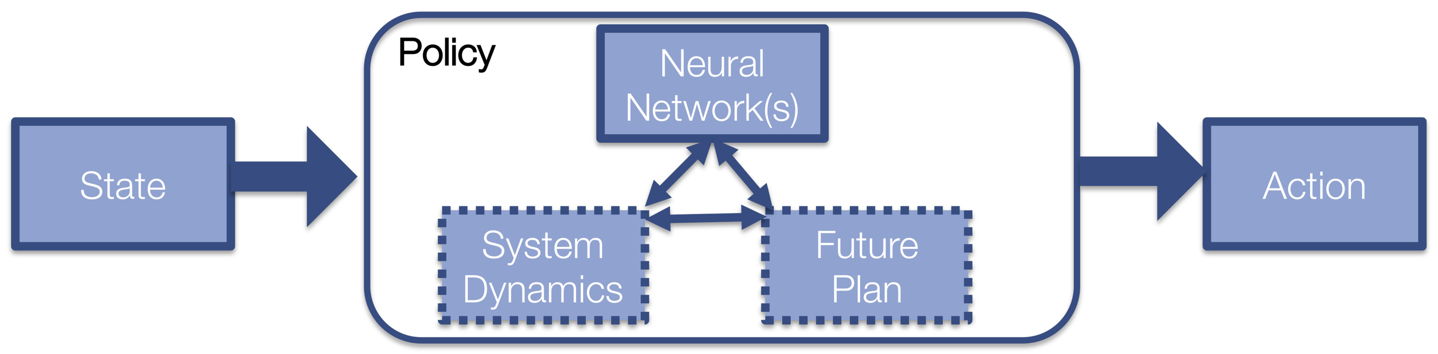
Differentiable MPC for End-to-End Planning and Control

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

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

Introduction and Motivation

Should RL policies have a systems dynamics model or not?



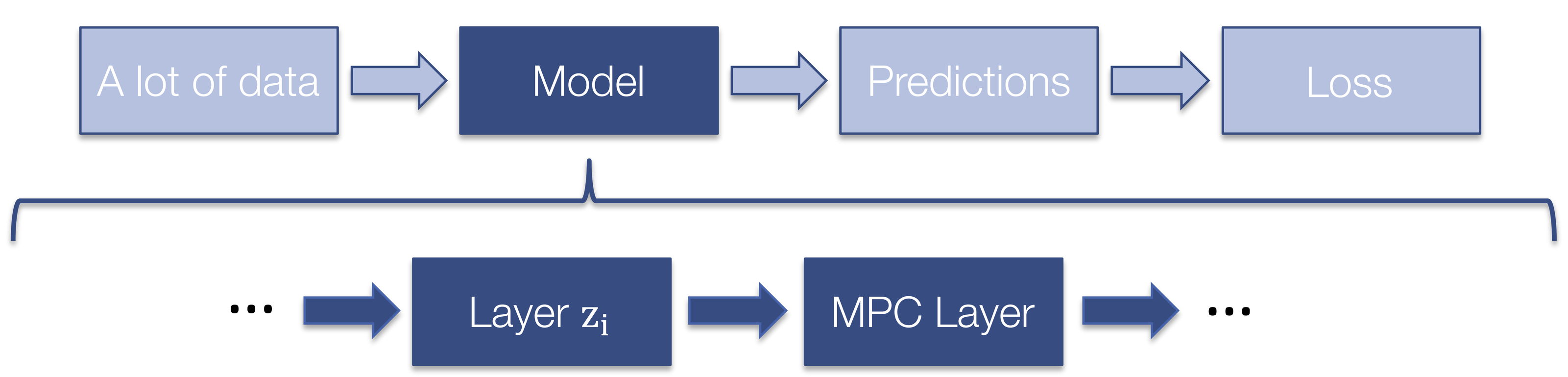
Model-free RL
More general, doesn't make as many assumptions about the world
Rife with poor data efficiency and learning stability issues

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** reinforcement learning (RL) methods is important to get the best of both methods.

- We propose to combine them with a **differentiable control layer** that can be backpropagated through *like any other layer*

Our Contribution: A Differentiable Control Layer



We consider **non-convex control optimization problems**, expanding the scope of OptNet layers

Where can these be used? These differentiable control layers can be integrated as **part of the policy class in model-free algorithms** or imitation learning. Unrolled controllers can be replaced with this.

