Differentiable Implicit Layers

A. Look[1], S. Doneva[2], M. Kandemir[1], R. Gemulla[2], J. Peters[3]
[1]Bosch Center for Artificial Intelligence, [2]University Mannheim, [3]TU Darmstadt

1 Motivation

Implicit functions are used in a wide range of domains, e.g. physics, numerics, or math. Learning of implicitly defined functions is not feasible with the standard deep learning practice. 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.

Aim: Scalable general purpose implicit layers.

2 Implicit Layers

An implicit layer obtains an output $\boldsymbol{y}$ as an argmin-solution [1]

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

- Output $\boldsymbol{y} \in \mathbb{R}^{D_y}$
- Score function $f : \mathbb{R}^{D_y + D_x + D_\theta} \to \mathbb{R}$
- Learnable weights $\boldsymbol{\theta} \in \mathbb{R}^{D_\theta}$
- Optional input $\boldsymbol{x} \in \mathbb{R}^{D_x}$

During training the solution $\boldsymbol{y}$ is evaluated on a **scalar loss function** $\mathcal{L}(\boldsymbol{y})$. Backpropagation $\partial\mathcal{L}/\partial\boldsymbol{\theta}$ via **Implicit Function Theorem (IFT)**

$$\frac{d\mathcal{L}}{d\boldsymbol{\theta}}^T = -\frac{\partial\mathcal{L}}{\partial\boldsymbol{y}}^T \left(\frac{\partial^2 f}{\partial \boldsymbol{y}^2}\right)^{-1} \left(\frac{\partial^2 f}{\partial\boldsymbol{\theta}\partial\boldsymbol{y}}\right). \tag{2}$$

Problem: Costly inverse Hessian $\boldsymbol{H}^{-1} = \left(\partial^2 f/\partial \boldsymbol{y}^2\right)^{-1}$.

3 The proposed Framework

Instead of explicitly estimating the inverse Hessian, rather estimate the vector-inverse Hessian product

$$\boldsymbol{g}^T = \frac{\partial\mathcal{L}}{\partial\boldsymbol{y}}^T \left(\frac{\partial^2 f}{\partial \boldsymbol{y}^2}\right)^{-1}. \tag{3}$$

Obtain $\boldsymbol{g}$ as the solution to the linear system of equations

$$\boldsymbol{H}\overbrace{\left(\boldsymbol{H}^{-1}\frac{\partial\mathcal{L}}{\partial\boldsymbol{y}}\right)}^{\boldsymbol{g}} = \frac{\partial\mathcal{L}}{\partial\boldsymbol{y}}. \tag{4}$$

- Hessian $\boldsymbol{H}$ is positive semi-definite
- LSE can be solved via the **Conjugate Gradient** method [2] without evaluating the Hessian

Algorithm 1 Forward/ Backward Evaluation for a DIL

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`$(\boldsymbol{y}, \partial\mathcal{L}/\partial\boldsymbol{y})$ ▷ Via IFT
  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` $= \boldsymbol{x}_0)^T + \epsilon\boldsymbol{x}_0$)
  Set $\boldsymbol{p}_0 = \boldsymbol{r}_0$, $k = 0$
  while $||\boldsymbol{r}_k|| > tol$ **do**
    $\boldsymbol{A}\boldsymbol{p}_k =$ `grad`$(\partial f/\partial\boldsymbol{y}, \boldsymbol{y},$ `grad_outputs` $= \boldsymbol{p}_k)^T + \epsilon\boldsymbol{p}_k$ ▷ VJP $= \text{JVP}^T$
    $\alpha_k = (\boldsymbol{r}_k^T\boldsymbol{r}_k)/(\boldsymbol{p}_k^T\boldsymbol{A}\boldsymbol{p}_k)$
    $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha\boldsymbol{p}_k$
    $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha\boldsymbol{A}\boldsymbol{p}_k$
    $\beta_k = (\boldsymbol{r}_{k+1}^T\boldsymbol{r}_{k+1})/(\boldsymbol{r}_k^T\boldsymbol{r}_k)$
    $\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k\boldsymbol{p}_k$
    $k = k + 1$
  return $\boldsymbol{x}_{k+1}$ ▷ Solution $\boldsymbol{g}$ in $\boldsymbol{H}\boldsymbol{g} = \frac{\partial\mathcal{L}}{\partial\boldsymbol{y}}$

