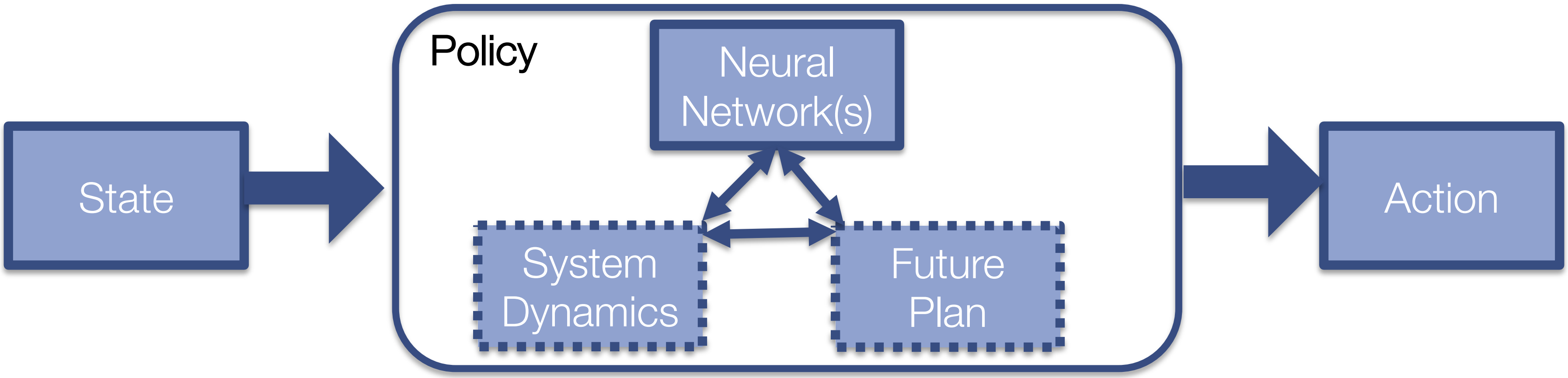
NeurIPS 2018

Differentiable MPC for End-to-End Planning and Control

Brandon Amos[1]
Ivan Dario Jimenez Rodriguez[2]
Jacob Sacks[2]
Byron Boots[2]
J. Zico Kolter[13]

[1]Carnegie Mellon University
[2]Georgia Tech
[3]Bosch Center for AI

Should RL policies have a system dynamics model or not?



Model-free RL

More general, doesn't make as many assumptions about the world

Rife with poor data efficiency and learning stability issues

Model-based RL (or control)

A useful prior on the world if it lies within your set of assumptions

Combining model-based and model-free RL

Recently there has been a lot of interest in model-based priors for model-free reinforcement learning:

Among others: Dyna-Q (Sutton, 1990), GPS (Levine and Koltun, 2013), Imagination-Augmented Agents (Weber et al., 2017), Value Iteration Networks (Tamar et al., 2016), TreeQN (Farquhar et al., 2017)

These typically involve:

1. **Using an RNN:** Efficient but not as expressive and general as MPC/iLQR
2. **Unrolling an LQR or gradient-based solver:** Expressive/general but inefficient

Our approach: Differentiable Model-Predictive Control

- **Explicitly** solves a control problem

Our Approach: Model Predictive Control

Traditionally viewed as a pure **planning problem** given known (potentially non-convex) **cost** and **dynamics**:

$$\begin{aligned} \tau_{1:T}^{\star} = \operatorname*{argmin}_{\tau_{1:T}} \; & \sum_t C_\theta(\tau_t) \quad \text{Cost} \\ \text{subject to } \; & x_1 = x_{init} \\ & x_{t+1} = f_\theta(\tau_t) \quad \text{Dynamics} \\ & \underline{u} \leq u \leq \overline{u} \end{aligned}$$

