Differentiable Implicit Layers

Andreas Look[1], **Simona Doneva**[2], **Melih Kandemir**[1] , **Rainer Gemulla**[2], **Jan Peters**[3],

[1]**Bosch Center for Artificial Intelligence**
Renningen, Germany
{andreas.look, melih.kandemir}@bosch.com

[2]**Data and Web Science Group**
University Mannheim, Germany
{sdoneva, rgemulla}@uni-mannheim.de

[3]**Intelligent Autonomous Systems**
TU Darmstadt, Germany
peters@ias.tu-darmstadt.de

Abstract

In this paper, we introduce an efficient backpropagation scheme for non-constrained implicit functions. These functions are parametrized by a set of learnable weights and may optionally depend on some input; making them perfectly suitable as a learnable layer in a neural network. We demonstrate our scheme on different applications: (i) neural ODEs with the implicit Euler method, and (ii) system identification in model predictive control.

1 Introduction

Implicit functions can be found in a wide range of domains, e.g. physics, numerics, or math. A famous example is Kepler's equation: $M = E - e\sin(E)$, which is elemental in orbital mechanics (see Fig 1). It estimates the relation between the eccentric anomaly E, mean anomaly M, and eccentricity e. Contrarily, learning such an implicitly defined function is not feasible with the standard deep learning practice, since it commonly consists of a chain of functional mappings described by algebraic operations. We introduce the framework of unconstrained and non-convex *Differentiable Implicit Layers* (DIL) as a plug-and-play extension for neural networks that enables efficient learning of such implicitly defined problems. An implicit layer [Gould et al., 2019] is defined as a mapping that takes an input $\boldsymbol{x} \in \mathbb{R}^{D_x}$ and produces an output $\boldsymbol{y} \in \mathbb{R}^{D_y}$ that is obtained as an argmin-solution to the scalar-valued score function $f : \mathbb{R}^{D_y + D_x + D_\theta} \to \mathbb{R}$, parameterized by $\boldsymbol{\theta} \in \mathbb{R}^{D_\theta}$:

$$\boldsymbol{y} := \underset{u}{\operatorname{argmin}}\, f(\boldsymbol{u}; \boldsymbol{x}, \boldsymbol{\theta}). \tag{1}$$

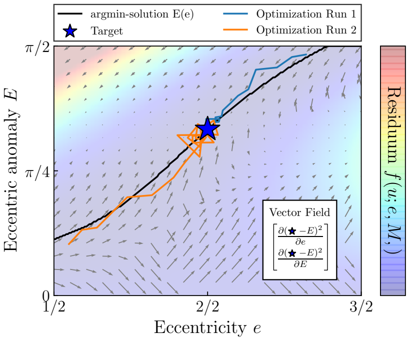


Figure 1: Kepler's Equation implicitly defines E as a function of e for a given M. We optimize e such that the implicitly defined $E(e)$ matches a target value. The IFT provides the means for estimating the implicitly defined gradient $\partial E/\partial e$. The gradient field provides information even at inexact argmin-solutions, i.e. points that are not on the argmin-line also point to the target.

We can interpret Kepler's equation as an argmin-problem with the parameters e and M: $E = \operatorname{argmin}_u (u - e\sin(u) - M; e, M)^2$. During network training, we target to optimize the parameters $\boldsymbol{\theta}$, such that the output $\boldsymbol{y}$ of the implicit layer exhibits a desired behaviour on a subsequent task, i.e.

Workshop on machine learning for engineering modeling, simulation and design @ NeurIPS 2020

minimizes a scalar loss $\mathcal{L}(\cdot)$. Consequently, we need to solve the nested argmin-problem:

$$\operatorname*{argmin}_{\theta} \mathcal{L}\Big(\operatorname*{argmin}_{u} f(\boldsymbol{u}; \boldsymbol{x}, \boldsymbol{\theta}), \boldsymbol{x}, \boldsymbol{\theta} \Big). \tag{2}$$

Note that the loss function may also depend on the parameters $\boldsymbol{\theta}$ (acting as a regularizer) and the input $\boldsymbol{x}$. Likewise, we aim to estimate in Fig. 1 the correct e such that the implicitly defined eccentric anomaly E matches a target value.

Most research on implicit layers for neural networks focuses on specific architectures [Liao et al., 2018, Bai et al., 2019] or argmin-problem classes [Amos and Kolter, 2017, Amos and Yarats, 2020], e.g. of convex type [Agrawal et al., 2019]. Concurrent work on general implicit networks [Gould et al., 2019, Zhang et al., 2020] without any restriction on problem or network type, did not scale to training of heavily parameterized implicit layers with high dimensional output. Common handicap of the aforementioned general approaches is the explicit calculation and inversion of large Jacobians. However, the existing solutions are prohibitively costly to be presented as a general purpose layer for neural networks

We propose a method that generalizes existing problem-specific solutions to a more comprehensive framework, while bringing them an unprecedented level of scalability. Our differentiable implicit layer consists of two parts: (i) the learnable argmin-problem, and (ii) the solver. The solver is used only during the forward evaluation, i.e. it does not influence the backward evaluation, by-passing a large set of potential numerical difficulties. During training, the solution $\boldsymbol{y}$ is evaluated on the downstream scalar loss function $\mathcal{L}(\cdot)$, for which we provide an efficient backward evaluation scheme by combining the *Implicit Function Theorem* (IFT) and the *Conjugate Gradient Method* (CG). In contrast to prior art, our approach omits the explicit calculation and inversion of large Jacobians, which are typically necessary for IFT evaluation. Our backward evaluation relies solely on efficient to estimate *vector-Jacobian products* (VJP). We summarize our contribution as below:

- We propose unconstrained and non-convex parameterized differentiable implicit layers for neural networks as a construct that vastly enhances the feasible problem set for the automatic differentiation technology.
- We make differentiable implicit layer training scalable for over-parameterized neural networks with a large output dimensionality.
- We demonstrate the efficiency of our method by applying it to (i) implicit solvers for neural ODEs, and (ii) model predictive control.

2 The proposed Framework

The forward evaluation of a DIL consists of applying a potentially non-differentiable solver to an argmin-problem in order to solve for $\boldsymbol{y}$ by minimizing the score function $f(\cdot\,; \boldsymbol{x}, \boldsymbol{\theta})$. The solver is used solely for the forward evaluation. Hence, we can treat the solver in our proposed framework as a blackbox. However, the main difficulty in developing an efficient framework for differentiable implicit layers lies in the backward evaluation. When $\boldsymbol{y}$ is passed on to a subsequent task, i.e. a scalar loss function $\mathcal{L}(\cdot)$, the Bi-Level IFT (Thm. 1), which is an extension to the standard IFT (see Appx. A), provides an estimate to the gradients ${}^{d\mathcal{L}(\boldsymbol{y})}/_{d\boldsymbol{x}}$, and ${}^{d\mathcal{L}(\boldsymbol{y})}/_{d\boldsymbol{\theta}}$.

