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

Abstract

We present foundations for using Model Predictive Control (MPC) as a differentiable policy class for reinforcement learning in continuous state and action spaces. This provides one way of leveraging and combining the advantages of model-free and model-based approaches. Specifically, we differentiate through MPC by using the KKT conditions of the convex approximation at a fixed point of the controller. Using this strategy, we are able to learn the cost and dynamics of a controller via end-to-end learning. Our experiments focus on imitation learning in the pendulum and cartpole domains, where we learn the cost and dynamics terms of an MPC policy class. We show that our MPC policies are significantly more data-efficient than a generic neural network and that our method is superior to traditional system identification in a setting where the expert is unrealizable.

1 Introduction

Model-free reinforcement learning has achieved state-of-the-art results in many challenging domains. However, these methods learn black-box control policies and typically suffer from poor sample complexity and generalization. Alternatively, model-based approaches seek to model the environment the agent is interacting in. Many model-based approaches utilize Model Predictive Control (MPC) to perform complex control tasks [González et al., 2011, Lenz et al., 2015, Liniger et al., 2014, Kamel et al., 2015, Erez et al., 2012, Alexis et al., 2011, Bouffard et al., 2012, Neunert et al., 2016]. MPC leverages a predictive model of the controlled system and solves an optimization problem online in a receding horizon fashion to produce a sequence of control actions. Usually the first control action is applied to the system, after which the optimization problem is solved again for the next time step.

Formally, MPC requires that at each time step we solve the optimization problem:

$$\underset{x_{1:T}\in\mathcal{X},u_{1:T}\in\mathcal{U}}{\operatorname{argmin}} \sum_{t=1}^{T} C_t(x_t,u_t) \quad \text{subject to} \quad x_{t+1}=f(x_t,u_t), \quad x_1=x_{\text{init}}, \tag{1}$$

where x_t, u_t are the state and control at time t, $\mathcal{X}$ and $\mathcal{U}$ are constraints on valid states and controls, $C_t : \mathcal{X}\times\mathcal{U}\to\mathbb{R}$ is a (potentially time-varying) cost function, $f : \mathcal{X}\times\mathcal{U}\to\mathcal{X}$ is a dynamics model, and x_{init} is the initial state of the system. The optimization problem in Equation (1) can be efficiently solved in many ways, for example with the finite-horizon iterative Linear Quadratic Regulator (iLQR) algorithm [Li and Todorov, 2004]. Although these techniques are widely used in control domains, much work in deep reinforcement learning or imitation learning opts instead to use a much simpler policy class such as a linear function or neural network. The advantages of these policy classes is that they are differentiable and the loss can be directly optimized with respect to them while it is typically not possible to do full end-to-end learning with model-based approaches.

In this paper, we consider the task of learning MPC-based policies in an end-to-end fashion, illustrated in Figure 1. That is, we treat MPC as a generic policy class $u=\pi(x_{\text{init}};C,f)$ parameterized by some representations of the cost C and dynamics model f. By differentiating *through* the optimization problem, we can learn the costs and dynamics model to perform a desired task. This is in contrast to

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

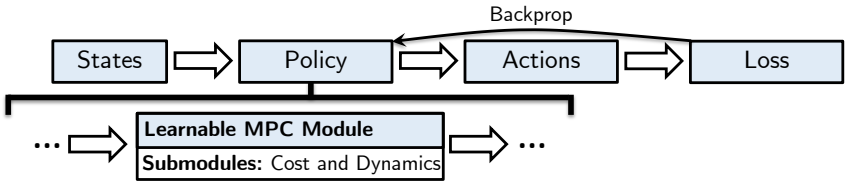

Figure 1: **Illustration of our contribution:** A learnable MPC module that can be integrated into a larger end-to-end reinforcement learning pipeline. Our method allows the controller to be updated with gradient information directly from the task loss.

regressing on collected dynamics or trajectory rollout data and learning each component in isolation, and comes with the typical advantages of end-to-end learning (the ability to train directly based upon the task loss of interest, the ability to "specialize" parameter for a given task, etc).

Still, efficiently differentiating through a complex policy class like MPC is challenging. Previous work with similar aims has either simply unrolled and differentiated through a simple optimization procedure [Tamar et al., 2017] or has considered generic optimization solvers that do not scale to the size of MPC problems [Amos and Kolter, 2017]. This paper makes the following two contributions to this space. First, we provide an efficient method for *analytically* differentiating through an iterative non-convex optimization procedure based upon a box-constrained iterative LQR solver [Tassa et al., 2014]; in particular, we show that the analytical derivative can be computed using *one additional* backward pass of a modified iterative LQR solver. Second, we empirically show that in imitation learning scenarios we can recover the *cost* and *dynamics* from an MPC expert with a loss based only on the actions (and not states). In one notable experiment, we show that directly optimizing the imitation loss results in better performance than vanilla system identification.

2 Background and Related Work

Pure model-free techniques for policy search have demonstrated promising results in many domains by learning *reactive polices* which directly map observations to actions [Mnih et al., 2013, Oh et al., 2016, Gu et al., 2016b, Lillicrap et al., 2015, Schulman et al., 2015, 2016, Gu et al., 2016a]. Despite their success, model-free methods have many drawbacks and limitations, including a lack of interpretability, poor generalization, and a high sample complexity. **Model-based methods** are known to be more sample-efficient than their model-free counterparts. These methods generally rely on learning a dynamics model directly from interactions with the real system and then integrate the learned model into the control policy [Schneider, 1997, Abbeel et al., 2006, Deisenroth and Rasmussen, 2011, Heess et al., 2015, Boedecker et al., 2014]. More recent approaches use a deep network to learn low-dimensional latent state representations and associated dynamics models in this learned representation. They then apply standard trajectory optimization methods on these learned embeddings [Lenz et al., 2015, Watter et al., 2015, Levine et al., 2016]. However, these methods still require a manually specified and hand-tuned cost function, which can become even more difficult in a latent representation. Moreover, there is no guarantee that the learned dynamics model can accurately capture portions of the state space relevant for the task at hand.

To leverage the benefits of both approaches, there has been significant interest in **combining the model-based and model-free paradigms.** In particular, much attention has been dedicated to utilizing model-based priors to accelerate the model-free learning process. For instance, synthetic training data can be generated by model-based control algorithms to guide the policy search or prime a model-free policy [Sutton, 1990, Theodorou et al., 2010, Levine and Abbeel, 2014, Gu et al., 2016b, Venkatraman et al., 2016, Levine et al., 2016, Chebotar et al., 2017, Nagabandi et al., 2017, Sun et al., 2017]. [Bansal et al., 2017] learns a controller and then distills it to a neural network policy which is then fine-tuned with model-free policy learning. However, this line of work usually keeps the model separate from the learned policy.

