \frac{\log \| \mathbf{J}_{t,0}\, \mathbf{u}_t \|}{t\, \Delta t} \right) \tag{11}$$

Here $\Delta t$ is the sampling time, $T = 16$ is the length of the trajectory and $\mathbf{u}_t$ is a random vector. We generate 10,000 imaginary trajectories of length 16 time steps and compute the average local MLE. We do this for both MBRL-DNN and MBRL-LNN. In each case, we use the corresponding policy and model to generate the trajectories. We record the results in Table 1. We find that in Pendulum, Reacher, Cartpole and Cart-2-pole, both MBRL-DNN as well as MBRL-LNN have a relatively small local MLE. This implies that in these environments, numerical errors will accumulate slowly for both the methods. In Acrobot, Cart-3-pole and Acro-3-bot, MBRL-DNN has a relatively large local MLE while MBRL-LNN has a relatively small local MLE. This implies that in these environments, numerical errors will accumulate fast for MBRL-DNN and slow for MBRL-LNN .

### 5.2 Actuation

Pendulum and Reacher are fully actuated. The rest of the environments are underactuated.

### 5.3 Deviation of Imaginary Trajectories from Ground Truth

We compare the imaginary trajectories produced by MBRL-DNN and MBRL-LNN with the ground truth. We generate 10,000 imaginary trajectories of length 16 time steps. For each trajectory, we compute the state error and the energy error at each time step, and also the overall trajectory error. The state error is simply the L1 error between the predicted state and the true state. We define the energy error as the change in total energy minus work done by external non-conservative forces. For energy to be conserved, the energy error must be zero. The trajectory error is the sum of the state errors along the trajectory.

We record the average trajectory error in Table 1. We plot the average state error and energy error as a function of time in Figure 4. We find that across environments, MBRL-LNN's imaginary trajectories have lower trajectory error and also conserve energy much better.