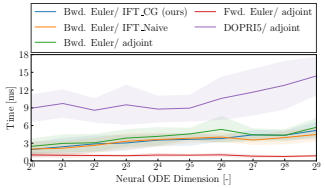
4 Applications

4.1 Implicit Neural Ordinary Differential Equations

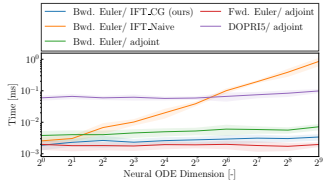
Solve a Neural Ordinary Differential Equation [3] with the backward Euler method [4] and obtain

$$d\boldsymbol{x} = \boldsymbol{h}(\boldsymbol{x};\boldsymbol{\theta})dt \xrightarrow[Euler]{Backward} \boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{h}(\boldsymbol{x}_{t+1};\boldsymbol{\theta})\Delta t. \tag{5}$$

Runtime Profiles. Forward/ backward evaluation times for random NODE initializations (Layers: 2, Hidden size: 30).



Time/ Forward evaluation.



Backward evaluation.

- Backward Euler faster than default NODE solver DOPRI5 [3]
- Backpropagation via IFT faster than adjoint method [3]

Predictive Perfomance. Average MSE on extrapolation tasks.

Model	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	$\mathbf{0.09 \pm 0.01}$	13.68 ± 2.02
Bwd. Euler$_{\text{IFT, Naive}}$	$\mathbf{0.38 \pm 0.05}$	$\mathbf{0.09 \pm 0.00}$	$\mathbf{11.57 \pm 1.79}$
Bwd. Euler$_{\text{IFT, CG}}$(ours)	$\mathbf{0.40 \pm 0.06}$	$\mathbf{0.09 \pm 0.01}$	$\mathbf{11.43 \pm 1.27}$

- Backward Euler outperforms other solvers
- Backpropagation via CG on par with naive IFT application

4.2 Differentiable Path Planning

Observe current state of the system $\boldsymbol{x}_{obs.}$ and plan optimal trajectory on a limited horizon H [5] by minimizing the total cost $\sum c$. Afterwards, execute first control $\boldsymbol{u}_0$ and move time step forwards.

$$\underset{u_{0:H}}{\operatorname{argmin}} \sum_{t=0}^{H} c(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{\theta}_c) \;\; , s.t.\; \boldsymbol{x}_{t+1} = \boldsymbol{h}(\boldsymbol{x}_t, \boldsymbol{u}_t; \boldsymbol{\theta}_h),\; \boldsymbol{x}_0 = \boldsymbol{x}_{obs.}, \tag{6}$$

Imitation Learning from Observations. Learn jointly linear cost c and neural dynamical model h by observing N expert trajectories $\boldsymbol{x}_{1:T}^{1:N}$ with horizon T on cartpole swing-up task. Train on dataset with low variance at initial state $\boldsymbol{x}_0$.

Model	Low Variance $\boldsymbol{x}_0 \sim \mathcal{N}(0, 0.04\boldsymbol{I})$	High Variance $\boldsymbol{x}_0 \sim \mathcal{N}(0, 0.08\boldsymbol{I})$
Expert	9.2 ± 0.0	(9.3 ± 0.1)
Behavioral Cloning	14.6 ± 0.6	18.4 ± 1.4
MPC$_{\text{IFT}}$(ours)	$\mathbf{13.7 \pm 0.4}$	$\mathbf{14.7 \pm 1.9}$

- Our method outperforms behavioral cloning and comes with improved generalization capabilities.

References

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

[2] J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, 1994.

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

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

[5] M. Diehl. Numerical optimal control, 2011.