There are a lot of model-based priors for reinforcement learning:

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 aulght-pased solver. Expressive, you're all put inclinue in **Algorithm is the Algorithm is defined in** $\begin{bmatrix} 0.2 \end{bmatrix}$

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:

and *no* observations. We consider the following methods: Baselines: *nn* is an LSTM that takes the state *x* as input and predicts the nominal action sequence. In $\frac{1}{K}$ $\frac{1}{K}$ and approximates the parameters of the dynamics by optimizing the next-state transitions. We show this in a pendulum domain where the true model has

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

Differentiable MPC for End-to-End Planning and Control https://locuslab.github.io/mpc.pytorch https://github.com/locuslab/differentiable-mpc Brandon Amos¹ • Ivan Dario Jimenez Rodriguez² • Jacob Sacks² • Byron Boots² • J. Zico Kolter¹³ 1Carnegie Mellon University • 2Georgia Tech • 3Bosch Center for AI

Introduction and Motivation

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.

control. Typically solved with sequential quadratic programming, an iterative method that forms convex quadratic $\frac{x_{t+1} - f_\theta(u_t)}{u \leq u \leq u}$ **Algorithm 1 Differential Approximations to the problem.**

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*

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: $||u_{1:T}^{\star} - \hat{u}||$ ' $1:T \mathsf{\overline{1}}\bar{2}$ (

Should RL policies have a systems dynamics model or not?

Imitation Learning Experiments: Pendulum and Cartpole

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

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: $||\tau_{1:T}^{\star} - \hat{\tau}_{1:T}||_2^2$ where $\tau_t = [u_t \ x_t]$ ̂

In a domain where the true model class is unrealizable, traditional P em identification (SvsID) may not be the best if you know cart root relatively did benchmark domains. Despite being single tasks, they are relatively challenging for a task that you want to use control for. Instead, directly optimizing experts and learners that produce a nominal action sequence *u*1:*^T* (*x*; ✓) where ✓ parameterizes the task loss is better. system identification (SysID) may not be the best if you know the 7

ation Learning Experiments: Unrealizable Pendulum and care care care care and car validation loss observed during the training the training run and report the training run and report the best
The training the training run and report the best test loss. The training run and report the best loss of the Imitation Learning Experiments: Unrealizable Pendulum 7 7 \overline{a} *.* (6)

1: *d*?

LQR, KKT Systems, and Differentiation $\mathbb{E}_{\mathbf{z}}$ with respect to $\mathbb{E}_{\mathbf{z}}$, MNI OYSLUIIS, an 0 $\overline{\mathcal{C}}$ μ Diligigi ilid G iven an optimal nominal trajectory G . The set is a set in G , G ∞ G \mathcal{C} \mathcal{C} \mathcal{C}

⌧1:*^T* = LQR*^T* (0; *C,* r⌧?

t

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 where the initial constraint *x*¹ = *x*init is represented by setting *F*⁰ = 0 and *f*⁰ = *x*init. Differentiating \int har *Katio cost and linear dynamic* a with the Riccali recurs $T_{\text{mean}} \cap \text{resolant}$ D_{mean} dynamic D_{mean} (L ρ_{max} as an efficient way of solving the following the following \mathbb{R}^n nul a quadralic cost α $\overline{}$ vina LQR with the *C^t F* > $\mathsf L$. . . \mathbf{A} H $\frac{1}{2}$. . . 3 variator (I OR): A special ca reduction of the *reduction* and the *reduction ^T* + *cT ,x,* ? *t,x*? *^t*+1 + *Ct,x*⌧ ? where *Ct,x*, *ct,x*, and *Ft,x* are the first block-rows of *Ct*, *ct*, and *Ft*, respectively. Now that we have the optimal trajectory and dual variables in the loss with respect to the gradient of the loss of the loss of t