Theorem 1 ***(Bi-Level IFT.)*** *Let $\boldsymbol{y}$ be the solution to a parameterized* argmin*-problem (Eq. 1). If $\boldsymbol{y}$ is evaluated on a downstream scalar loss function $\mathcal{L}(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{\theta})$ (Eq. 2), the gradients with respect to the input $\boldsymbol{x}$ (exchangeable $\boldsymbol{\theta}$) are obtained exclusively by vector-Matrix products as:*

$$\frac{d\mathcal{L}^T}{d\boldsymbol{x}} = - \underbrace{\frac{\partial \mathcal{L}^T}{\partial \boldsymbol{y}} \overbrace{\left(\frac{\partial^2 f}{\partial \boldsymbol{y}^2}\right)^{-1}}^{\boldsymbol{H}^{-1}}}_{\text{vector-inv. Hessian product} := g^T} \left(\frac{\partial^2 f}{\partial \boldsymbol{x} \partial \boldsymbol{y}}\right) + \frac{\partial \mathcal{L}^T}{\partial \boldsymbol{x}} = - \underbrace{\boldsymbol{g}^T \left(\frac{\partial^2 f}{\partial \boldsymbol{x} \partial \boldsymbol{y}}\right)}_{\text{VJP}} + \frac{\partial \mathcal{L}^T}{\partial \boldsymbol{x}}.$$

Conjugate-Gradient-Method. Explicitly inverting the Hessian $\boldsymbol{H}$ is intractable during training of a neural network, since it is computational too expensive $\propto \mathcal{O}({D_y}^3)$. Moreover modern automatic differentiation libraries lack the capability of estimating the Hessian efficiently. Instead, we directly estimate the vector-inverse Hessian product $\boldsymbol{g}$ as a solution to the *linear system of equations* (LSE):

$$\underbrace{\boldsymbol{H}\overbrace{\left(\boldsymbol{H}^{-1}\frac{\partial\mathcal{L}}{\partial\boldsymbol{y}}\right)}^{\boldsymbol{g}}}_{\text{VJP: }(\boldsymbol{g}^T\boldsymbol{H})^T}=\frac{\partial\mathcal{L}}{\partial\boldsymbol{y}}. \tag{3}$$

Since the Hessian is evaluated at a minimum, i.e. the solution $\boldsymbol{y}$ to the argmin-problem, the Hessian $\boldsymbol{H}$ is *positive semi-definite* (PSD) and the conjugate gradient method is suitable for solving the LSE. The resulting LSE can be solved via the CG method without the need of evaluating the Hessian explicitly. Each CG step requires one `grad`-function call, which estimates the *vector-Jacobian product* (VJP), and converges in the absence of round-off errors after at most D_y steps [Saad, 2003]. In contrast, the naive method of explicitly inverting the Hessian $\boldsymbol{H}$ requires firstly D_y VJP evaluations in order to build the Hessian, which are as many as CG requires for the full evaluation of the vector-inverse Hessian product $\boldsymbol{g}$. The costly inversion of the Hessian comes additionally on top.

Algorithm. We summarize our framework in Alg. 1. During the `forward`-evaluation of a DIL the score function f with optional input $\boldsymbol{x}$ is minimized with a blackbox `solver`. As a result we obtain the solution $\boldsymbol{y}$. The `backward`-evaluation receives the vector-valued gradient of the loss function $\mathcal{L}(\cdot)$ with respect to the optimal solution $\boldsymbol{y}$, i.e. $\partial\mathcal{L}(\boldsymbol{y})/\partial\boldsymbol{y}$. The gradients with respect to the parameters $\boldsymbol{\theta}$ and input $\boldsymbol{x}$ are estimated via the Bi-Level IFT (Thm. 1). The function `VJP_CG` uses a CG method, which relies on vector-Jacobian products, in order to estimate vector-inverse Hessian product $\boldsymbol{g}$ without explicitly calculating the Hessian $\boldsymbol{H}$.

Algorithm 1 Forward/ Backward Evaluation for Differentiable Implicit Layers

```
Input: Score Function $f(\cdot;\boldsymbol{x},\boldsymbol{\theta})$, Parameters $\boldsymbol{\theta}$, solver$(\cdot)$
function Forward$(\boldsymbol{x})$                                              ▷ Optional Input $\boldsymbol{x}$
    $\boldsymbol{y}$ = solver$(f(\cdot;\boldsymbol{x},\theta))$         ▷ Solve for argmin-solution $\boldsymbol{y}$ with user defined solver
    return $\boldsymbol{y}$
function Backward$(\boldsymbol{y}, \partial\mathcal{L}/\partial\boldsymbol{y})$                                   ▷ According to Thm. 2
    Set $\boldsymbol{g}_1 = \partial f/\partial\boldsymbol{y}$ , $\boldsymbol{g}_2 = \partial\mathcal{L}/\partial\boldsymbol{y}$          ▷ Score, Loss function gradient at optimal solution
    $\boldsymbol{g}$ = VJP_CG$(\boldsymbol{y},\boldsymbol{g}_1,\boldsymbol{g}_2)$          ▷ vector-inv. Hessian product $\boldsymbol{g}^T = \frac{\partial\mathcal{L}}{\partial\boldsymbol{y}}^T(\frac{\partial^2 f}{\partial\boldsymbol{y}^2})^{-1}$
    $d\mathcal{L}/d\boldsymbol{x} = -$grad$(\boldsymbol{g}_1,\boldsymbol{x},$ grad_outputs $=\boldsymbol{g})^T$          ▷ Return $-\boldsymbol{g}^T\frac{\partial^2 f}{\partial\boldsymbol{x}\partial\boldsymbol{y}}$ in $\frac{d\mathcal{L}}{d\boldsymbol{x}}^T$
    $d\mathcal{L}/d\boldsymbol{\theta} = -$grad$(\boldsymbol{g}_1,\boldsymbol{\theta},$ grad_outputs $=\boldsymbol{g})^T$          ▷ Return $-\boldsymbol{g}^T\frac{\partial^2 f}{\partial\boldsymbol{\theta}\partial\boldsymbol{y}}$ in $\frac{d\mathcal{L}}{d\boldsymbol{\theta}}^T$
    return $d\mathcal{L}/d\boldsymbol{x}$, $d\mathcal{L}/d\boldsymbol{\theta}$
function VJP_CG$(\boldsymbol{y}, \partial f/\partial\boldsymbol{y}, \partial\mathcal{L}/\partial\boldsymbol{y})$                            ▷ CG with efficient grad calls
    Init $\boldsymbol{x}_0$
    Set $\boldsymbol{r}_0 = \partial\mathcal{L}/\partial\boldsymbol{y} -$ (grad$(\partial f/\partial\boldsymbol{y},\boldsymbol{y},$ grad_outputs $=\boldsymbol{x}_0)^T + \epsilon\boldsymbol{x}_0)$     ▷ Add $\epsilon\boldsymbol{x}_0$ to tackle a singular Hessian
    Set $\boldsymbol{p}_0 = \boldsymbol{r}_0$, $k = 0$
    while $||\boldsymbol{r}_k|| > tol$ do
        $\boldsymbol{Ap}_k$ = grad$(\partial f/\partial\boldsymbol{y},\boldsymbol{y},$ grad_outputs $=\boldsymbol{p}_k)^T + \epsilon\boldsymbol{p}_k$          ▷ Symmetric Jacobian: VJP = JVP$^T$
        $\alpha_k = (\boldsymbol{r}_k^T\boldsymbol{r}_k)/(\boldsymbol{p}_k^T\boldsymbol{Ap}_k)$
        $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha\boldsymbol{p}_k$
        $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha\boldsymbol{Ap}_k$
        $\beta_k = (\boldsymbol{r}_{k+1}^T\boldsymbol{r}_{k+1})/(\boldsymbol{r}_k^T\boldsymbol{r}_k)$
        $\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k\boldsymbol{p}_k$
        $k = k + 1$
    return $\boldsymbol{x}_{k+1}$                                              ▷ Solution $\boldsymbol{g}$ in $\boldsymbol{Hg} = \frac{\partial\mathcal{L}}{\partial\boldsymbol{y}}$
```