Alternatively, the policy can include an **explicit planning module** which *leverages learned models* of the system or environment, both of which are learned through model-free techniques. For example, the classic Dyna-Q algorithm [Sutton, 1990] simultaneously learns a model of the environment and uses it to plan. More recent work has explored incorporating such structure into deep networks and learning the policies in an end-to-end fashion. Tamar et al. [2016] uses a recurrent network to predict

the value function by approximating the value iteration algorithm with convolutional layers. Karkus et al. [2017] connects a dynamics model to a planning algorithm and formulates the policy as a structured recurrent network. Silver et al. [2016] and Oh et al. [2017] perform multiple rollouts using an abstract dynamics model to predict the value function. A similar approach is taken by Weber et al. [2017] but directly predicts the next action and reward from rollouts of an explicit environment model. Farquhar et al. [2017] extends model-free approaches, such as DQN [Mnih et al., 2015] and A3C [Mnih et al., 2016], by planning with a tree-structured neural network to predict the cost-to-go. While these approaches have demonstrated impressive results in discrete state and action spaces, they are not applicable to continuous control problems.

To tackle continuous state and action spaces, Pascanu et al. [2017] propose a neural architecture which uses an abstract environmental model to plan and is trained directly from an external task loss. Pong et al. [2018] learn goal-conditioned value functions and use them to plan single or multiple steps of actions in an MPC fashion. Similarly, Pathak et al. [2018] train a goal-conditioned policy to perform rollouts in an abstract feature space but ground the policy with a loss term which corresponds to true dynamics data. The aforementioned approaches can be interpreted as a distilled optimal controller which does not separate components for the cost and dynamics. Taking this analogy further, another strategy is to differentiate through an optimal control algorithm itself. Okada et al. [2017] and Pereira et al. [2018] present a way to differentiate through path integral optimal control [Williams et al., 2016, 2017] and learn a planning policy end-to-end. Srinivas et al. [2018] shows how to embed differentiable planning (unrolled gradient descent over actions) within a goal-directed policy. In a similar vein, Tamar et al. [2017] differentiates through an iterative LQR (iLQR) solver [Li and Todorov, 2004, Xie et al., 2017, Tassa et al., 2014] to learn a cost-shaping term offline. This shaping term enables a shorter horizon controller to approximate the behavior of a solver with a longer horizon to save computation during runtime.

Contributions of our paper. All of these methods require differentiating through planning procedures by explicitly "unrolling" the optimization algorithm itself. While this is a reasonable strategy, it is both memory- and computationally-expensive and challenging when unrolling through many iterations because the time- and space-complexity of the backward pass grows linearly with the forward pass. In contrast, we address this issue by showing how to *analytically* differentiate through the fixed point of a nonlinear MPC solver. Specifically, we compute the derivatives of an iLQR solver with a *single* LQR step in the backward pass. This makes the learning process more computationally tractable while still allowing us to plan in continuous state and action spaces. Unlike model-free approaches, explicit cost and dynamics components can be extracted and analyzed on their own. Moreover, in contrast to pure model-based approaches, the dynamics model and cost function can be learned entirely end-to-end.

3 Differentiable LQR

Discrete-time finite-horizon LQR is a well-studied control method that optimizes a convex quadratic objective function with respect to affine state-transition dynamics from an initial system state x_{init}. Specifically, LQR finds the optimal nominal trajectory $\tau^\star_{1:T} = \{x_t, u_t\}_{1:T}$ by solving the optimization problem

$$\tau^\star_{1:T} = \underset{\tau_{1:T}}{\text{argmin}} \quad \sum_t \frac{1}{2}\tau_t^\top C_t \tau_t + c_t^\top \tau_t \quad \text{subject to} \quad x_1 = x_{\text{init}},\ x_{t+1} = F_t\tau_t + f_t. \tag{2}$$

From a policy learning perspective, this can be interpreted as a module with unknown parameters $\theta = \{C, c, F, f\}$, which can be integrated into a larger end-to-end learning system. The learning process involves taking derivatives of some loss function ℓ, which are then used to update the parameters. Instead of directly computing each of the individual gradients, we present an efficient way of computing the derivatives of the loss function with respect to the parameters

$$\frac{\partial \ell}{\partial \theta} = \frac{\partial \ell}{\partial \tau^\star_{1:T}} \frac{\partial \tau^\star_{1:T}}{\partial \theta}. \tag{3}$$

By interpreting LQR from an optimization perspective [Boyd, 2008], we associate dual variables $\lambda_{1:T}$ with the state constraints. The Lagrangian of the optimization problem is then given by

$$\mathcal{L}(\tau, \lambda) = \sum_t \frac{1}{2}\tau_t^\top C_t \tau_t + \sum_{t=0}^{T-1} \lambda_t^\top (F_t\tau_t + f_t - x_{t+1}), \tag{4}$$

Module 1 Differentiable LQR *(The LQR algorithm is defined in Appendix A)*

Input: Initial state x_{init}
Parameters: $\theta = \{C, c, F, f\}$

Forward Pass:

1: $\tau^\star_{1:T} = \text{LQR}_T(x_{\text{init}}; C, c, F, f)$ ▷ Solve (2)
2: Compute $\lambda^\star_{1:T}$ with (7)

Backward Pass:

1: $d^\star_{\tau_{1:T}} = \text{LQR}_T(0; C, \nabla_{\tau^\star}\ell, F, 0)$ ▷ Solve (9), ideally reusing the factorizations from the forward pass
2: Compute $d^\star_{\lambda_{1:T}}$ with (7)
3: Compute the derivatives of ℓ with respect to C, c, F, f, and x_{init} with (8)

where the initial constraint $x_1 = x_{\text{init}}$ is represented by setting $F_0 = 0$ and $f_0 = x_{\text{init}}$. Differentiating Equation (4) with respect to $\tau^\star_t$ yields

$$\nabla_{\tau_t}\mathcal{L}(\tau^\star, \lambda^\star) = C_t\tau^\star_t + C_t + F_t^\top\lambda^\star_t - \begin{bmatrix}\lambda^\star_{t-1}\\ 0\end{bmatrix} = 0, \tag{5}$$

