Higher-Order Optimization of Neural ODEs via Optimal Control Principle

Guan-Horng Liu

Center of Machine Learning
Georgia Institute of Technology
ghliu@gatech.edu

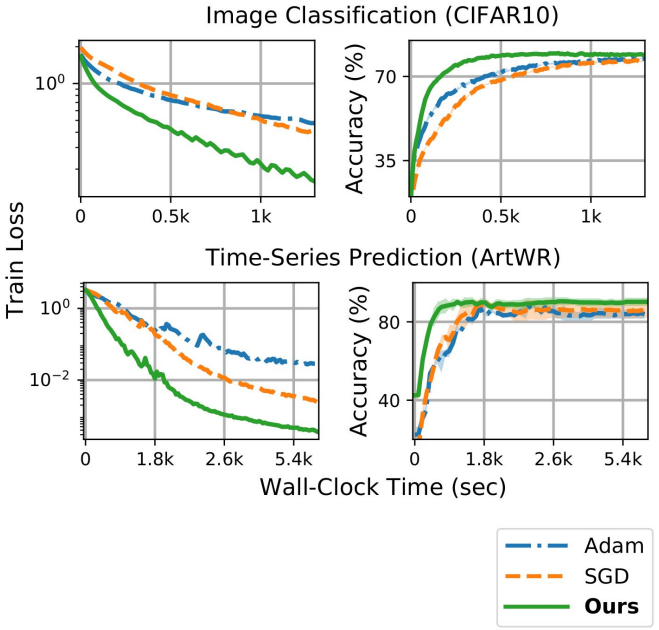
12/13 DataSig | Rough Path Interest Group

DataSig

Second-Order Neural ODE Optimizer (NeurIPS'21 spotlight)

A new optimizer for deep continuous-time models (*e.g.*, Neural ODEs) that enjoys

- Strong empirical results
  - superior convergence & test-time performance
  - hyperparameter robustness
  - architecture optimization
- Solid theoretical analysis
  - continuous-time optimal control theory
  - generalization of first-order adjoint method to higher-order at the *same **O(1)*** memory (in depth)



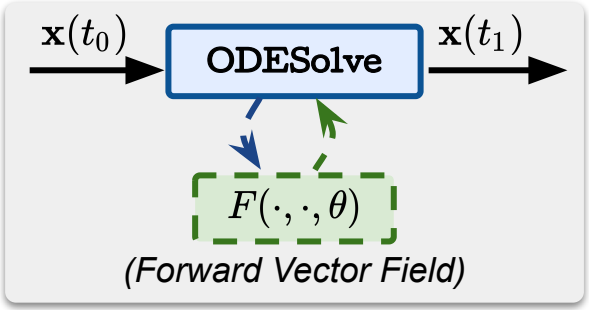
Training Process of Neural ODEs

Adjoint-based optimization

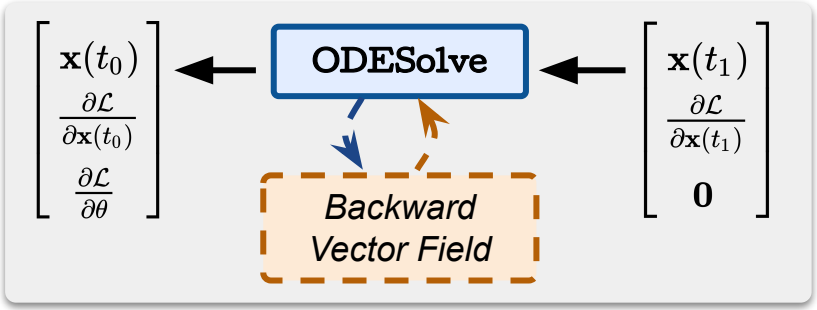
$$\min_{\theta} \mathcal{L}(\mathbf{x}(t_1)), \quad \text{s.t.} \quad \frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = F(t, \mathbf{x}(t), \theta), \quad \mathbf{x}(t_0) = \boldsymbol{x}_{t_0}$$