3 Applications

If not explicitly stated otherwise, we use CG during the evaluation of the IFT (as in Alg. 1). In the first experiment we introduce implicit neural ODEs and compare our method to the adjoint training method [Chen et al., 2018]. Lastly we explore our method in the context of model predictive control.

3.1 Solving Neural ODEs with the Implicit Euler Method

Dynamical systems are commonly described by an ordinary differential equation (ODE). The commonplace way to identify a dynamical system by neural networks is the neural ODE (NODE) [Chen et al., 2018]. NODEs have been observed to introduce implementation challenges. Firstly, the adjoint training method [Chen et al., 2018] is well known to cause numerical instabilities due to non-reversibility of the NODE [Gholami et al., 2019]. Further, the backward evaluation of the adjoint requires an additional computational costly solution to the induced ODE problem. Our DIL framework is capable of addressing all of these points by introducing an implicit NODE formalism. In the following, we focus for simplicity on the backward Euler solver.

Backward Euler NODE. When solving a NODE with the backward or implicit Euler method [Hairer et al., 1993], we obtain the update rule:

$$d\boldsymbol{x} = \boldsymbol{h}(\boldsymbol{x};\boldsymbol{\theta})dt \xrightarrow[Euler]{Backward} \boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{h}(\boldsymbol{x}_{t+1};\boldsymbol{\theta})\Delta t,$$

with the state $\boldsymbol{x} \in \mathbb{R}^{D_x}$ and neural dynamical model $\boldsymbol{h} : \mathbb{R}^{D_x} \to \mathbb{R}^{D_x}$ with parameters $\boldsymbol{\theta}$. The backward Euler scheme is L-stable [Butcher, 2003] and has convergence order 1. The property of L-stability, which only implicit solvers have, allows to use larger step sizes and, above all, making the method suitable for stiff systems. Note the nuance that the backward Euler method uses $\boldsymbol{h}(\boldsymbol{x}_{t+1};\boldsymbol{\theta})$ as opposed to the forward Euler, which uses $\boldsymbol{h}(\boldsymbol{x}_t;\boldsymbol{\theta})$. Solving such an implicit problem can be translated to residual minimization:

$$\underset{x_{t+1}}{\operatorname{argmin}}\, r(\boldsymbol{x}_{t+1};\boldsymbol{x}_t,\boldsymbol{\theta}) = \underset{x_{t+1}}{\operatorname{argmin}}\, ||\boldsymbol{x}_{t+1} - (\boldsymbol{x}_t + \boldsymbol{h}(\boldsymbol{x}_{t+1};\boldsymbol{\theta})\Delta t)||. \tag{4}$$

When viewing the residual $r(\boldsymbol{x}_{t+1};\boldsymbol{x}_t,\boldsymbol{\theta})$ as the learnable score function f with parameters $\boldsymbol{\theta}$ and input $\boldsymbol{x}_t$, we obtain a DIL and can evaluate the backward pass with our proposed Alg. 1. Now it remains open how to estimate the solution $\boldsymbol{x}_{t+1}$. We obtain the solution $\boldsymbol{x}_{t+1}$ via fixed-point iteration for non-stiff problems or for stiff problems via the Newton iteration:

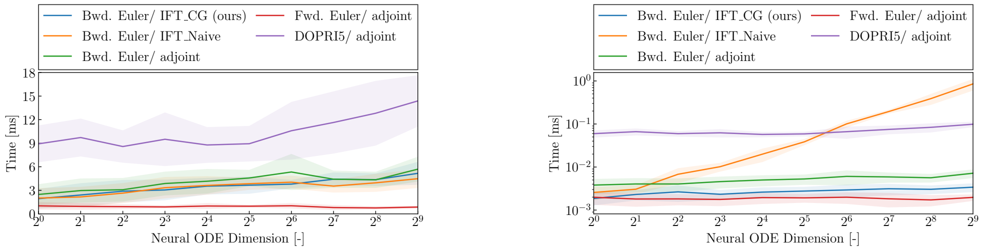
$$\boldsymbol{x}_{t+1}^{(i+1)} = \boldsymbol{x}_{t+1}^{(i)} - \underbrace{\boldsymbol{H}_r^{-1}\frac{\partial r}{\partial \boldsymbol{x}_{t+1}^{(i)}}}_{\text{inv. Hessian-vector product} := \boldsymbol{g}_r},$$

with the Hessian $\boldsymbol{H}_r$ of $r(\boldsymbol{x}_{t+1};\boldsymbol{x}_t,\boldsymbol{\theta})$. Chen and Duvenaud [2019] propose to approximate $\boldsymbol{H}_r$ by its diagonal values or the identity matrix. However it is more favourable to have an exact evaluation procedure, instead of relying on such approximations. Note that $\boldsymbol{H}_r$ is not necessary PSD, unless it is evaluated at the solution $\boldsymbol{x}_{t+1}$. Consequently, the CG method as defined in Alg. 1 is not applicable in order to estimate $\boldsymbol{g}_r$. However, we may still use the CG method if we modify the original LSE [Shewchuk, 1994] by multiplying both sides with $\boldsymbol{H}_r^T$:

$$\underbrace{\boldsymbol{H}_r \overbrace{\left(\boldsymbol{H}_r^{-1}\frac{\partial r}{\partial \boldsymbol{x}_{t+1}^{(i)}}\right)}^{\boldsymbol{g}_r}}_{\text{VJP: } (\boldsymbol{g}_r^T\boldsymbol{H}_r)^T = \tilde{\boldsymbol{g}_r}} = \frac{\partial r}{\partial \boldsymbol{x}_{t+1}^{(i)}} \xrightarrow[\text{with } \boldsymbol{H}_r^T]{\text{Multiply both sides}} \underbrace{\overbrace{\boldsymbol{H}_r^T\boldsymbol{H}_r}^{PSD}\boldsymbol{g}_r}_{\text{VJP: } (\tilde{\boldsymbol{g}_r}^T\boldsymbol{H}_r)^T} = \boldsymbol{H}_r^T\frac{\partial r}{\partial \boldsymbol{x}_{t+1}^{(i)}}.$$

Note that $\boldsymbol{H}_r^T\boldsymbol{H}_r$ is PSD and hence CG is applicable. Though the left-hand side of the modified LSE looks prohibiting at first sight, it can be evaluated efficiently by any `autodiff`-library via two `grad`-evaluations. Consequently, CG can be used with two `grad`-function calls per iteration. The backward evaluation can be performed by the IFT as proposed in Alg. 1 or alternatively with the adjoint method [Chen et al., 2018].