Thus, the normal approach to solving LQR problems with dynamic Riccati recursion can be viewed as an efficient way of solving the KKT system

$$\overbrace{\begin{array}{cccc} & \tau_t & \lambda_t & \tau_{t+1} & \lambda_{t+1} \\ \ddots & & & & \\ & C_t & F_t^\top & & \\ & F_t & & [-I \quad 0] & \\ & & \begin{bmatrix}-I\\0\end{bmatrix} & C_{t+1} \quad F_{t+1}^\top & \\ & & & F_{t+1} & \\ & & & & \ddots \end{array}}^{K} \begin{bmatrix}\vdots\\ \tau^\star_t\\ \lambda^\star_t\\ \tau^\star_{t+1}\\ \lambda^\star_{t+1}\\ \vdots\end{bmatrix} = -\begin{bmatrix}\vdots\\ c_t\\ f_t\\ c_{t+1}\\ f_{t+1}\\ \vdots\end{bmatrix}. \tag{6}$$

Given an optimal nominal trajectory $\tau^\star_{1:T}$, Equation (5) shows how to compute the optimal dual variables λ with the backward recursion

$$\lambda^\star_T = C_{T,x}\tau^\star_T + c_{T,x} \qquad \lambda^\star_t = F_{t,x}^\top\lambda^\star_{t+1} + C_{t,x}\tau^\star_t + c_{t,x}, \tag{7}$$

where $C_{t,x}$, $c_{t,x}$, and $F_{t,x}$ are the first block-rows of C_t, c_t, and F_t, respectively. Now that we have the optimal trajectory and dual variables, we can compute the gradients of the loss with respect to the parameters. Since LQR is a constrained convex quadratic argmin, the derivatives of the loss with respect to the LQR parameters can be obtained by implicitly differentiating the KKT conditions. Applying the approach from Section 3 of Amos and Kolter [2017], the derivatives are

$$\begin{aligned} \nabla_{C_t}\ell &= \frac{1}{2}\left(d^\star_{\tau_t}\otimes\tau^\star_t + \tau^\star_t\otimes d^\star_{\tau_t}\right) & \nabla_{c_t}\ell &= d^\star_{\tau_t} & \nabla_{x_{\text{init}}}\ell &= d^\star_{\lambda_0} \\ \nabla_{F_t}\ell &= d^\star_{\lambda_{t+1}}\otimes\tau^\star_t + \lambda^\star_{t+1}\otimes d^\star_{\tau_t} & \nabla_{f_t}\ell &= d^\star_{\lambda_t} \end{aligned} \tag{8}$$

where $\otimes$ is the outer product operator, and $d^\star_\tau$ and $d^\star_\lambda$ are obtained by solving the linear system

$$K\begin{bmatrix}\vdots\\ d^\star_{\tau_t}\\ d^\star_{\lambda_t}\\ \vdots\end{bmatrix} = -\begin{bmatrix}\vdots\\ \nabla_{\tau^\star_t}\ell\\ 0\\ \vdots\end{bmatrix}. \tag{9}$$

We observe that Equation (9) is of the same form as the linear system in Equation (6) for the LQR problem. Therefore, we can leverage this insight and solve Equation (9) efficiently by solving another LQR problem that replaces c_t with $\nabla_{\tau^\star_t}\ell$ and f_t with 0. Moreover, this approach enables us to re-use the factorization of K from the forward pass instead of recomputing. Module 1 summarizes the forward and backward passes for a differentiable LQR module.

4 Differentiable MPC

While LQR is a powerful tool, it does not cover realistic control problems with non-linear dynamics and cost. Furthermore, most control problems have natural bounds on the control space that can often be expressed as box constraints. These highly non-convex problems, which we will refer to as model predictive control (MPC), are well-studied in the control literature and can be expressed in the general form

$$\tau_{1:T}^\star = \underset{\tau_{1:T}}{\text{argmin}} \quad \sum_t C_{\theta,t}(\tau_t) \quad \text{subject to} \quad x_1 = x_{\text{init}},\ x_{t+1} = f_\theta(\tau_t),\ \underline{u} \leq u \leq \overline{u}, \tag{10}$$

where the non-convex cost function C_θ and non-convex dynamics function f_θ are (potentially) parameterized by some θ. We note that more generic constraints on the control and state space can be represented as penalties and barriers in the cost function. The standard way of solving the control problem Equation (10) is by iteratively forming and optimizing a convex approximation

$$\tau_{1:T}^i = \underset{\tau_{1:T}}{\text{argmin}} \quad \sum_t \tilde{C}_{\theta,t}^i(\tau_t) \quad \text{subject to} \quad x_1 = x_{\text{init}},\ x_{t+1} = \tilde{f}_\theta^i(\tau_t),\ \underline{u} \leq u \leq \overline{u}, \tag{11}$$

where we have defined the second-order Taylor approximation of the cost around τ^i as

$$\tilde{C}_{\theta,t}^i = C_{\theta,t}(\tau_t^i) + (p_t^i)^\top(\tau_t - \tau_t^i) + \frac{1}{2}(\tau_t - \tau_t^i)^\top H_t^i(\tau_t - \tau_t^i) \tag{12}$$

with $p_t^i = \nabla_{\tau_t^i} C_{\theta,t}$ and $H_t^i = \nabla_{\tau_t^i}^2 C_{\theta,t}$. We also have a first-order Taylor approximation of the dynamics around τ^i as

$$\tilde{f}_{\theta,t}^i(\tau_t) = f_{\theta,t}(\tau_t^i) + F_t^i(\tau_t - \tau_t^i) \tag{13}$$

with $F_t^i = \nabla_{\tau_t^i} f_{\theta,t}$. In practice, a fixed point of Equation (11) is often reached, especially when the dynamics are smooth. As such, differentiating the non-convex problem Equation (10) can be done exactly by using the final convex approximation. Without the box constraints, the fixed point in Equation (11) could be differentiated with LQR as we show in Section 3. In the next section, we will show how to extend this to the case where we have box constraints on the controls as well.