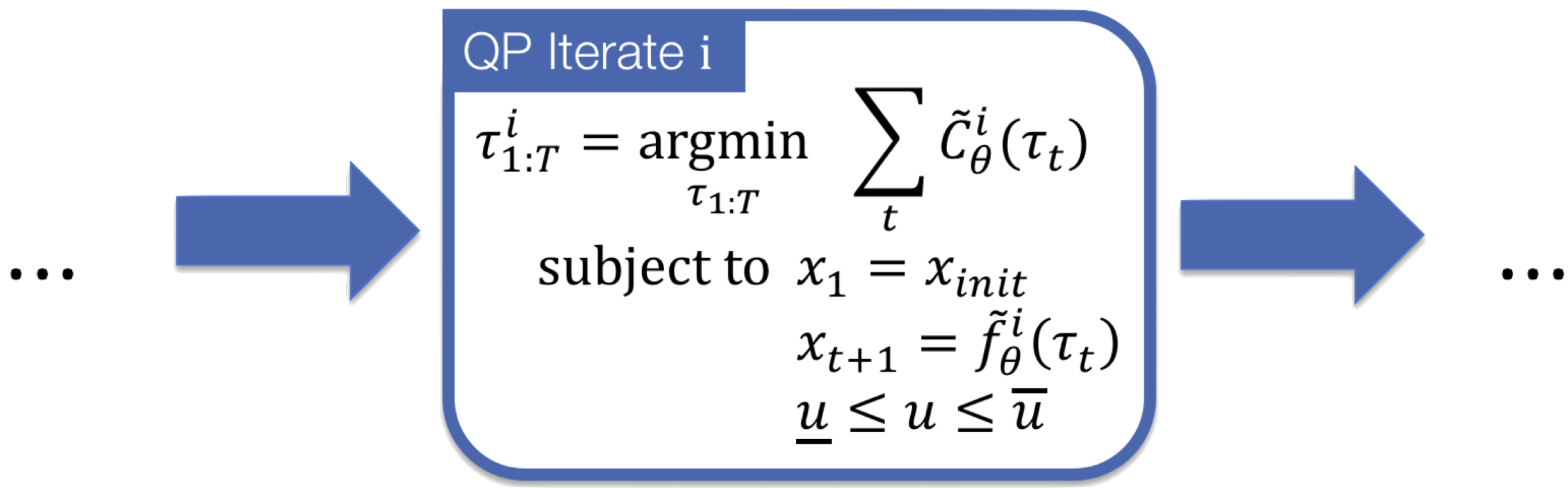
where $\tau_t = \{x_t, u_t\}$

Execute u_1 in the environment, observe the next observation, and repeat.

Cost and dynamics explicitly represented and learned.

Model Predictive Control with SQP

- The standard way of solving MPC is to use **sequential quadratic programming (SQP),** using LQR in most cases
- **Form approximations** to the cost and dynamics around the current iterate
- Repeat until a **fixed point** is reached and **differentiate through it**



LQR, KKT Systems, and Differentiation

Solving **LQR** with **dynamic Riccati recursion** efficiently solves the **KKT system**

$$
\overbrace{\begin{array}{c|cc|cc|c}
 & \tau_t & \lambda_t & \tau_{t+1} & \lambda_{t+1} & \\
\ddots & & & & & \\
\hline
 & C_t & F_t^\top & & & \\
 & F_t & & [-I \quad 0] & & \\
\hline
 & & \begin{bmatrix} -I \\ 0 \end{bmatrix} & C_{t+1} & F_{t+1}^\top & \\
 & & & F_{t+1} & & \\
\hline
 & & & & & \ddots
\end{array}}^{K}
\begin{bmatrix} \vdots \\ \tau_t^\star \\ \lambda_t^\star \\ \tau_{t+1}^\star \\ \lambda_{t+1}^\star \\ \vdots \end{bmatrix}
= -
\begin{bmatrix} \vdots \\ c_t \\ f_t \\ c_{t+1} \\ f_{t+1} \\ \vdots \end{bmatrix}
$$

Backwards Pass: Use the OptNet approach from [Amos and Kolter, 2017] to implicitly **differentiate** the LQR KKT conditions:

$$
\frac{\partial \ell}{\partial C_t} = \frac{1}{2}\left(d_{\tau_t}^\star \otimes \tau_t^\star + \tau_t^\star \otimes d_{\tau_t}^\star\right) \qquad \frac{\partial \ell}{\partial c_t} = d_{\tau_t}^\star \qquad \frac{\partial \ell}{\partial x_{\text{init}}} = d_{\lambda_0}^\star
$$