Runtime Profiles. Although a root finding problem (Eq. 4) needs to be solved during the forward evaluation of a NODE with the backward Euler method, it is still faster than the default adaptive step size solver DOPRI5 [Chen et al., 2018], as shown in Fig. 4a). If viewing the NODE with backward Euler solver as a DIL, we observe during the backward evaluation a significant decrease in the required computation time compared to the adjoint method (see Fig. 4b). Another benefit of the DIL viewpoint is the independence of the backward evaluation time from the NODE stiffness, which tends to increase throughout the training [Chen and Duvenaud, 2019].



(a) NODE Forward evaluation.

(b) NODE Backward evaluation.

Figure 2: Mean ± standard deviation of forward/ backward evaluation times averaged over 100 NODE initializations (Layers: 2, Hidden size: 30). Equal accept criterions of the solution $\boldsymbol{x}_{k+1}$ were used for all methods.

Predictive Performance. We benchmark the proposed Backward Euler NODE on three time series forecasting tasks. In the first experiment we generate 320 equally spaced observations according to the Van der Pol equation[1]. First 107 observations are used for training, next 106 observations for validation, and last 106 observations for testing. In the second experiment, we generate 300 equally spaced observations according to spiral dynamics [2]. We use the first 100 points for training, next 50 for validation and the final 150 for testing. In the third experiment, we follow Yildiz et al. [2019] for designing the experimental setup using data from the CMU motion capture library. The dataset is split into 16 sequences for training, three for validation, and four for test. A detailed sketch of the used architectures are given in Appx. D.1. We observe a consistent performance improvement compared to the adjoint method if the NODE discretized by backward Euler is trained with the IFT. Using the much faster CG method during the backward evaluation comes with no performance loss compared to the naive IFT evaluation.

Table 1: Average test MSE and standard errors on two extrapolation tasks: Van der Pol (20runs, 2dim, 106 step extrapolation), Spiral Data (20runs, 2dim, 150 step extrapolation), and CMU Walking (10runs, 50dim, 297 step extrapolation).

NODE Models	Van der Pol	Spiral Data	CMU Walking
DOPRI5$_{\text{adj.}}$	0.89 ± 0.15	0.13 ± 0.01	15.92 ± 2.10
Fwd. Euler$_{\text{adj.}}$	0.68 ± 0.07	0.20 ± 0.01	12.17 ± 1.39
Bwd. Euler$_{\text{adj.}}$	0.67 ± 0.12	**0.09** ± **0.01**	13.68 ± 2.02
Bwd. Euler$_{\text{IFT, Naive}}$	**0.38** ± **0.05**	**0.09** ± **0.00**	**11.57** ± **1.79**
Bwd. Euler$_{\text{IFT, CG}}$(ours)	**0.40** ± **0.06**	**0.09** ± **0.01**	**11.43** ± **1.27**

3.2 Differentiable Path Planning

We adapt the well established setup of *model predictive control* (MPC) with moving horizon [Diehl, 2011]. At each time step we observe the current state of the system $\boldsymbol{x}_{obs.}$ and plan the optimal trajectory on a limited horizon H. After planning, the first control $\boldsymbol{u}_0$ is executed and the time step is moved one step forwards. The optimization problem at the planning step can be formalized as:

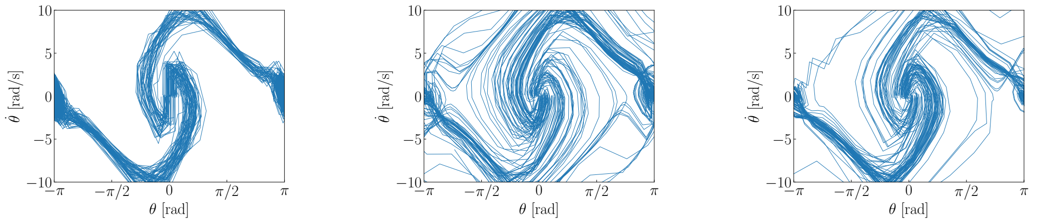
$$\underset{u_{0:H}}{\operatorname{argmin}} \sum_{t=0}^{H} c(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{\theta}_c) \ \ , s.t. \ \boldsymbol{x}_{t+1} = \boldsymbol{h}(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{\theta}_h), \ \boldsymbol{x}_0 = \boldsymbol{x}_{obs.}, \tag{5}$$

with the control $\boldsymbol{u}_t \in \mathbb{R}^{D_u}$, state $\boldsymbol{x}_t \in \mathbb{R}^{D_x}$, dynamics $\boldsymbol{h} : \mathbb{R}^{D_x} \to \mathbb{R}^{D_x}$ with parameters $\boldsymbol{\theta}_h$, and cost function $c : \mathbb{R}^{D_x + D_u} \to \mathbb{R}$ with parameters $\boldsymbol{\theta}_c$. By inserting the constraints we can interpret the optimization problem (Eq. 6) as an instance of our framework DIL. We treat the observed state $\boldsymbol{x}_{obs.}$ as the optional input and $\boldsymbol{\theta}_h$, $\boldsymbol{\theta}_c$ as the parameters of the score function. The output of this implicit layer is the control sequence $\boldsymbol{u}_{0:H}$. We can efficiently return the derivatives of the control sequence with respect to $\boldsymbol{x}_k$, $\boldsymbol{\theta}_c$, and $\boldsymbol{\theta}_h$ via Alg. 1. An alternative approach is obtained by linearizing the optimization problem. However, due to ill convergence properties, this approach did not scale to neural dynamical models [Amos et al., 2018]. In the following paragraph we provide a proof of concept that the cost function can be indeed learned by backpropagation through the trajectory planning step, when dynamics is governed by a neural network. Therefore, we interpret Eq. 6 as a

[1] $\frac{\partial^2 x}{\partial t^2} - \mu(1 - x^2)\frac{\partial x}{\partial t} + x = 0$, with $dt = 0.1$ s and $\mu = 3$

[2] $\frac{\partial \boldsymbol{x}}{\partial t} = \boldsymbol{A}\boldsymbol{x}^3$, with $dt = 0.1$ s and $a_{1,1} = -1, a_{1,2} = 2, a_{2,1} = -2, a_{2,2} = -1/10,$

DIL (MPC_{IFT}). We provide in Appx. C an additional experiment for the case of linear dynamics and cost, in which we recover true dynamics and cost using only the observed control sequence.