4.1 Differentiating Box-Constrained QPs

First, we consider how to differentiate a more generic box-constrained convex QP of the form

$$x^\star = \underset{x}{\text{argmin}} \quad \frac{1}{2}x^\top Q x + p^\top x \quad \text{subject to} \quad Ax = b,\ \underline{x} \leq x \leq \overline{x}. \tag{14}$$

Given active inequality constraints at the solution in the form $\tilde{G}x = \tilde{h}$, this problem turns into an equality-constrained optimization problem with the solution given by the linear system

$$\begin{bmatrix} Q & A^\top & \tilde{G}^\top \\ A & 0 & 0 \\ \tilde{G} & 0 & 0 \end{bmatrix} \begin{bmatrix} x^\star \\ \lambda^\star \\ \tilde{\nu}^\star \end{bmatrix} = - \begin{bmatrix} p \\ b \\ \tilde{h} \end{bmatrix} \tag{15}$$

With some loss function ℓ that depends on $x^\star$, we can use the approach in Amos and Kolter [2017] to obtain the derivatives of ℓ with respect to Q, p, A, and b as

$$\nabla_Q \ell = \frac{1}{2}(d_x^\star \otimes x^\star + x^\star \otimes d_x^\star) \qquad \nabla_p \ell = d_x^\star \qquad \nabla_A \ell = d_\lambda^\star \otimes x^\star + \lambda^\star \otimes d_x^\star \qquad \nabla_b \ell = -d_\lambda^\star \tag{16}$$

where $d_x^\star$ and $d_\lambda^\star$ are obtained by solving the linear system

$$\begin{bmatrix} Q & A^\top & \tilde{G}^\top \\ A & 0 & 0 \\ \tilde{G} & 0 & 0 \end{bmatrix} \begin{bmatrix} d_x^\star \\ d_\lambda^\star \\ d_{\tilde{\nu}}^\star \end{bmatrix} = - \begin{bmatrix} \nabla_{x^\star}\ell \\ 0 \\ 0 \end{bmatrix} \tag{17}$$

The constraint $\tilde{G}d_x^\star = 0$ is equivalent to the constraint $d_{x_i}^\star = 0$ if $x_i^\star \in \{\underline{x}_i, \overline{x}_i\}$. Thus solving the system in Equation (17) is equivalent to solving the optimization problem

$$d_x^\star = \underset{d_x}{\text{argmin}} \quad \frac{1}{2}d_x^\top Q d_x + (\nabla_{x^\star}\ell)^\top d_x \quad \text{subject to} \quad Ad_x = 0,\ d_{x_i} = 0 \ \text{ if } \ x_i^\star \in \{\underline{x}_i, \overline{x}_i\} \tag{18}$$

Module 2 Differentiable MPC *(The MPC algorithm is defined in Appendix A)*

Given: Initial state x_{init} and initial control sequence u_{init}
Parameters: θ of the objective $C_\theta(\tau)$ and dynamics $f_\theta(\tau)$

Forward Pass:

1: $\tau^\star_{1:T} = \text{MPC}_{T,\underline{u},\overline{u}}(x_{\text{init}}, u_{\text{init}}; C_\theta, F_\theta)$ ▷ Solve Equation (10)
2: *The solver should reach the fixed point in (11) to obtain approximations to the cost H^n_θ and dynamics F^n_θ*
3: Compute $\lambda^\star_{1:T}$ with (7)

Backward Pass:

1: $\tilde{F}^n_\theta$ is F^n_θ with the rows corresponding to the tight control constraints zeroed
2: $d^\star_{\tau_{1:T}} = \text{LQR}_T(0; H^n_\theta, \nabla_{\tau^\star}\ell, \tilde{F}^n_\theta, 0)$ ▷ Solve (19), ideally reusing the factorizations from the forward pass
3: Compute $d^\star_{\lambda_{1:T}}$ with (7)
4: Differentiate ℓ with respect to the approximations H^n_θ and F^n_θ with (8)
5: Differentiate these approximations with respect to θ and use the chain rule to obtain $\partial\ell/\partial\theta$

4.2 Differentiating MPC with Box Constraints

At a fixed point, we can use Equation (16) to compute the derivatives of the MPC problem, where $d^\star_\tau$ and $d^\star_\lambda$ are found by solving the linear system in Equation (9) with the additional constraint that $d_{u_{t,i}} = 0$ if $u^\star_{t,i} \in \{\underline{u}_{t,i}, \overline{u}_{t,i}\}$. Solving this system can be equivalently written as a zero-constrained LQR problem of the form

$$\begin{aligned} d^\star_{\tau_{1:T}} = \underset{d_{\tau_{1:T}}}{\text{argmin}} \quad & \sum_t \frac{1}{2} d^\top_{\tau_t} H^n_t d_{\tau_t} + (\nabla_{\tau^\star_t}\ell)^\top d_{\tau_t} \\ \text{subject to} \quad & d_{x_1} = 0,\ d_{x_{t+1}} = F^n_t d_{\tau_t},\ d_{u_{t,i}} = 0 \ \text{ if } \ u^\star_i \in \{\underline{u}_{t,i}, \overline{u}_{t,i}\} \end{aligned} \tag{19}$$

where n is the iteration that Equation (11) reaches a fixed point, and H^n and F^n are the corresponding approximations to the objective and dynamics defined earlier. Module 2 summarizes the proposed differentiable MPC module. To solve the MPC problem in Equation (10) and reach the fixed point in Equation (11), we use the box-DDP heuristic [Tassa et al., 2014]. For the zero-constrained LQR problem in Equation (19) to compute the derivatives, we use an LQR solver that zeros the appropriate controls.

4.3 Drawbacks of Our Approach

Sometimes the controller does not run for long enough to reach a fixed point of Equation (11), or a fixed point doesn't exist, which often happens when using neural networks to approximate the dynamics. When this happens, Equation (19) cannot be used to differentiate through the controller, because it assumes a fixed point. Differentiating through the final iLQR iterate that's not a fixed point will usually give the wrong gradients. Treating the iLQR procedure as a compute graph and differentiating through the unrolled operations is a reasonable alternative in this scenario that obtains surrogate gradients to the control problem. However, as we empirically show in Section 5.1, the backward pass of this method scales linearly with the number of iLQR iterations used in the forward. Instead, fixed-point differentiation is constant time and only requires a single iLQR solve.

5 Experimental Results

In this section, we present several results that highlight the performance and capabilities of differentiable MPC in comparison to neural network policies and vanilla system identification (SysId). We show 1) superior runtime performance compared to an unrolled solver, 2) the ability of our method to recover the cost and dynamics of a controller with imitation, and 3) the benefit of directly optimizing the task loss over vanilla SysId.

We have released our differentiable MPC solver as a standalone open source package that is available at `https://github.com/locuslab/mpc.pytorch` and our experimental code for this paper is also openly available at `https://github.com/locuslab/differentiable-mpc`. Our experiments are implemented with PyTorch [Paszke et al., 2017].

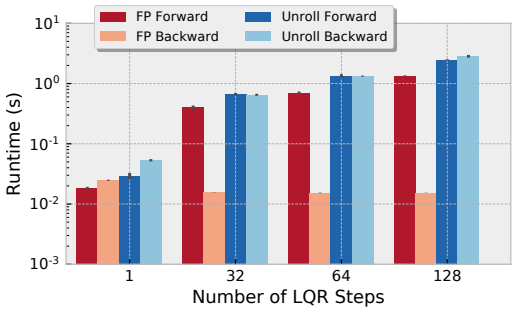

Figure 2: Runtime comparison of fixed point differentiation (FP) to unrolling the iLQR solver (Unroll), averaged over 10 trials.