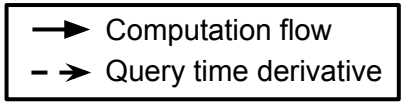
Neural ODE

Forward ODE



Backward Adjoint ODE

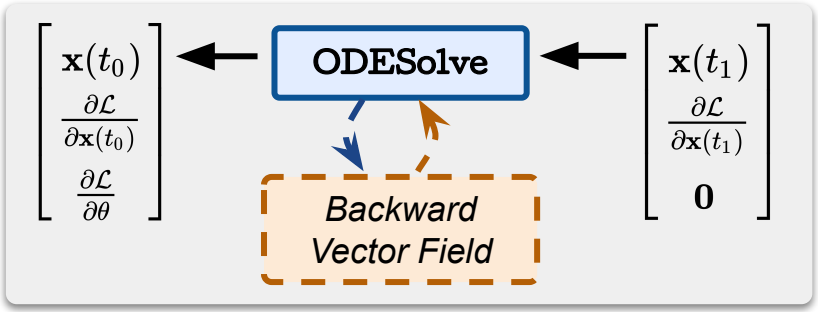




Training Process of Neural ODEs

Application to higher-order optimization (prior attempts)

Backward Adjoint ODE



$$\frac{\partial}{\partial \theta}\mathcal{L} = \mathtt{grad}(\mathcal{L}, \theta)$$

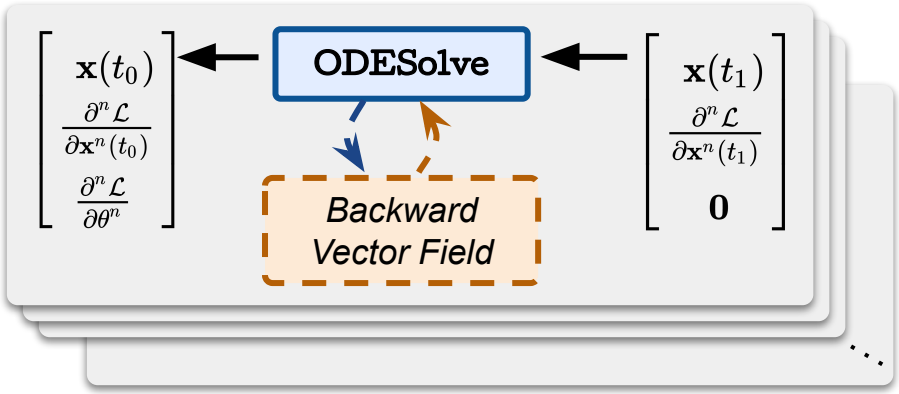
Computation flow

Query time derivative

Training Process of Neural ODEs

Application to higher-order optimization (prior attempts)

Backward Recursive Adjoint ODE



$$\frac{\partial^n \mathcal{L}}{\partial \theta^n} = \texttt{grad}\left(\frac{\partial^{n-1} \mathcal{L}}{\partial \theta^{n-1}}, \theta\right)$$

☹ linear runtime dependency

☹ accumulated integration errors

Computation flow

Query time derivative

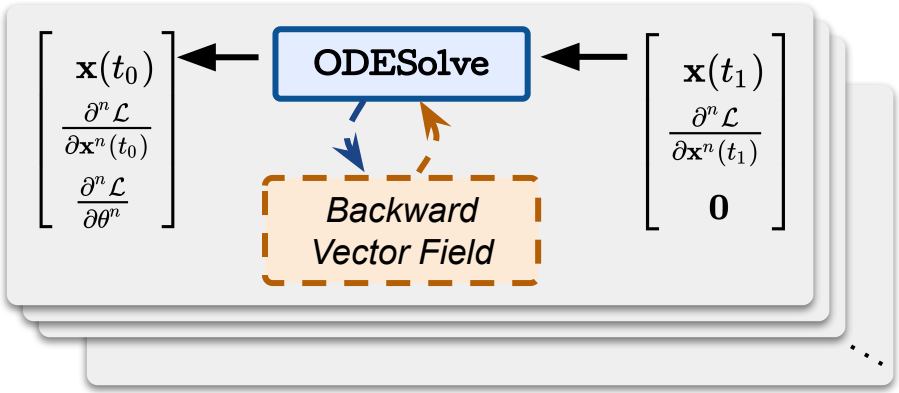
Table: Numerical errors of recursive adjoint method