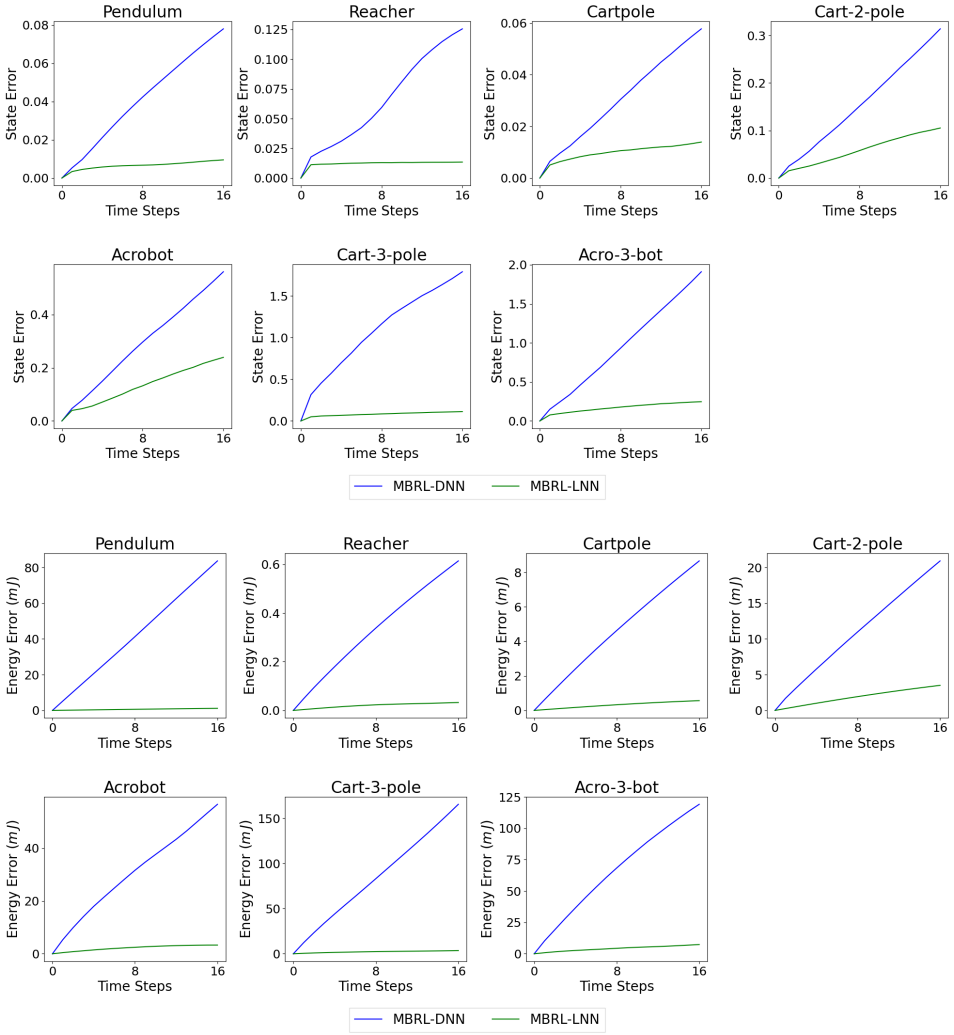



Figure 4: Deviation of imaginary trajectories from ground truth.

In environments like Pendulum, Reacher, Cartpole and Cart2pole, MBRL-DNN's imaginary trajectories have low trajectory error (0.6933, 1.0984, 0.5141 and 2.6164 respectively) and MBRL-LNN's imaginary trajectories also have low trajectory error (0.1099, 0.2036, 0.1645 and 0.9744 respectively). In environments like Acrobot, Cart-3-pole and Acro-3-bot, MBRL-DNN's imaginary trajectories have high trajectory error (4.8975, 18.2503 and 16.0338 respectively), whereas MBRL-LNN's imaginary trajectories still have low trajectory error (2.2121, 1.3601 and 2.8153 respectively).

The reason for this is as follows. In environments like Pendulum, Reacher, Cartpole and Cart-2-pole, MBRL-DNN has a relatively low dynamics error as well as local MLE. Hence, MBRL-DNN's imaginary trajectories have low trajectory error. In environments like Acrobot, Acro-3-bot and Cart-3-pole, MBRL-DNN has a relatively large dynamics error as well as local MLE. Hence, MBRL-DNN's imaginary trajectories have high trajectory error. In contrast, MBRL-LNN has a low dynamics error and local MLE across environments. Hence, MBRL-LNN's imaginary trajectories have low trajectory error across environments.

### 5.4 Conclusion

Thus, highly non-linear dynamics, poor predictability (short-term and/or long-term) and underactuation are some of the factors that make Cart-2-pole, Acrobot, Acro-3-bot and Cart-3-pole complex

environments. In these complex environments, MBRL-LNN outperforms both MBRL-DNN as well as SAC in terms of average-return and sample efficiency.

## 6 Future Work

In future work, we plan to model dissipative effects such as friction and damping. There exist white box as well as black box approaches to learning dissipation. In the white box approach, we assume a particular functional form for the dissipative forces and learn just the parameters. In the black box approach, we learn the dissipative forces directly from data using a neural network.

Our current work does not model contact dynamics. However, many real world robotic tasks like manipulation and locomotion involve contacts. In future work, we plan to extend LNNs with differentiable contact models. Unlike simulation of smooth multi-joint dynamics, which is well established, simulation of contact dynamics is an area of active research. 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 speed. In future work, we plan to pursue the latter approach and use differentiable convex optimization layers (Agrawal et al., 2019) to develop differentiable contact models, which can extend LNNs. There has been some recent work in this direction by Zhong et al., 2021a.

## 7 Summary

We demonstrate the benefits of using physics-informed dynamics models in the model-based RL setting. We focus on robotic systems undergoing rigid body motion. We utilize the structure of rigid body dynamics to learn Lagrangian neural networks and use them within a model-based RL algorithm. We find that our Lagrangian model-based RL approach achieves better average-return and sample efficiency compared to standard model-based RL as well as state-of-the-art model-free RL algorithms such as Soft-Actor-Critic, in complex environments.

## References


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.

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.

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.

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

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.

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.

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.

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

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.

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.

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.

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.

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.

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

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.

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)*, 2019.

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.

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

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.

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

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

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. 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.

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.

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.

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, 2021a.

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, 2021b.