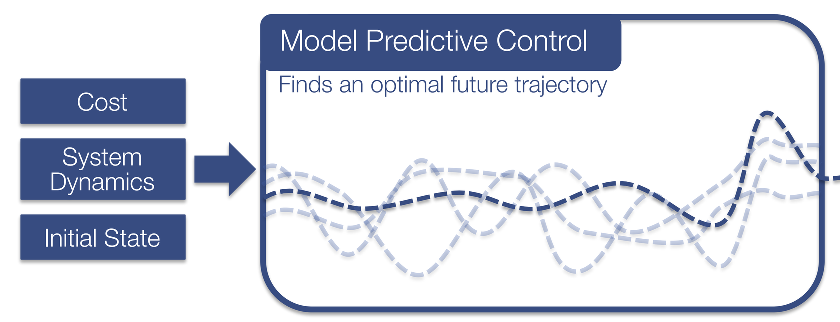
Related Work: Combining model-based and model-free RL

There are a lot of model-based priors for 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)

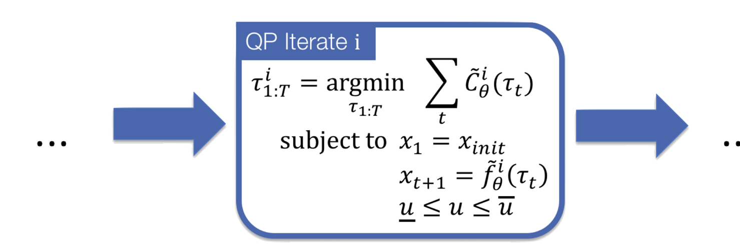
These typically involve:
1. **Using an RNN:** Efficient but not as expressive and general as MPC/iLQR
2. **Unrolling an LQR or gradient-based solver:** Expressive/general but inefficient

Model Predictive Control



$$\tau^\star_{1:T} = \operatorname*{argmin}_{\tau_{1:T}} \sum_t C_\theta(\tau_t) \quad \text{Cost}$$
$$\text{subject to } x_1 = x_{init}$$
$$x_{t+1} = f_\theta(\tau_t) \quad \text{Dynamics}$$
$$\underline{u} \le u \le \overline{u}$$

A widely-used powerhouse of modern control. Typically solved with **sequential quadratic programming**, an iterative method that forms **convex quadratic approximations** to the problem.

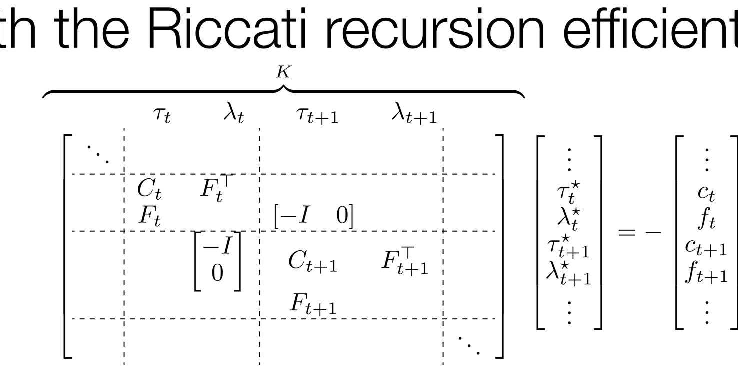


Differentiating MPC: If a fixed-point is reached, then differentiate through the corresponding convex approximation.

Our standalone differentiable MPC solver:
https://locuslab.github.io/mpc.pytorch

LQR, KKT Systems, and Differentiation

Linear-Quadratic Regulator (LQR): A special case of MPC that is convex with a quadratic cost and linear dynamics.
Solving LQR with the Riccati recursion efficiently solves the KKT system



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

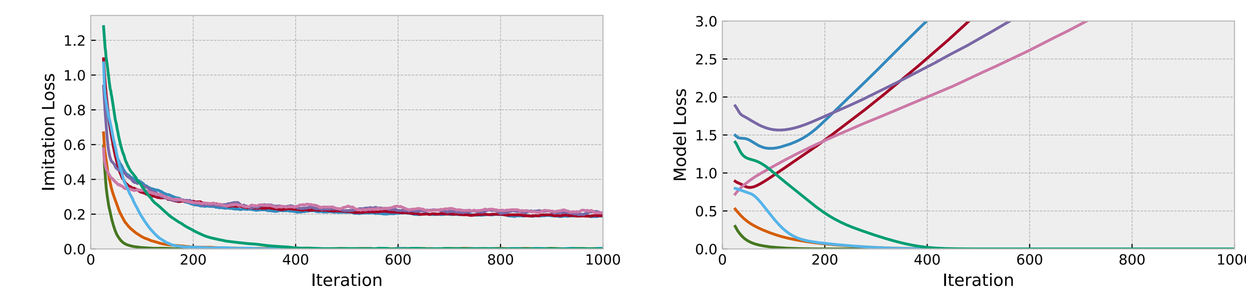
(Just an LQR solve!)