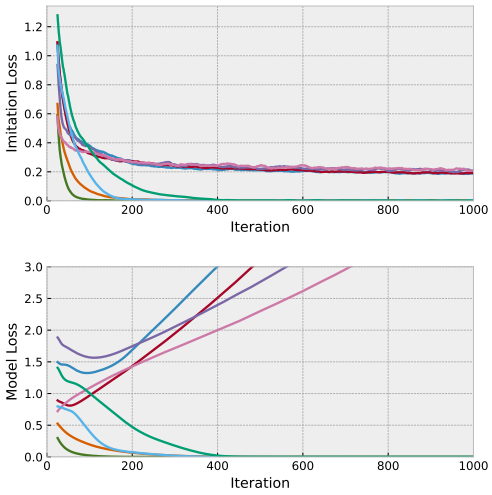

Figure 3: Model and imitation losses for the LQR imitation learning experiments.

5.1 MPC Solver Performance

Figure 2 highlights the performance of our differentiable MPC solver. We compare to an alternative version where each box-constrained iLQR iteration is individually unrolled, and gradients are computed by differentiating through the entire unrolled chain. As illustrated in the figure, these unrolled operations incur a substantial extra cost. Our differentiable MPC solver 1) is slightly more computationally efficient even in the forward pass, as it does not need to create and maintain the backward pass variables; 2) is more memory efficient in the forward pass for this same reason (by a factor of the number of iLQR iterations); and 3) is *significantly* more efficient in the backward pass, especially when a large number of iLQR iterations are needed. The backward pass is essentially free, as it can reuse all the factorizations for the forward pass and does not require multiple iterations.

5.2 Imitation Learning: Linear-Dynamics Quadratic-Cost (LQR)

In this section, we show results to validate the MPC solver and gradient-based learning approach for an imitation learning problem. The expert and learner are LQR controllers that share all information except for the linear system dynamics $f(x_t, u_t) = Ax_t + Bu_t$. The controllers have the same quadratic cost (the identity), control bounds $[-1, 1]$, horizon (5 timesteps), and 3-dimensional state and control spaces. Though the dynamics can also be recovered by fitting next-state transitions, we show that we can alternatively use imitation learning to recover the dynamics using only controls.

Given an initial state x, we can obtain nominal actions from the controllers as $u_{1:T}(x; \theta)$, where $\theta = \{A, B\}$. We randomly initialize the learner's dynamics with $\hat{\theta}$ and minimize the **imitation loss**

$$\mathcal{L} = \mathbb{E}_x \left[||\tau_{1:T}(x; \theta) - \tau_{1:T}(x; \hat{\theta})||_2^2 \right], .$$

We do learning by differentiating $\mathcal{L}$ with respect to $\hat{\theta}$ (using mini-batches with 32 examples) and taking gradient steps with RMSprop [Tieleman and Hinton, 2012]. Figure 3 shows the model and imitation loss of eight randomly sampled initial dynamics, where the **model loss** is $\mathrm{MSE}(\theta, \hat{\theta})$. The model converges to the true parameters in half of the trials and achieves a perfect imitation loss. The other trials get stuck in a local minimum of the imitation loss and causes the approximate model to significantly diverge from the true model. These faulty trials highlight that despite the LQR problem being convex, the optimization problem of some loss function w.r.t. the controller's parameters is a (potentially difficult) non-convex optimization problem that typically does not have convergence guarantees.

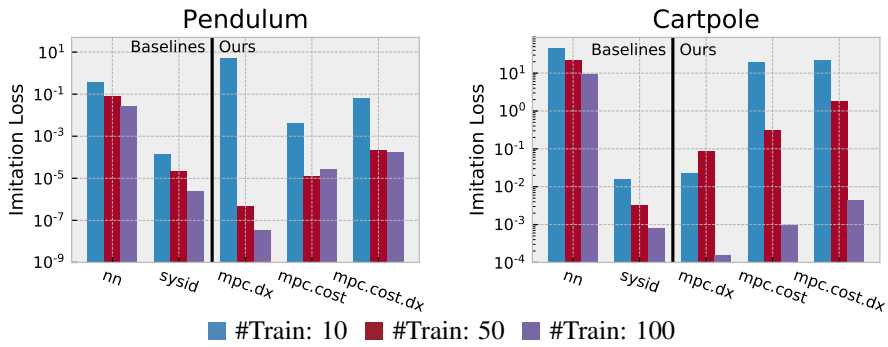

Figure 4: Learning results on the (simple) pendulum and cartpole environments. We select the best validation loss observed during the training run and report the best test loss.

5.3 Imitation Learning: Non-Convex Continuous Control

We next demonstrate the ability of our method to do imitation learning in the pendulum and cartpole benchmark domains. Despite being simple tasks, they are relatively challenging for a generic poicy to learn quickly in the imitation learning setting. In our experiments we use MPC experts and learners that produce a nominal action sequence $u_{1:T}(x;\theta)$ where θ parameterizes the model that's being optimized. The goal of these experiments is to optimize the imitation loss $\mathcal{L} = \mathbb{E}_x \left[||u_{1:T}(x;\theta) - u_{1:T}(x;\hat{\theta})||_2^2 \right]$, again which we can uniquely do using *only* observed controls 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 this setting we optimize the imitation loss directly. *sysid* assumes the cost of the controller is known and approximates the parameters of the dynamics by optimizing the next-state transitions.

Our Methods: *mpc.dx* assumes the cost of the controller is known and approximates the parameters of the dynamics by directly optimizing the imitation loss. *mpc.cost* assumes the dynamics of the controller is known and approximates the cost by directly optimizing the imitation loss. *mpc.cost.dx* approximates both the cost and parameters of the dynamics of the controller by directly optimizing the imitation loss.

In all settings that involve learning the dynamics (*sysid*, *mpc.dx*, and *mpc.cost.dx*) we use a parameterized version of the true dynamics. In the pendulum domain, the parameters are the mass, length, and gravity; and in the cartpole domain, the parameters are the cart's mass, pole's mass, gravity, and length. For cost learning in *mpc.cost* and *mpc.cost.dx* we parameterize the cost of the controller as the weighted distance to a goal state $C(\tau) = ||w_g \circ (\tau - \tau_g)||_2^2$. We have found that simultaneously learning the weights w_g and goal state τ_g is instable and in our experiments we alternate learning of w_g and τ_g independently every 10 epochs. We collected a dataset of trajectories from an expert controller and vary the number of trajectories our models are trained on. A single trial of our experiments takes 1-2 hours on a modern CPU. We optimize the *nn* setting with Adam [Kingma and Ba, 2014] with a learning rate of 10^{-4} and all other settings are optimized with RMSprop [Tieleman and Hinton, 2012] with a learning rate of 10^{-2} and a decay term of 0.5.