`*, F,* 0) . Solve (9), reusing the factorizations from the forward pass

Pass: Use the Optivet approach from JAI variables with the backward recursion *terentiate* LQR: \overline{r} \overline{r} where *Ct,x*, *ct,x*, and *Ft,x* are the first block-rows of *Ct*, *ct*, and *Ft*, respectively. Now that we have $\partial \ell$ is $\partial \ell$ under the gradients of the loss with respect to the loss with respect to $\partial \ell$ $\delta \otimes a_{\tau_t}$ $\delta \otimes a_{\tau_t} = a_{\tau_t}$ $\delta \otimes a_{\tau_{t-1}} = a_{\lambda_0}$ $\delta \otimes a_{\tau_t}$ $\mathbb{E}_{\mathbb{P}^n}$ respectively differentiating the CMT conditions. The $\mathbb{E}_{\mathbb{P}^n}$ conditions. *^t* = *F* > *t,x*? *^t*+1 + *Ct,x*⌧ ? \Box **C**
C
Fackwarde Paee: Hea the OntNlat an Daunwarus rass. Optimal optimal ap Ω Ω $\overline{\Omega}$ is a constraint constraint and $\overline{\Omega}$ with respect to the LQR parameters can be obtained by implicitly differentiating the KKT conditions. $\mathcal{A} = \mathcal{A} \cup \mathcal{A}$ of \mathcal{A} and \mathcal{A} of \mathcal{A} of \mathcal{A} and \mathcal{A} $\partial \ell$ ∂C_t $=\frac{1}{2}$ 2 $\left(d_{\tau_{t}}^{\star} \otimes \tau_{t}^{\star} + \tau_{t}^{\star} \otimes d_{\tau_{t}}^{\star} \right)$ $\partial \ell$ ∂c_t $= d^{\star}_{\tau_t}$ $\partial \ell$ ∂x_init $= d^{\star}_{\lambda_0}$ $\mathsf{V}\mathsf{I}$ @*F^t t*
innroach from [Amoo and L where operate approach from product out is the relief.
Is the linear and and developed by the linear system of the linear system of the linear system of the linear sys י**ו ור⊃ן ו**
an I ∩l *K* $\sqrt{2}$ $\overline{}$ $\overline{}$ $\overline{}$. . . d^{\star}_{τ} τ_t d^{\star}_{λ} λ_t $\overline{1}$ 7 7 7 $=$ $\sqrt{2}$ $\overline{1}$ $\overline{}$ $\overline{}$. . . $V_{\tau_t^{\star}}$ *t* ℓ 0 $\overline{1}$ \mathcal{L} 7 $\mathbf{1}$ Backwards Pass: Use the OptNet approach from [Amos and Kolter, 2017] to implicitly differentiate LQR: where (Just an LQR solve!)

 $\partial \ell$ ∂F_t $= d^\star_{\lambda_{t+1}} \otimes \tau_t^\star + \lambda^\star_{t+1} \otimes d^\star_{\tau_t}$

where $\mathcal{L}_{\mathcal{A}}$ is the outer product operator, and $\mathcal{L}_{\mathcal{A}}$ is the outer product operator, and $\mathcal{L}_{\mathcal{A}}$

 $A_{\tau} \otimes d_{\tau}^{\star} = d_{\lambda}^{\star}$ \bigcup_t $\partial \ell$ ∂f_t $= d^{\star}_{\lambda_t}$

 $I\cap\cap\cap$ \cap \blacksquare J O H J J **UUI SIANUAIONE UNIERHILIADE IVIPUS** 1:*^T* with (7) $3x + 2y + 1 = 0$ $\frac{1}{2}$ 1
E H 6 6 6 Our standalone differentiable MPC solver: 0 https://locuslab.github.io/mpc.pytorch

^t + ⌧ ?

Model Predictive Control Parameters: ✓ = *{C, c, F, f}*

$$
\begin{bmatrix}\n\vdots & \ddots & \ddots & \vdots \\
\ddots & \ddots & \ddots & \ddots & \vdots \\
\ddots & \ddots & \ddots & \ddots & \vdots \\
\ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
\ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
\ddots & \ddots & \ddots & \ddots & \ddots & \ddots\n\end{bmatrix}\n\begin{bmatrix}\n\vdots \\
\tau_t^* \\
\lambda_t^* \\
\lambda_{t+1}^* \\
\lambda_{t+1}^* \\
\lambda_{t+1}^* \\
\vdots\n\end{bmatrix} = -\n\begin{bmatrix}\n\vdots \\
c_t \\
f_t \\
c_{t+1} \\
f_{t+1} \\
\vdots\n\end{bmatrix}
$$

 \blacksquare

Algorithm 1 Differentiable LQR Module *(The LQR algorithm is defined in Appendix A)* Input: Initial state *x*init $\sum_{i=1}^{n} f_i(x) = \sum_{i=1}^{n} f_i(x)$ through the corresponding convex are 2: Compute ? 1:*^T* with (7) \overline{a} \overline{b} \overline{c} \overline{c} \overline{d} $\overline{$ **t** reached then dif $\overline{}$ 6 י י
צ⊆ Product Differentiating MPC: If a fixed-point is reached, then differentiate through the corresponding convex approximation.

⌧ and *d*?

are obtained by solving the linear system

Input: Initial state *x*init

4

...

5

.

.

3

4

. . . 5

4

. . .

Imitation Learning Experiment: LQR