	`rk4`	`implicit adams`	`dopri5`
$\frac{\partial \mathcal{L}}{\partial \theta}$	7.63×10^{-5}	2.11×10^{-3}	3.44×10^{-4}
$\frac{\partial^2 \mathcal{L}}{\partial \theta^2}$	6.83×10^{-3}	2.50×10^{-1}	44.10

Training Process of Neural ODEs

Application to higher-order optimization (prior attempts)

Backward Recursive Adjoint ODE



$$\frac{\partial^n \mathcal{L}}{\partial \theta^n} = \texttt{grad}\left(\frac{\partial^{n-1} \mathcal{L}}{\partial \theta^{n-1}}, \theta\right)$$

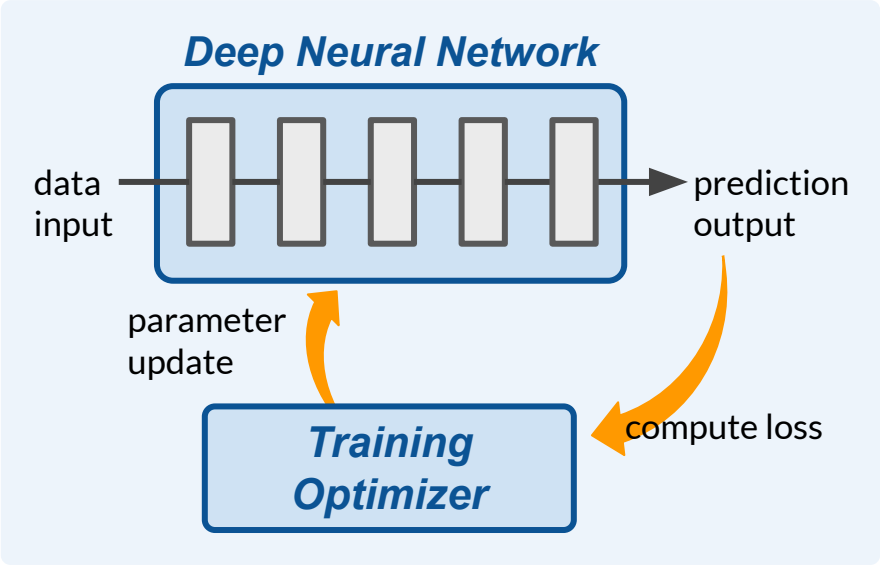
☹ linear runtime dependency

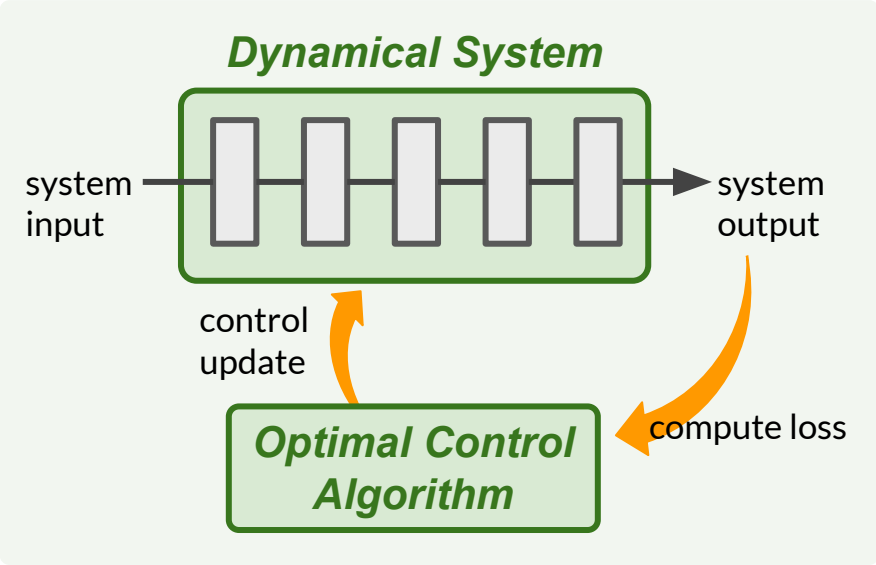
☹ accumulated integration errors

This Talk

Higher-order computational framework from an optimization viewpoint.