(a) Training Data with low variance at initial state.

(b) Behavioural cloning with high variance at initial state.

(c) MPC_{IFT} (ours) with high variance at initial state.

Figure 3: Cart pole swing-up trajectories. Ground truth from the expert (left) and learned from expert observations by behavioral cloning (middle) and our method (right).

Imitation Learning from Observations. Suppose we observe a dataset $\mathcal{D}_{exp.}$, which consists of N trajectories $\boldsymbol{x}_{1:T}^{1:N}$ with horizon T, generated by an expert policy. Note , the controls $\boldsymbol{u}_{1:T}^{1:N}$ are not observed. Let the expert policy be realized as the solution to the MPC problem as defined in Eq. 6. We target to recover the expert policy by fitting a student policy to a sequence of state transitions observed from the expert [Torabi et al., 2018, 2019]. The student policy is also evaluated as the argmin-solution to the MPC problem (Eq. 6), though with a learned cost and dynamics function. We approximate the dynamics functions with neural networks without using any prior information. The cost function evaluates the distance between the observed state and a learnable target state. In our setup we can query the true dynamical model, but do not know its functional form. The learnable dynamics function is trained on $(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{x}')$ triplets, with $\boldsymbol{x} \sim \mathcal{D}_{exp.}$, $\boldsymbol{u} \sim \mathcal{U}(\boldsymbol{u}_{min}, \boldsymbol{u}_{max})$, and $\boldsymbol{x}'$ as the true next state. The cost function is trained on the MSE between observed expert trajectories and predicted trajectories. Alg. 2 in Appx. B summarizes the learning procedure.

Imitating an Noisy Expert. We benchmark the aforementioned imitation learning method on the cartpole swing-up task. We replicate the setup from Gal et al. [2016], i.e. pole length 0.6 m, cart mass 0.5 kg, pole mass 0.5 kg, time discretization 0.1 s, and $p(\boldsymbol{x}_0) = \mathcal{N}(0, 0.04\boldsymbol{I})$. The expert dataset consists of 100 trajectories with a length of 40 steps. We evaluate the expert policy as the solution to the argmin-problem (Eq. 6) via *random shooting* (RS) [Rao, 2009]. We use a horizon of 10 steps and 1000 particles for RS. Hence, the trajectories in $\mathcal{D}_{exp.}$ are rather noisy, as shown in Fig. 3a. During training of MPC_{IFT} we initially use a prediction horizon of 1 and increase it throughout training. We compare our proposed method to *behavioral cloning* (BC) [Bain and Sammut, 1996], which learns a policy $\pi : \mathbb{R}^{D_x} \rightarrow \mathbb{R}^{D_u}$. Since we do not observe the control, we map the predicted control directly to the next state via the learnable dynamics function and minimize the MSE between future states. The details of the used network architectures are given in Appx. D.2. As shown in table 2, our method MPC_{IFT} outperforms behavioral cloning for and comes with improved generalization capabilities.

Table 2: Average cost and standard error for cartpole swingup task (50 initial positions, 10 runs). We test generalization capabilities by testing on higher variance at the initial state. $*$ [Bain and Sammut, 1996]

Model	Low Variance $x_0 \sim \mathcal{N}(0, 0.04\boldsymbol{I})$	High Variance $x_0 \sim \mathcal{N}(0, 0.08\boldsymbol{I})$
Expert	9.2 ± 0.0	(9.3 ± 0.1)
BC*	14.6 ± 0.6	18.4 ± 1.4
MPC_{IFT}(ours)	**13.7 ± 0.4**	**14.7 ± 1.9**

4 Related Work

Recurrent backpropagation (RBP) [Pineda, 1988, Almeida, 1990] is the first training method for a specific type of implicit neural networks, i.e. infinitely deep recurrent neural networks. Recent work on RBP extended this approach to efficient gradient estimation [Liao et al., 2018] or scaled it to large neural networks [Zhang et al., 2018, Bai et al., 2019]. Other lines of work focused on specific network architectures [Ghaoui et al., 2019] or argmin-problem structure, e.g. problems of convex [Agrawal et al., 2019, Wang et al., 2019] or quadratic [Amos and Kolter, 2017, Donti et al., 2017] type. Gould et al. [2019] and Zhang et al. [2020] considered constrained non-convex implicit layers as a generic building block. They proposed to evaluate the backward evaluation using the IFT. However, their work used in the implicit layers functions with symbolic second order derivatives [Gould et al., 2019] or estimated explicitly all terms [Zhang et al., 2020].

5 Scope and Limitations

In this work we have introduced the new general purpose framework of *Differentiable Implicit Layers*. For the first time implicit layers, without any restriction on problem or solution type, have been scaled to heavily parameterized neural networks with large output dimensionality. We have demonstrated our framework on a wide scope of applications. However, our framework assumes that the underlying argmin-problem can be solved accurately. If the solution is incorrect, the Bi-Level IFT (Thm. 1) does not apply anymore. It remains open up to which error tolerance convergence of an DIL can be guaranteed. Preliminary tests suggested a generous tolerance, regarding the error of the argmin-solution. Furthermore *Conjugate Gradient* methods with flexible preconditioning [Golub and Ye, 1999, Bouwmeester et al., 2015] offer a interesting perspective in order to further speed up and improve the backward evaluation of a DIL.

6 Acknowledgements

We thank Michael Tiemann and Katharina Ott for helpful discussions.

References

A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter. Differentiable Convex Optimization Layers. In *NeurIPS*. 2019.

L. B. Almeida. *A Learning Rule for Asynchronous Perceptrons with Feedback in a Combinatorial Environment*. 1990.

B. Amos and J. Z. Kolter. OptNet: Differentiable Optimization as a Layer in Neural Networks. In *ICML*. 2017.

B. Amos and D. Yarats. The Differentiable Cross-Entropy Method. In *ICML*. 2020.

B. Amos, I. D. J. Rodriguez, J. Sacks, B. Boots, and J. Z. Kolter. Differentiable MPC for End-to-End Planning and Control. In *NeurIPS*. 2018.

S. Bai, J. Z. Kolter, and V. Koltun. Deep Equilibrium Models. In *NeurIPS*. 2019.

M. Bain and C. Sammut. A Framework for Behavioural Cloning. In *Machine Intelligence*. 1996.

H. Bouwmeester, A. Dougherty, and A. V. Knyazev. Nonsymmetric Preconditioning for Conjugate Gradient and Steepest Descent Methods. *Procedia Computer Science*, 51, 2015.

J. C. Butcher. *Numerical Methods for Ordinary Differential Equations*. Wiley, 2003.

T. Q. Chen and D. Duvenaud. Neural Networks with Cheap Differential Operators. In *NeurIPS*. 2019.

T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. Neural Ordinary Differential Equations. In *NeurIPS*. 2018.

M. Diehl. *Numerical Optimal Control* . 2011.

P. Donti, B. Amos, and J. Z. Kolter. Task-based End-to-end Model Learning in Stochastic Optimization. In *NeurIPS*. 2017.