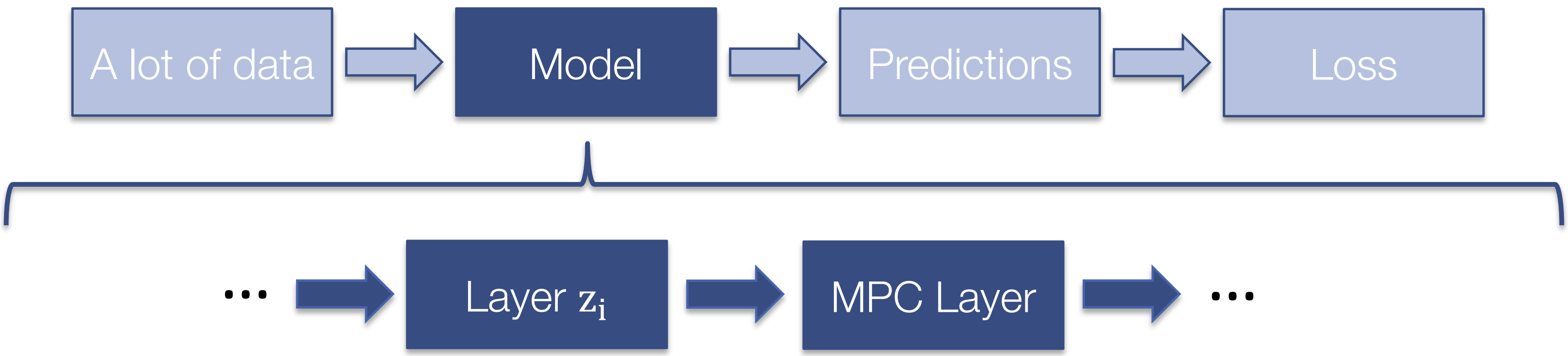
$$
\frac{\partial \ell}{\partial F_t} = d_{\lambda_{t+1}}^\star \otimes \tau_t^\star + \lambda_{t+1}^\star \otimes d_{\tau_t}^\star \qquad \frac{\partial \ell}{\partial f_t} = d_{\lambda_t}^\star
$$

where $K \begin{bmatrix} \vdots \\ d_{\tau_t}^\star \\ d_{\lambda_t}^\star \\ \vdots \end{bmatrix} = - \begin{bmatrix} \vdots \\ \nabla_{\tau_t^\star} \ell \\ 0 \\ \vdots \end{bmatrix}$

Just another LQR problem!

A Differentiable MPC Module

We can differentiate through (non-convex) MPC with a single (convex) LQR solve by differentiating the SQP fixed point



What can we do with this now?

Replace neural network policies in model-free algorithms with MPC policies, and also **replace the unrolled controllers** in other settings (hindsight plan, universal planning networks)

The cost can also be learned! No longer have to hard-code in a known value.

A PyTorch MPC Solver

https://locuslab.github.io/mpc.pytorch

mpc.pytorch

A fast and differentiable model predictive control (MPC) solver for PyTorch. Crafted by Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots, and J. Zico Kolter. For more context and details, see our ICML 2017 paper on OptNet and our (forthcoming) NIPS 2018 paper on differentiable MPC.

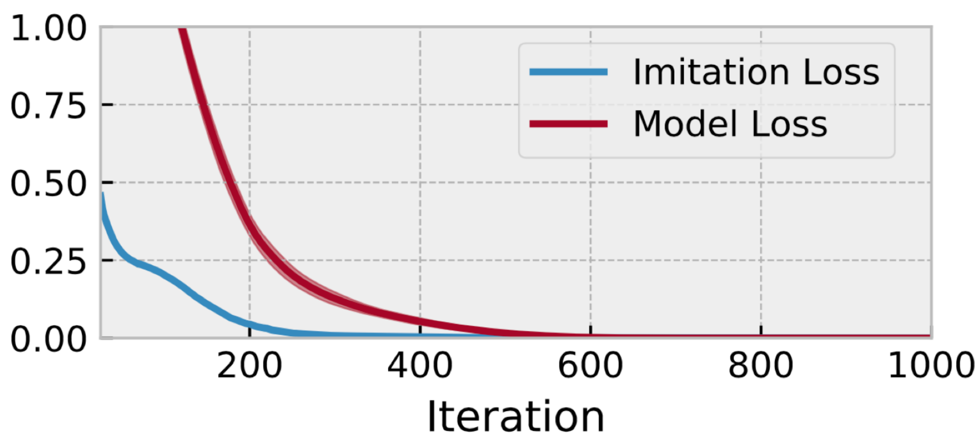
View On GitHub

Control is important!