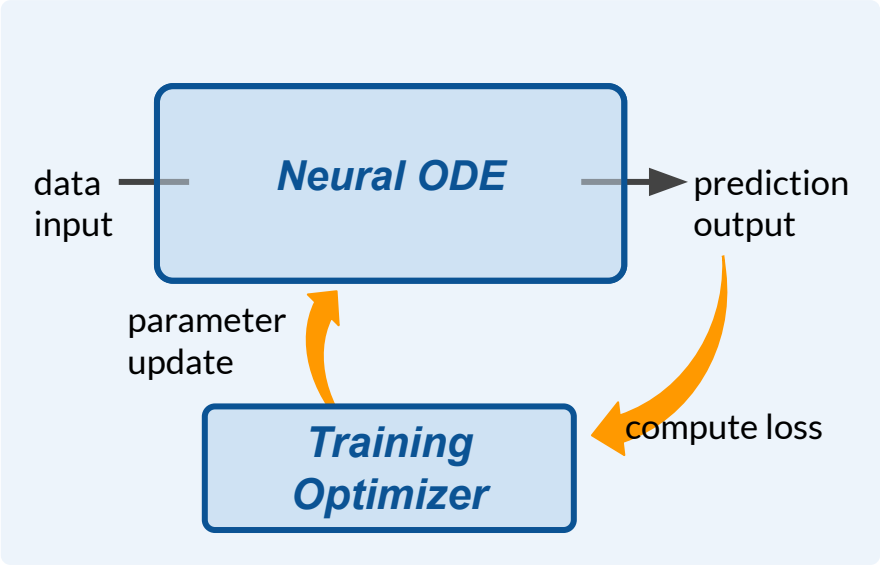
Optimal Control Perspective (OC)

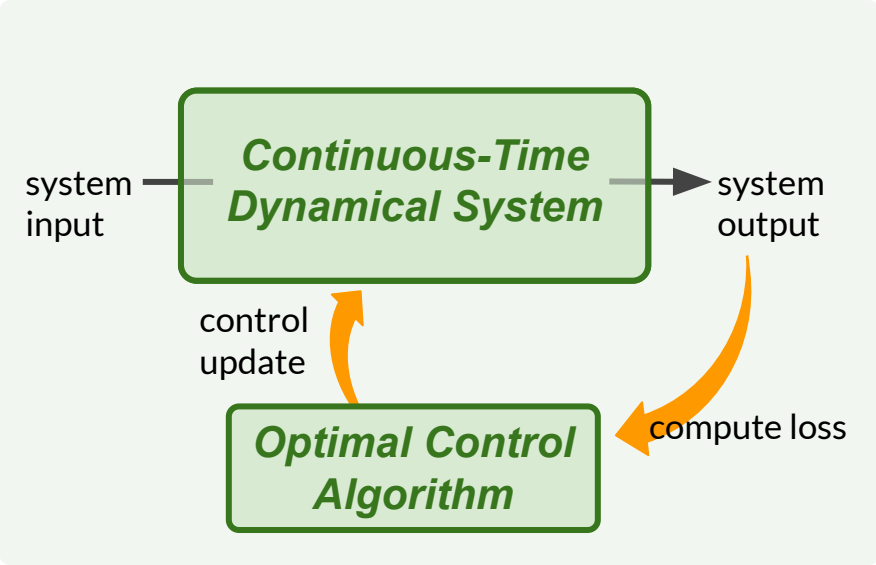




- Treat the *propagation of each layer* as a distinct *time step of a nonlinear dynamical system*.
- Interpret *layer parameter* as the *time-varying control* (*Weinan et al., 2018*; *Liu & Theodorou, 2019*).
- New optimization theory & robustifying prior methods with OC optimality (*Liu et al., 2021a,b*).

Optimal Control Perspective (OC)





- *Neural ODEs*, by construction, aim to represent *continuous-time dynamical systems*.
- *Backward adjoint ODE* originates from optimality conditions in *Optimal Control* (*Pontryagin et al., 1962*).