$$\frac{\partial \ell}{\partial C_t} = \frac{1}{2}\left(d^\star_{\tau_t} \otimes \tau^\star_t + \tau^\star_t \otimes d^\star_{\tau_t}\right) \quad \frac{\partial \ell}{\partial c_t} = d^\star_{\tau_t} \quad \frac{\partial \ell}{\partial x_{init}} = d^\star_{\lambda_0}$$
$$\frac{\partial \ell}{\partial F_t} = d^\star_{\lambda_{t+1}} \otimes \tau^\star_t + \lambda^\star_{t+1} \otimes d^\star_{\tau_t} \quad \frac{\partial \ell}{\partial f_t} = d^\star_{\lambda_t}$$

where $K \begin{bmatrix} \vdots \\ d^\star_{\tau_t} \\ d^\star_{\lambda_t} \\ \vdots \end{bmatrix} = - \begin{bmatrix} \vdots \\ \nabla_{\tau^\star_t} \ell \\ 0 \\ \vdots \end{bmatrix}$

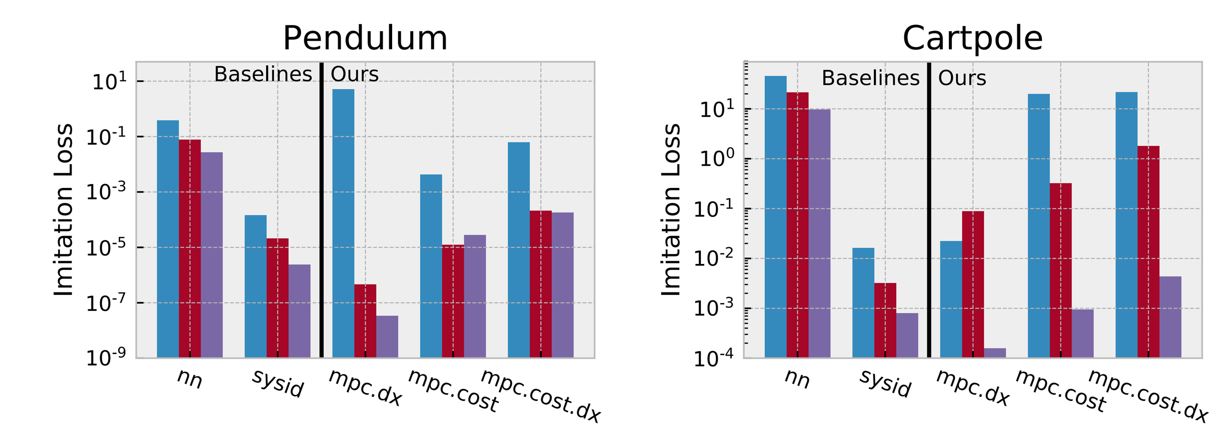
Imitation Learning Experiment: LQR

Given: Expert trajectories from a hand-crafted controller
Goal: Reconstruct **missing parts** (cost and dynamics) of the controller with **imitation learning** given only nominal trajectories
Loss: $\|\tau^\star_{1:T} - \hat{\tau}_{1:T}\|_2^2$ where $\tau_t = [u_t \;\; x_t]$



Imitation Learning Experiments: Pendulum and Cartpole

Given: Expert trajectories from a hand-crafted controller
Goal: Reconstruct **missing parts** (cost and dynamics) of the controller with **imitation learning** given only nominal trajectories
Loss: $\|u^\star_{1:T} - \hat{u}_{1:T}\|_2^2$



Imitation Learning Experiments: Unrealizable Pendulum

In a domain where the **true model class is unrealizable,** traditional system identification (SysID) **may not be the best** if you know the task that you want to use control for. Instead, **directly optimizing the task loss is better.**

We show this in a **pendulum domain** where the true model has noise terms (damping and wind)