Figure 4 shows that in nearly every case we are able to directly optimize the imitation loss with respect to the controller and we significantly outperform a general neural network policy trained on the same information. In many cases we are able to recover the true cost function and dynamics of the expert. More information about the training and validation losses are in Appendix B. The comparison between our approach *mpc.dx* and SysId is notable, as we are able to recover equivalent performance to SysId with our models using *only* the control information and *without* using state information.

Again, while we emphasize that these are simple tasks, there are stark differences between the approaches. Unlike the generic network-based imitation learning, the MPC policy can exploit its inherent structure. Specifically, because the network contains a well-defined notion of the dynamics and cost, it is able to learn with much lower sample complexity that a typical network. But unlike pure system identification (which would be reasonable only for the case where the physical parameters are unknown but all other costs are known), the differentiable MPC policy can naturally be adapted to objectives *besides* simple state prediction, such as incorporating the additional cost learning portion.

Figure 5: Convergence results in the non-realizable Pendulum task.

5.4 Imitation Learning: SysId with a non-realizable expert

All of our previous experiments that involve SysId and learning the dynamics are in the unrealistic case when the expert's dynamics are in the model class being learned. In this experiment we study a case where the expert's dynamics are *outside* of the model class being learned. In this setting we will do imitation learning for the parameters of a dynamics function with vanilla SysId and by directly optimizing the imitation loss (*sysid* and the *mpc.dx* in the previous section, respectively).

SysId often fits observations from a noisy environment to a simpler model. In our setting, we collect optimal trajectories from an expert in the pendulum environment that has an additional damping term and also has another force acting on the point-mass at the end (which can be interpreted as a "wind" force). We do learning with dynamics models that *do not* have these additional terms and therefore we *cannot* recover the expert's parameters. Figure 5 shows that even though vanilla SysId is slightly better at optimizing the next-state transitions, it finds an inferior model for imitation compared to our approach that directly optimizes the imitation loss.

We argue that the goal of doing SysId is rarely in isolation and always serves the purpose of performing a more sophisticated task such as imitation or policy learning. Typically SysId is merely a surrogate for optimizing the task and we claim that the task's loss signal provides useful information to guide the dynamics learning. Our method provides one way of doing this by allowing the task's loss function to be directly differentiated with respect to the dynamics function being learned.

6 Conclusion

This paper lays the foundations for differentiating and learning MPC-based controllers within reinforcement learning and imitation learning. Our approach, in contrast to the more traditional strategy of "unrolling" a policy, has the benefit that it is much less computationally and memory intensive, with a backward pass that is essentially free given the number of iterations required for a the iLQR optimizer to converge to a fixed point. We have demonstrated our approach in the context of imitation learning, and have highlighted the potential advantages that the approach brings over generic imitation learning and system identification.

We also emphasize that one of the primary contributions of this paper is to define and set up the framework for differentiating through MPC in general. Given the recent prominence of attempting to incorporate planning and control methods into the loop of deep network architectures, the techniques here offer a method for efficiently integrating MPC policies into such situations, allowing these architectures to make use of a very powerful function class that has proven extremely effective in practice. The future applications of our differentiable MPC method include tuning model parameters to task-specific goals and incorporating joint model-based and policy-based loss functions; and our method can also be extended for stochastic control.

Acknowledgments

BA is supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE1252522. We thank Alfredo Canziani, Shane Gu, and Yuval Tassa for insightful discussions.

References

Pieter Abbeel, Morgan Quigley, and Andrew Y Ng. Using inaccurate models in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 1–8. ACM, 2006.

Kostas Alexis, Christos Papachristos, George Nikolakopoulos, and Anthony Tzes. Model predictive quadrotor indoor position control. In *Control & Automation (MED), 2011 19th Mediterranean Conference on*, pages 1247–1252. IEEE, 2011.

Brandon Amos and J Zico Kolter. OptNet: Differentiable Optimization as a Layer in Neural Networks. In *Proceedings of the International Conference on Machine Learning*, 2017.

Somil Bansal, Roberto Calandra, Sergey Levine, and Claire Tomlin. Mbmf: Model-based priors for model-free reinforcement learning. *arXiv preprint arXiv:1709.03153*, 2017.

Joschika Boedecker, Jost Tobias Springenberg, Jan Wulfing, and Martin Riedmiller. Approximate real-time optimal control based on sparse gaussian process models. In *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, 2014.

P. Bouffard, A. Aswani, , and C. Tomlin. Learning-based model predictive control on a quadrotor: Onboard implementation and experimental results. In *IEEE International Conference on Robotics and Automation*, 2012.

Stephen Boyd. Lqr via lagrange multipliers. Stanford EE 363: Linear Dynamical Systems, 2008. URL http://stanford.edu/class/ee363/lectures/lqr-lagrange.pdf.

Yevgen Chebotar, Karol Hausman, Marvin Zhang, Gaurav Sukhatme, Stefan Schaal, and Sergey Levine. Combining model-based and model-free updates for trajectory-centric reinforcement learning. *arXiv preprint arXiv:1703.03078*, 2017.

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, 2011.

T. Erez, Y. Tassa, and E. Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *International Conference on Intelligent Robots and Systems*, 2012.

Gregory Farquhar, Tim Rocktäschel, Maximilian Igl, and Shimon Whiteson. Treeqn and atreec: Differentiable tree planning for deep reinforcement learning. *arXiv preprint arXiv:1710.11417*, 2017.

Ramón González, Mirko Fiacchini, José Luis Guzmán, Teodoro Álamo, and Francisco Rodríguez. Robust tube-based predictive control for mobile robots in off-road conditions. *Robotics and Autonomous Systems*, 59 (10):711–726, 2011.

Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E Turner, and Sergey Levine. Q-prop: Sample-efficient policy gradient with an off-policy critic. *arXiv preprint arXiv:1611.02247*, 2016a.

Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *Proceedings of the International Conference on Machine Learning*, 2016b.

Nicolas Heess, Gregory Wayne, David Silver, Tim Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2944–2952, 2015.

Mina Kamel, Kostas Alexis, Markus Achtelik, and Roland Siegwart. Fast nonlinear model predictive control for multicopter attitude tracking on so (3). In *Control Applications (CCA), 2015 IEEE Conference on*, pages 1160–1166. IEEE, 2015.