Optimal control is a widespread field that involve finding an optimal sequence of future actions to take in a system or environment. This is the most useful in domains when you can analytically model your system and can easily define a cost to optimize over your system. This project focuses on solving model predictive control (MPC) with the box-DDP heuristic. MPC is a powerhouse in many real-world domains ranging from short-time horizon robot control tasks to long-time horizon control of chemical processing plants. More recently, the reinforcement learning community, strife with poor sample-complexity and instability issues in model-free learning, has been actively searching for useful model-based applications and priors.

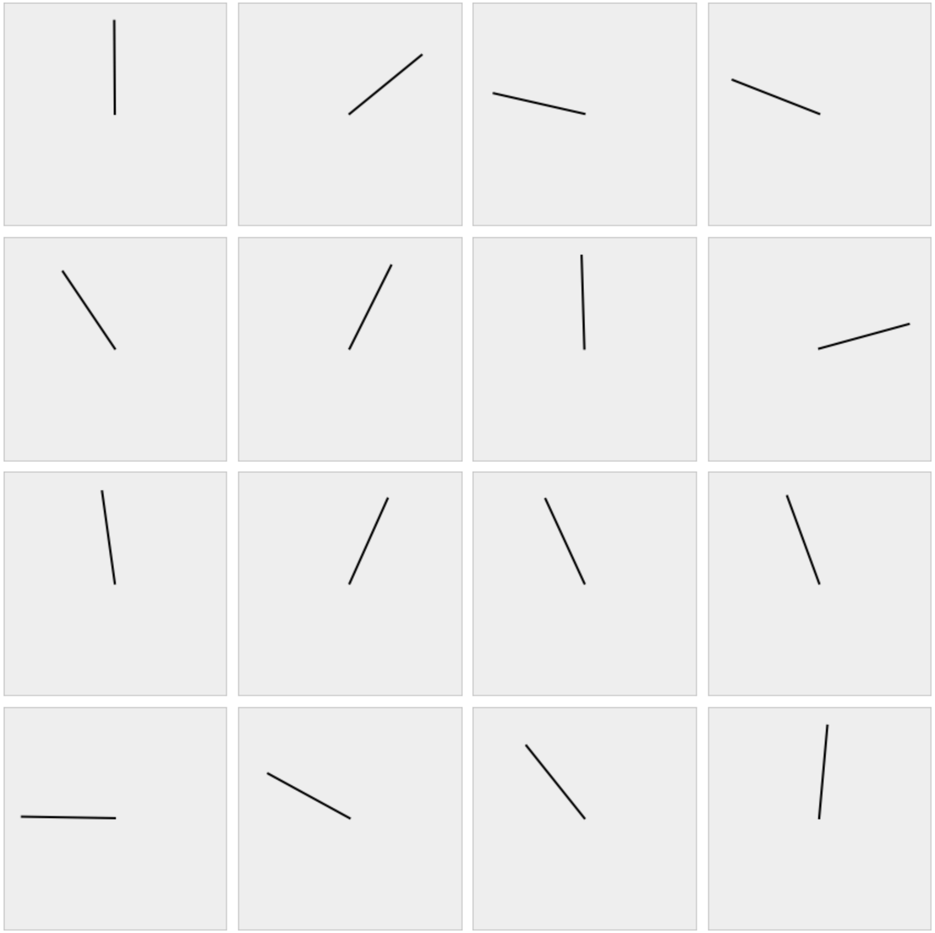
Imitation learning with a linear model

Linear dynamics: $f(x_t, u_t) = Ax_t + Bu_t$
Parameters: $\theta = \{A, B\}$
Trajectory: $\tau_\theta(x_{\text{init}})$ obtained by MPC
Given known θ and sample trajectories, **learn** $\hat{\theta}$
Trajectory (Training) Loss: $\text{MSE}(\tau_\theta(x_{\text{init}}), \tau_{\hat{\theta}}(x_{\text{init}}))$
Model Loss: $\text{MSE}(\theta, \hat{\theta})$



Not guaranteed to converge, but a good sanity check that it does in small cases.