S. East, M. Gallieri, J. Masci, J. Koutnik, and M. Cannon. Infinite-Horizon Differentiable Model Predictive Control. In *ICLR*. 2020.

Y. Gal, R. McAllister, and C. E. Rasmussen. Improving PILCO with Bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, International Conference on Machine Learning*. 2016.

L. E. Ghaoui, F. Gu, B. Travacca, and A. Askari. Implicit Deep Learning. *arXiv*, abs/1908.06315, 2019.

A. Gholami, K. Keutzer, and G. Biros. ANODE: Unconditionally Accurate Memory-Efficient Gradients for NeuralODEs. In *IJCAI*. 2019.

G. H. Golub and Q. Ye. Inexact Preconditioned Conjugate Gradient Method with Inner-Outer Iteration. *SIAM J. Sci. Comput.*, 21(4), 1999.

S. Gould, R. Hartley, and D. Campbell. Deep Declarative Networks: A New Hope. *arXiv*, abs/1909.04866, 2019.

E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Springer, 1993.

R. Liao, Y. Xiong, E. Fetaya, L. Zhang, K. Yoon, X. Pitkow, R. Urtasun, and R. Zemel. Reviving and Improving Recurrent Back-Propagation. In *ICML*. 2018.

F. J. Pineda. Generalization of Back propagation to Recurrent and Higher Order Neural Networks. In *NeurIPS*. 1988.

A. V. Rao. A survey of numerical methods for optimal control. *Advances in the Astronautical Sciences*, 135, 2009.

B. Recht. A tour of reinforcement learning: The view from continuous control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2, 2019.

Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2003.

J. R. Shewchuk. An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. Technical report, 1994.

F. Torabi, G. Warnell, and P. Stone. Behavioral Cloning from Observation. In *IJCAI*. 2018.

F. Torabi, G. Warnell, and P. Stone. Generative Adversarial Imitation from Observation. In *Imitation, Intent, and Interaction Workshop at ICML*. 2019.

P. Wang, P. L. Donti, B. Wilder, and J. Z. Kolter. SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *ICML*. 2019.

C. Yildiz, M. Heinonen, and H. Lähdesmäki. ODE2VAE: Deep generative second order ODEs with Bayesian neural networks. In *NeurIPS*. 2019.

Q. Zhang, Y. Gu, M. Mateusz, M. Baktashmotlagh, and A. Eriksson. Implicitly defined layers in neural networks. *arXiv*, abs/2003.01822, 2020.

Z. Zhang, A. Kag, A. Sullivan, and V. Saligrama. Equilibrated Recurrent Neural Network: Neuronal Time-Delayed Self-Feedback Improves Accuracy and Stability. *arXiv*, abs/1903.00755, 2018.

A Implicit-Function-Theorem

Theorem 2 ***(IFT.)*** *Let $\boldsymbol{y}$ be the solution to an parametrized* argmin*-problem (Eq. 1). The gradient with respect to $\boldsymbol{x}$ (exchangeable $\boldsymbol{\theta}$) is obtained as:*

$$\frac{d\boldsymbol{y}}{d\boldsymbol{x}} = -\left(\frac{\partial^2 f(\boldsymbol{y};\boldsymbol{x},\boldsymbol{\theta})}{\partial \boldsymbol{y}^2}\right)^{-1}\left(\frac{\partial^2 f(\boldsymbol{y};\boldsymbol{x},\boldsymbol{\theta})}{\partial \boldsymbol{x}\partial \boldsymbol{y}}\right).$$

Proof.

$$\begin{aligned}
\frac{\partial f(\boldsymbol{y};\boldsymbol{x},\boldsymbol{\theta})}{\partial \boldsymbol{y}} &= 0 && \text{Since } f \text{ is evaluated at a minimum.}\\
\frac{d}{d\boldsymbol{x}}\left(\frac{\partial f(\boldsymbol{y};\boldsymbol{x},\boldsymbol{\theta})}{\partial \boldsymbol{y}}\right) &= 0 && \text{Differentiate both sides.}\\
\frac{d}{d\boldsymbol{x}}\left(\frac{\partial f(\boldsymbol{y};\boldsymbol{x},\boldsymbol{\theta})}{\partial \boldsymbol{y}}\right) &= \frac{\partial^2 f(\boldsymbol{y};\boldsymbol{x},\boldsymbol{\theta})}{\partial \boldsymbol{x}\partial \boldsymbol{y}} + \frac{\partial^2 f(\boldsymbol{y};\boldsymbol{x},\boldsymbol{\theta})}{\partial \boldsymbol{y}^2}\frac{d\boldsymbol{y}}{d\boldsymbol{x}} && \text{By Chain rule.}\\
0 &= \frac{\partial^2 f(\boldsymbol{y};\boldsymbol{x},\boldsymbol{\theta})}{\partial \boldsymbol{x}\partial \boldsymbol{y}} + \frac{\partial^2 f(\boldsymbol{y};\boldsymbol{x},\boldsymbol{\theta})}{\partial \boldsymbol{y}^2}\frac{d\boldsymbol{y}}{d\boldsymbol{x}} && \text{Both results combined.}\\
\frac{d\boldsymbol{y}}{d\boldsymbol{x}} &= -\left(\frac{\partial^2 f(\boldsymbol{y};\boldsymbol{x},\boldsymbol{\theta})}{\partial \boldsymbol{y}^2}\right)^{-1}\frac{\partial^2 f(\boldsymbol{y};\boldsymbol{x},\boldsymbol{\theta})}{\partial \boldsymbol{x}\partial \boldsymbol{y}} && \text{Final result.}
\end{aligned}$$

A.1 Bi-Level IFT

Theorem 1 (Bi-Level IFT.) Let $\boldsymbol{y}$ be the solution to an parametrized argmin-problem (Eq. 1). If $\boldsymbol{y}$ is evaluated on a downstream scalar loss function $l(\boldsymbol{y})$, the gradient with respect to $\boldsymbol{x}$ (exchangeable $\boldsymbol{\theta}$) is obtained exclusively by vector-Matrix products as:

$$\frac{d\mathcal{L}}{d\boldsymbol{x}}^T = -\underbrace{\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}}^T \overbrace{\left(\frac{\partial^2 f}{\partial \boldsymbol{y}^2}\right)^{-1}}^{\boldsymbol{H}^{-1}}}_{\text{vector-inv. Hessian product} := g}\left(\frac{\partial^2 f}{\partial \boldsymbol{x}\partial \boldsymbol{y}}\right) + \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}}^T = -\underbrace{\boldsymbol{g}^T\left(\frac{\partial^2 f}{\partial \boldsymbol{x}\partial \boldsymbol{y}}\right)}_{\text{vector-Jacobian product}} + \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}}^T.$$

Proof.