Peter Karkus, David Hsu, and Wee Sun Lee. Qmdp-net: Deep learning for planning under partial observability. In *Advances in Neural Information Processing Systems*, pages 4697–4707, 2017.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Ian Lenz, Ross A Knepper, and Ashutosh Saxena. Deepmpc: Learning deep latent features for model predictive control. In *Robotics: Science and Systems*, 2015.

Sergey Levine. Optimal control and planning. Berkeley CS 294-112: Deep Reinforcement Learning, 2017. URL http://rll.berkeley.edu/deeprlcourse/f17docs/lecture_8_model_based_planning.pdf.

Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*, pages 1071–1079, 2014.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. 2004.

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.

Alexander Liniger, Alexander Domahidi, and Manfred Morari. Optimization-based autonomous racing of 1:43 scale rc cars. In *Optimal Control Applications and Methods*, pages 628–647, 2014.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *arXiv preprint arXiv:1708.02596*, 2017.

Michael Neunert, Cedric de Crousaz, Fardi Furrer, Mina Kamel, Farbod Farshidian, Roland Siegwart, and Jonas Buchli. Fast Nonlinear Model Predictive Control for Unified Trajectory Optimization and Tracking. In *ICRA*, 2016.

Junhyuk Oh, Valliappa Chockalingam, Satinder Singh, and Honglak Lee. Control of memory, active perception, and action in minecraft. *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016.

Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. In *Advances in Neural Information Processing Systems*, pages 6120–6130, 2017.

Masashi Okada, Luca Rigazio, and Takenobu Aoshima. Path integral networks: End-to-end differentiable optimal control. *arXiv preprint arXiv:1706.09597*, 2017.

Razvan Pascanu, Yujia Li, Oriol Vinyals, Nicolas Heess, Lars Buesing, Sebastien Racanière, David Reichert, Théophane Weber, Daan Wierstra, and Peter Battaglia. Learning model-based planning from scratch. *arXiv preprint arXiv:1707.06170*, 2017.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Deepak Pathak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Yide Shentu, Evan Shelhamer, Jitendra Malik, Alexei A Efros, and Trevor Darrell. Zero-shot visual imitation. *arXiv preprint arXiv:1804.08606*, 2018.

Marcus Pereira, David D. Fan, Gabriel Nakajima An, and Evangelos Theodorou. Mpc-inspired neural network policies for sequential decision making. *arXiv preprint arXiv:1802.05803*, 2018.

Vitchyr Pong, Shixiang Gu, Murtaza Dalal, and Sergey Levine. Temporal difference models: Model-free deep rl for model-based control. *arXiv preprint arXiv:1802.09081*, 2018.

Jeff G Schneider. Exploiting model uncertainty estimates for safe dynamic control learning. In *Advances in neural information processing systems*, pages 1047–1053, 1997.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1889–1897, 2015.

John Schulman, Philpp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *International Conference on Learning Representations*, 2016.

David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, et al. The predictron: End-to-end learning and planning. *arXiv preprint arXiv:1612.08810*, 2016.

Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks. *arXiv preprint arXiv:1804.00645*, 2018.

Liting Sun, Cheng Peng, Wei Zhan, and Masayoshi Tomizuka. A fast integrated planning and control framework for autonomous driving via imitation learning. In *arXiv preprint arXiv:1707.02515*, 2017.

Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the seventh international conference on machine learning*, pages 216–224, 1990.

Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems*, pages 2154–2162, 2016.

Aviv Tamar, Garrett Thomas, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. Learning from the hindsight plan—episodic mpc improvement. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 336–343. IEEE, 2017.

Yuval Tassa, Nicolas Mansard, and Emo Todorov. Control-limited differential dynamic programming. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 1168–1175. IEEE, 2014.

Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. A generalized path integral control approach to reinforcement learning. *Journal of Machine Learning Research*, 11(Nov):3137–3181, 2010.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Arun Venkatraman, Roberto Capobianco, Lerrel Pinto, Martial Hebert, Daniele Nardi, and J Andrew Bagnell. Improved learning of dynamics models for control. In *International Symposium on Experimental Robotics*, pages 703–713. Springer, 2016.

Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in neural information processing systems*, pages 2746–2754, 2015.

Théophane Weber, Sébastien Racanière, David P Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. Imagination-augmented agents for deep reinforcement learning. *arXiv preprint arXiv:1707.06203*, 2017.

Grady Williams, Paul Drews, Brian Goldfain, James M Rehg, and Evangelos A Theodorou. Aggressive driving with model predictive path integral control. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 1433–1440. IEEE, 2016.

Grady Williams, Andrew Aldrich, and Evangelos A Theodorou. Model predictive path integral control: From theory to parallel computation. *Journal of Guidance, Control, and Dynamics*, 40(2):344–357, 2017.

Zhaoming Xie, C. Karen Liu, and Kris Hauser. Differential Dynamic Programming with Nonlinear Constraints. In *International Conference on Robotics and Automation (ICRA)*, 2017.

A LQR and MPC Algorithms

Algorithm 1 $\mathrm{LQR}_T(x_{\text{init}}; C, c, F, f)$ *Solves Equation (2) as described in Levine [2017]*

The **state space** is n-dimensional and the **control space** is m-dimensional.
$T \in \mathbb{Z}_+$ is the **horizon length**, the number of nominal timesteps to optimize for in the future.
$x_{\text{init}} \in \mathbb{R}^n$ is the initial state
$C \in \mathbb{R}^{T \times n+m \times n+m}$ and $c \in \mathbb{R}^{T \times n+m}$ are the quadratic cost terms. Every C_t must be PSD.
$F \in \mathbb{R}^{T \times n \times n+m}$ $f \in \mathbb{R}^{T \times n}$ are the affine cost terms.

▷ **Backward Recursion**
$V_T = v_T = 0$
for t = T to 1 **do**
$Q_t = C_t + F_t^\top V_{t+1} F_t$
$q_t = c_t + F_t^\top V_{t+1} f_t + F_t^\top v_{t+1}$
$K_t = -Q_{t,uu}^{-1} Q_{t,ux}$
$k_t = -Q_{t,uu}^{-1} q_{t,u}$
$V_t = Q_{t,xx} + Q_{t,xu} K_t + K_t^\top Q_{t,ux} + K_t^\top Q_{t,uu} K_t$
$v_t = q_{t,x} + Q_{t,xu} k_t + K_t^\top q_{t,u} + K_t^\top Q_{t,uu} k_t$
end for

▷ **Forward Recursion**
$x_1 = x_{\text{init}}$
for t = 1 to T **do**
$u_t = K_t x_t + k_t$
$x_{t+1} = F_t \begin{bmatrix} x_t \\ u_t \end{bmatrix} + f_t$
end for

return $x_{1:T}, u_{1:T}$

Algorithm 2 $\text{MPC}_{T,\underline{u},\overline{u}}(x_{\text{init}}, u_{\text{init}}; C, f)$ *Solves Equation (10) as described in Tassa et al. [2014]*

The **state space** is n-dimensional and the **control space** is m-dimensional.
$T \in \mathbb{Z}_+$ is the **horizon length**, the number of nominal timesteps to optimize for in the future.
$\underline{u}, \overline{u} \in \mathbb{R}^m$ are respectively the control **lower-** and **upper-bounds**.
$x_{\text{init}} \in \mathbb{R}^n, u_{\text{init}} \in \mathbb{R}^{T \times m}$ are respectively the initial state and nominal control sequence
$C : \mathbb{R}^{n \times m} \to \mathbb{R}$ is the non-convex and twice-differentiable **cost function**.
$F : \mathbb{R}^{n \times m} \to \mathbb{R}^n$ is the non-convex and once-differentiable **dynamics function**.