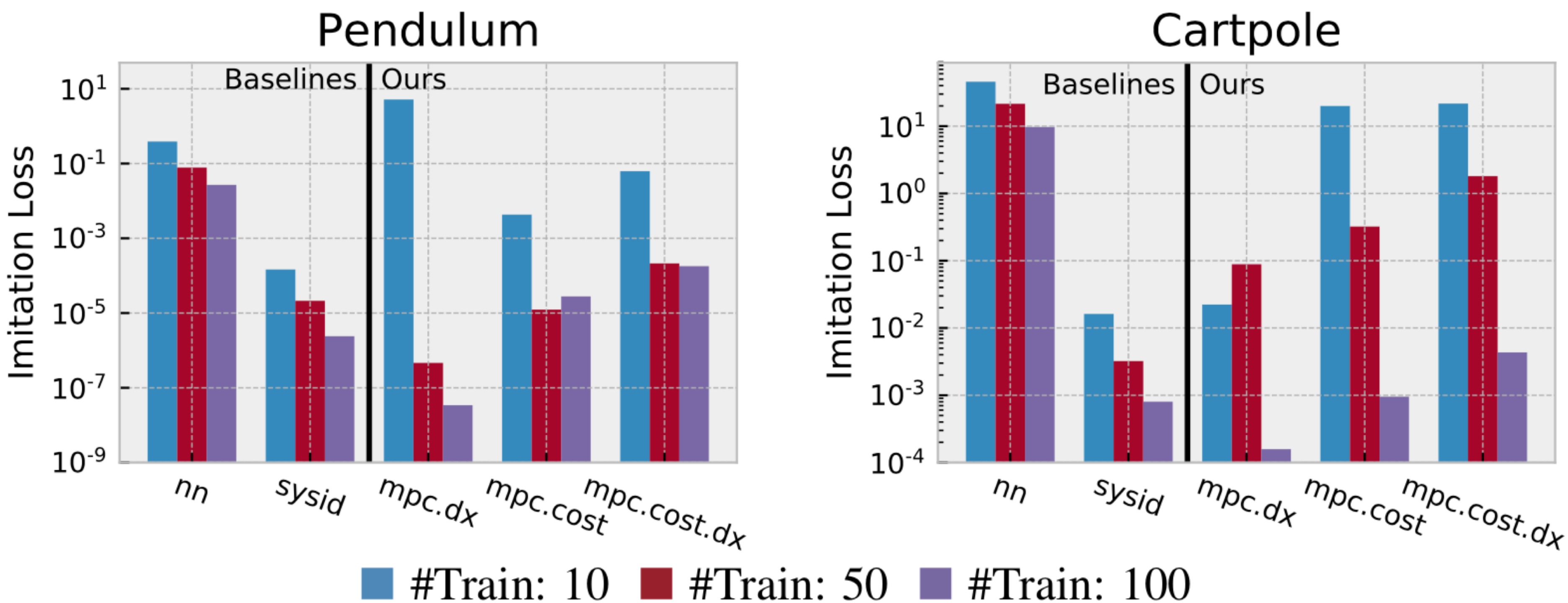
Simple Pendulum Control



Imitation learning with the pendulum/cartpole

Again optimizes the imitation loss with respect to the controller's parameters

Using **only action trajectories** we can recover the true parameters



Optimizing the task loss is often better than SysID in the unrealizable case

True System: Pendulum environment with noise (damping and a wind force)
Approximate Model: Pendulum without the noise terms

NeurIPS 2018

Differentiable MPC for End-to-End Planning and Control

B. Amos, I. Rodriguez, J. Sacks, B. Boots, J. Z. Kolter

Explicit controllers can be learned just as any other layer and integrated with larger black-pox policy classes

Directly optimizing the task loss of controllers is important to do in addition to standard system identification once a task is known

https://locuslab.github.io/mpc.pytorch
https://github.com/locuslab/differentiable-mpc