$$\begin{aligned}
\frac{d\mathcal{L}(\boldsymbol{y},\boldsymbol{x},\boldsymbol{\theta})}{d\boldsymbol{x}}^T &= \frac{\partial \mathcal{L}(\boldsymbol{y},\boldsymbol{x},\boldsymbol{\theta})}{\partial \boldsymbol{y}}^T\frac{d\boldsymbol{y}}{d\boldsymbol{x}} + \frac{\partial \mathcal{L}(\boldsymbol{y},\boldsymbol{x},\boldsymbol{\theta})}{\partial \boldsymbol{x}}^T && \text{Total Derivative}\\
\frac{d\mathcal{L}(\boldsymbol{y},\boldsymbol{x},\boldsymbol{\theta})}{d\boldsymbol{x}}^T &= -\frac{\partial \mathcal{L}(\boldsymbol{y},\boldsymbol{x},\boldsymbol{\theta})}{\partial \boldsymbol{y}}^T\left(\frac{\partial^2 f(\boldsymbol{y};\boldsymbol{x},\boldsymbol{\theta})}{\partial \boldsymbol{y}^2}\right)^{-1}\frac{\partial^2 f(\boldsymbol{y};\boldsymbol{x},\boldsymbol{\theta})}{\partial \boldsymbol{x}\partial \boldsymbol{y}} + \frac{\partial \mathcal{L}(\boldsymbol{y},\boldsymbol{x},\boldsymbol{\theta})}{\partial \boldsymbol{x}}^T && \frac{d\boldsymbol{y}}{d\boldsymbol{x}}\text{ via IFT.}
\end{aligned}$$

B Algorithm for Imitation Learning from Observations with Differentiable MPC

C Differentiable MPC for the Mass-Spring-Damper model

Background We consider a MPC controller with the cost and policy dynamics obtained by solving an unconstrained infinite-horizon Linear Quadratic Regulator (LQR). The LQR optimizes a quadratic cost function and defines linear dynamics:

$$\underset{u_{0:H}}{\text{argmin}}\sum_{t=0}^{H}\boldsymbol{x}_t^TQ\boldsymbol{x}_t + \boldsymbol{u}_t^TR\boldsymbol{u}_t \ \ , s.t.\ \boldsymbol{x}_{t+1} = A\boldsymbol{x}_t + B\boldsymbol{u}_t,\ \boldsymbol{x}_0 = \boldsymbol{x}_{obs}. \tag{6}$$

where $A \in \mathbb{R}^{n\times n}$ is the state transition matrix, $B \in \mathbb{R}^{n\times m}$ the input matrix, and $Q \in \mathbb{R}^{n\times n}$ and $R \in \mathbb{R}^{m\times m}$ are a constant state and weight matrix respectively. The optimal control action that minimizes Eq. 7 is a linear function of the state and a state feedback gain matrix $K \in \mathbb{R}^{m\times n}$ [Recht, 2019]:

$$\boldsymbol{u}_t = -K_t\boldsymbol{x}_t\ , \tag{7}$$

for K defined as:

$$K = (R + B^TSB)^{-1}B^TSA, \tag{8}$$

Algorithm 2 Imitation Learning from Observations with Differentiable MPC

```
Input: $\mathcal{D}_{exp.}$, True MDP $\boldsymbol{h}_{true}(\cdot)$, Learnable Dynamics $\boldsymbol{h}(\cdot;\boldsymbol{\theta}_h)$, Learnable Cost $c(\cdot;\boldsymbol{\theta}_c)$
function main($\boldsymbol{x}$)
    while not converged do
        train_h()
        train_c()
function train_h()
    $\boldsymbol{x} \sim \mathcal{D}_{exp.}$                                    ▷ Sample Initial State
    $\boldsymbol{u} \sim \mathcal{U}$                                           ▷ Sample Random Control
    $\boldsymbol{x}' = h_{true}(\boldsymbol{x}, \boldsymbol{u})$                ▷ Query true MDP
    $\hat{\boldsymbol{x}} = \boldsymbol{h}(\boldsymbol{u}, \boldsymbol{\theta}_h)$    ▷ Imagine next state
    $\boldsymbol{\theta}_h = \boldsymbol{\theta}_h - \texttt{lr}\nabla_{\boldsymbol{\theta}_h}||\boldsymbol{x}' - \hat{\boldsymbol{x}}||$    ▷ One Gradient Step on RMSE
function train_c()
    $\boldsymbol{x}, \boldsymbol{x}' \sim \mathcal{D}_{exp.}$                   ▷ Sample subsequent States
    Set score func. $f = \sum_{t=1}^{H} c(\boldsymbol{u}_t, \boldsymbol{h}(\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}; \boldsymbol{\theta}_h); \boldsymbol{\theta}_g)$    ▷ Initial value $\boldsymbol{h}(\boldsymbol{x}_0, \boldsymbol{u}_0; \boldsymbol{\theta}_h) = \boldsymbol{x}_{obs}$ (Eq. 6)
    $\boldsymbol{u}_{1:H}$ = Forward($\boldsymbol{x}$)                           ▷ Obtain MPC solution with Alg. 1 via RandomShooting.
    $\hat{\boldsymbol{x}} = \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}_1; \boldsymbol{\theta}_h)$    ▷ Execute first control and imagine next state.
    $\boldsymbol{\theta}_c = \boldsymbol{\theta}_c - \texttt{lr}\nabla_{\boldsymbol{\theta}_c}||\boldsymbol{x}' - \hat{\boldsymbol{x}}||$    ▷ One Gradient Step on RMSE for $\boldsymbol{\theta}_c$. Backward of Forward with Backward in Alg. 1.
```

where $S \in \mathbb{R}^{n \times n}$ satisfies the Discrete Algebraic Ricatti Equation (DARE) :

$$A^T SA - S - (A^T SB)(R + B^T SB)^{-1}(B^T SA) + Q = 0. \tag{9}$$

As the time horizon tends to infinity the value function and the optimal state feedback gains K are time-invariant. Thus for all t the control can be computed as: $\boldsymbol{u}_t = -K\boldsymbol{x}_t$, which can be obtained as a solution to the DARE.

In order to use the infinite-horizon LQR in differentiation-based learning, we need to be able to differentiate through the DARE solution. Recently it has been shown how this can be done using an analytic derivative [East et al., 2020]. Alternatively, we suggest that if we treat the DARE as the optimization problem

$$\underset{S}{\operatorname{argmin}}\, A^T SA - S - (A^T SB)(R + B^T SB)^{-1}(B^T SA) + Q \tag{10}$$

we can use the IFT to compute $\frac{\partial S}{\partial A}$, $\frac{\partial S}{\partial B}$, $\frac{\partial S}{\partial Q}$ and $\frac{\partial S}{\partial R}$. For solving the DARE we use build in `scipy` routines.