$x_1^1 = x_{\text{init}}$
for t = 1 to T-1 **do**
  $x_{t+1}^1 = f(x_t, u_{\text{init},t})$
end for
$\tau^1 = [x^1, u_{\text{init}}]$

for i = 1 to *[converged]* **do**
  for t = 1 to T **do**
    ▷ Form the *second-order Taylor expansion* of the cost as in Equation (12)
    $C_t^i = \nabla^2_{\tau_t^i} C(\tau_t^i)$
    $c_t^i = \nabla_{\tau_t^i} C(\tau_t^i) - (C_t^i)^\top \tau_t^i$

    ▷ Form the *first-order Taylor expansion* of the dynamics as in Equation (13)
    $F_t^i = \nabla_{\tau_t^i} f(\tau_t^i)$
    $f_t^i = f(\tau_t^i) - F_t^i \tau_t^i$
  end for
  $\tau_{1:T}^{i+1} = \text{MPCstep}_{T,\underline{u},\overline{u}}(x_{\text{init}}, C, f, \tau_{1:T}^i, C^i, c^i, F^i, f^i)$
end for

function $\text{MPCstep}_{T,\underline{u},\overline{u}}(\, x_{\text{init}}, C, f, \tau_{1:T}, \tilde{C}, \tilde{c}, \tilde{F}, \tilde{f})$
  ▷ C, f are the *true cost* and *dynamics* functions. $\tau_{1:T}$ is the *current trajectory* iterate.
  ▷ $\tilde{C}, \tilde{c}, \tilde{F}, \tilde{f}$ are the *approximate cost* and *dynamics* terms around the current trajectory.

  ▷ **Backward Recursion:** Over the linearized trajectory.
  $V_T = v_T = 0$
  for t = T to 1 **do**
    $Q_t = \tilde{C}_t + \tilde{F}_t^\top V_{t+1} \tilde{F}_t$
    $q_t = \tilde{c}_t + \tilde{F}_t^\top V_{t+1} \tilde{f}_t + \tilde{F}_t^\top v_{t+1}$

    $k_t = \text{argmin}_{\delta u} \;\; \frac{1}{2} \delta u^\top Q_{t,uu} \delta u + Q_x^\top \delta u \;\; \text{s.t.} \;\; \underline{u} \leq u + \delta u \leq \overline{u}$
    ▷ Can be solved with a *Projected-Newton method* as described in Tassa et al. [2014].
    ▷ Let f, c respectively index the *free* and *clamped* dimensions of this optimization problem.

    $K_{t,f} = -Q_{t,uu,f}^{-1} Q_{t,ux}$
    $K_{t,c} = 0$

    $V_t = Q_{t,xx} + Q_{t,xu} K_t + K_t^\top Q_{t,ux} + K_t^\top Q_{t,uu} K_t$
    $v_t = q_{t,x} + Q_{t,xu} k_t + K_t^\top q_{t,u} + K_t^\top Q_{t,uu} k_t$
  end for

  ▷ **Forward Recursion and Line Search:** Over the true cost and dynamics.
  repeat
    $\hat{x}_1 = \tau_{x_1}$
    for t = 1 to T **do**
      $\hat{u}_t = \tau_{u_t} + \alpha k_t + K_t(\hat{x}_t - \tau_{x_t})$
      $\hat{x}_{t+1} = f(\hat{x}_t, \hat{u}_t)$
    end for
    $\alpha = \gamma \alpha$
  until $\sum_t C([\hat{x}_t, \hat{u}_t]) \leq \sum_t C(\tau_t)$

  return $\hat{x}_{1:T}, \hat{u}_{1:T}$
end function

B Imitation learning experiment losses

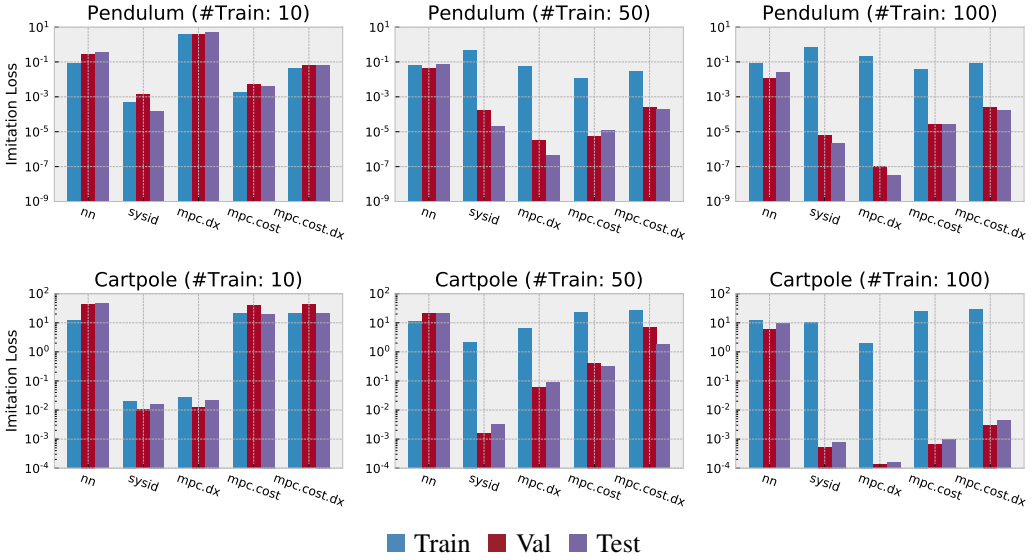

Figure 6: Learning results on the (simple) pendulum and cartpole environments. We select the best validation loss observed during the training run and report the corresponding train and test loss. Every datapoint is averaged over four trials.