Experimental Setup The setup is inspired by the imitation learning experiments shown in East et al. [2020] and Amos et al. [2018]. The system matrices and initial input are defined as follows:

$$\mathbf{A} = \begin{bmatrix} 0.00 & 1.00 \\ -\dfrac{k}{m} & -\dfrac{c}{m} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 0.00 \\ -\dfrac{1}{m} \end{bmatrix}, \mathbf{Q} = \begin{bmatrix} 1.00 & 0.00 \\ 0.00 & 1.00 \end{bmatrix}, \mathbf{R} = 2.00,\ \mathbf{x_0} = \begin{bmatrix} 0 \\ 3 \end{bmatrix},$$

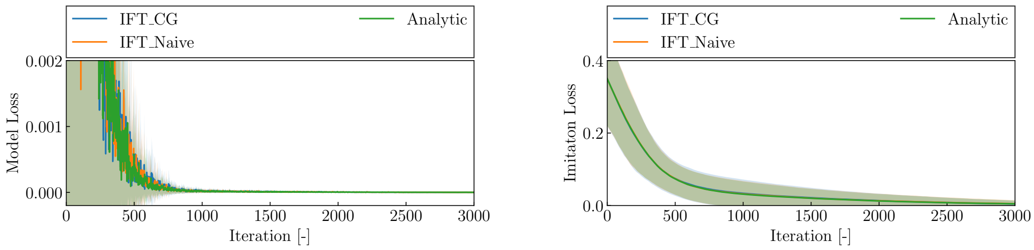
where the state variables (x_t) indicate the position and velocity of the given mass m. The parameters k and c are a stiffness parameter and a damping coefficient respectively. The values for m and k were fixed to $= 1$. The considered c values were $[1, 0.1, -0.6]$. Since the performance was similar for all values, we report results for $c = 1$ only.

The training data is generated by simulating a system for a given c value for the linear system dynamics $\boldsymbol{x}_{t+1} = A\boldsymbol{x}_t + B\boldsymbol{u}_t$. The expert matrix A was used to compute the true control matrix K and the trajectory for $\boldsymbol{x}_t$ was unrolled for a given time horizon. During this process the predicted controls $\boldsymbol{u}_t = -K\boldsymbol{x}_t$ are recorded as the "expert controls" to imitate. The first 50 elements of this trajectory were provided as the training data. At train time a starting point was selected randomly and a prediction 6 steps ahead was made with the current matrix $\hat{A}$. The learner matrix $\hat{A}$ was initialized with the correct state transition matrix plus an uniformly distributed random perturbation in the interval $[-0.5, 0.5]$ added to each element. The predicted controls were compared to the experts target controls with the goal to minimize the imitation loss:

$$\mathcal{L} = ||u_{1:T}(x; A) - u_{1:T}(x; \hat{A})||_2^2 \tag{11}$$

Note that in contrast to the previous experiment with imitation learning, here the state transitions are not available to the learner.

Results Figure 4 shows the imitation and model losses over 3000 optimization iterations. The reported *Analytic* results are obtained by our replication of the analytic gradients, as proposed in East et al. [2020]. We can see that for all initializations the imitation loss converges to a low value. Furthermore the declining model loss indicates that the learned dynamics converge to a close approximation of the true dynamics. We can also see that the IFT-CG approach closely follows the performance of the naive implementation, and they show the same learning performance as the analytic gradient.



(a) Model Loss.

(b) Imitation Loss.

Figure 4: Average and standard deviation over five different initializations. The imitation loss measures the difference between the expert and learner control values u. The model loss is computed as the L2-distance, $||A - \hat{A}||_2$, between the target expert matrix A and the learner matrix $\hat{A}$.

D Architectures

D.1 Backward Euler NODE

Van der Pol. We use a single neural network with two hidden layers.

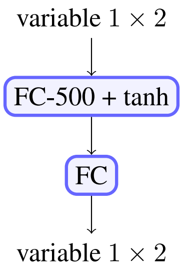


Figure 5: Neural ODE architecture for VDP.

Spiral. We use a single neural network with two hidden layers.

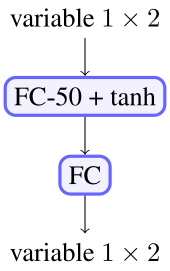


Figure 6: Neural ODE architecture for VDP.

CMU Walking. We use a similar architecture as Yildiz et al. [2019].

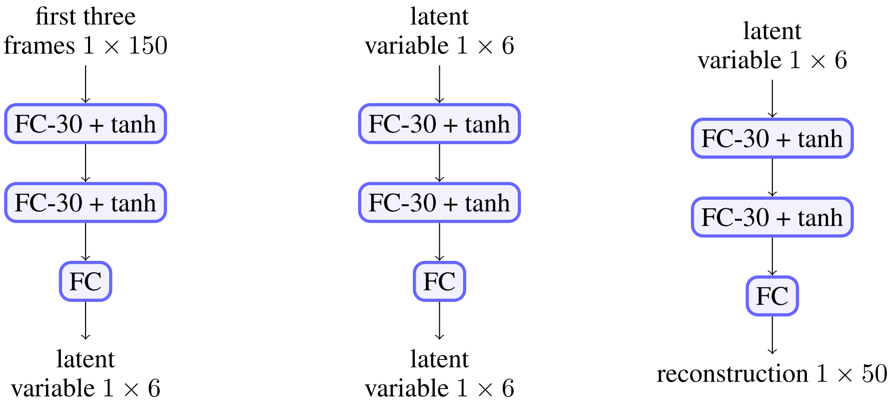
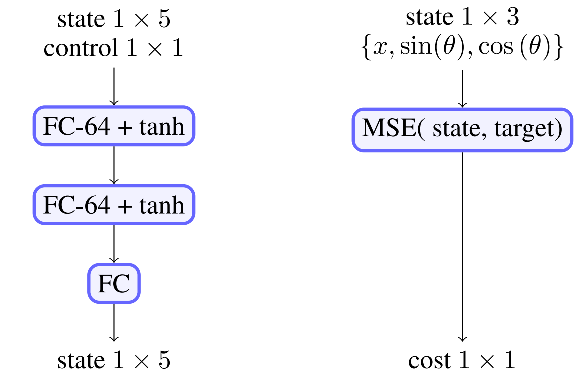


Figure 7: Encoder-NODE-Decoder neural architectures for CMU.

D.2 Differentiable MPC

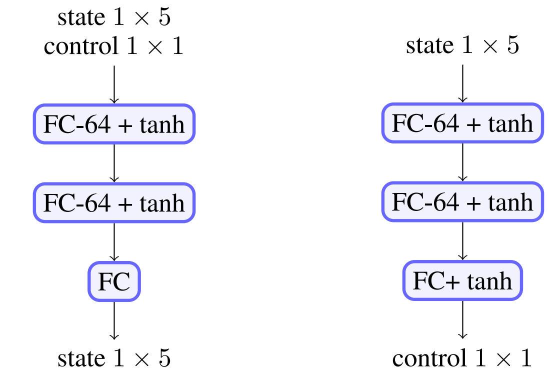
Dynamics network architecture is shared across MPC$_{\text{IFT}}$ and behavioural cloning. Admissable control was in the range $[-1, 1]$.



(a) Dynamics function.

(b) Cost function.

Figure 8: Architectures used for MPC$_{\text{IFT}}$.



(a) Dynamics function.

(b) Policy.

Figure 9: Architectures used for behavioural cloning.