Training Neural ODE by Solving OCP

Original Training Process

$$\min_{\theta} \mathcal{L}(\mathbf{x}(t_1)),$$

$$\text{s.t.} \quad \frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \underbrace{F(t, \mathbf{x}(t), \theta)}_{\text{Neural ODE}}, \quad \mathbf{x}(t_0) = \boldsymbol{x}_{t_0},$$

$$(\mathcal{L}, 0) := (\Phi, \ell)$$

Optimal Control Programming (OCP)

$$\min_{\theta} \left[\Phi(\mathbf{x}_{t_1}) + \int_{t_0}^{t_1} \ell(t, \mathbf{x}_t, \mathbf{u}_t) \mathrm{d}t \right],$$

$$\text{s.t.} \begin{cases} \frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \underline{F(t, \mathbf{x}(t), \mathbf{u}(t))}, & \mathbf{x}(t_0) = \boldsymbol{x}_{t_0} \\ \frac{\mathrm{d}\mathbf{u}(t)}{\mathrm{d}t} = \mathbf{0}, & \mathbf{u}(t_0) = \theta \end{cases}$$

ODE with time-invariant control

Define accumulated loss:

$$Q(t, \mathbf{x}_t, \mathbf{u}_t) := \Phi(\mathbf{x}_{t_1}) + \int_{t}^{t_1} \ell(\tau, \mathbf{x}_\tau, \mathbf{u}_\tau) \mathrm{d}\tau$$

Adjoint-based derivatives

$$\frac{\partial \mathcal{L}}{\partial \theta}$$

$$\frac{\partial^2 \mathcal{L}}{\partial \theta \partial \theta}$$

$$\left(\frac{\partial \mathcal{L}}{\partial \theta}, \frac{\partial^2 \mathcal{L}}{\partial \theta \partial \theta}\right) := (Q_{\boldsymbol{u}}(t_0), Q_{\boldsymbol{uu}}(t_0))$$

OCP-based derivatives

$$\frac{\partial Q(t_0, \mathbf{x}_{t_0}, \mathbf{u}_{t_0})}{\partial \mathbf{u}_{t_0}} \equiv Q_{\boldsymbol{u}}(t_0)$$

$$\frac{\partial^2 Q(t_0, \mathbf{x}_{t_0}, \mathbf{u}_{t_0})}{\partial \mathbf{u}_{t_0} \partial \mathbf{u}_{t_0}} \equiv Q_{\boldsymbol{uu}}(t_0)$$

Generalization of Adjoint Process

Our goal is to solve derivatives of $Q(t, \mathbf{x}_t, \mathbf{u}_t)$ w.r.t. the control $\mathbf{u}$ at t_0 :

Theorem 1 (Second-Order Differential Programming)

The derivatives of Q expanded along a solution path $(\boldsymbol{x}_t, \boldsymbol{u}_t)$, which solves the forward ODE, obey the following set of coupled backward ODEs:

$$-\frac{\mathrm{d}Q_{\boldsymbol{x}}}{\mathrm{d}t} = \ell_{\boldsymbol{x}} + F_{\boldsymbol{x}}^{\mathsf{T}} Q_{\boldsymbol{x}} \qquad -\frac{\mathrm{d}Q_{\boldsymbol{u}}}{\mathrm{d}t} = \ell_{\boldsymbol{u}} + F_{\boldsymbol{u}}^{\mathsf{T}} Q_{\boldsymbol{x}}$$

$$-\frac{\mathrm{d}Q_{\boldsymbol{xx}}}{\mathrm{d}t} = \ell_{\boldsymbol{xx}} + F_{\boldsymbol{x}}^{\mathsf{T}} Q_{\boldsymbol{xx}} + Q_{\boldsymbol{xx}} F_{\boldsymbol{x}} \qquad -\frac{\mathrm{d}Q_{\boldsymbol{xu}}}{\mathrm{d}t} = \ell_{\boldsymbol{xu}} + F_{\boldsymbol{x}}^{\mathsf{T}} Q_{\boldsymbol{xu}} + Q_{\boldsymbol{xx}} F_{\boldsymbol{u}}$$

$$-\frac{\mathrm{d}Q_{\boldsymbol{uu}}}{\mathrm{d}t} = \ell_{\boldsymbol{uu}} + F_{\boldsymbol{u}}^{\mathsf{T}} Q_{\boldsymbol{xu}} + Q_{\boldsymbol{ux}} F_{\boldsymbol{u}} \qquad -\frac{\mathrm{d}Q_{\boldsymbol{ux}}}{\mathrm{d}t} = \ell_{\boldsymbol{ux}} + F_{\boldsymbol{u}}^{\mathsf{T}} Q_{\boldsymbol{xx}} + Q_{\boldsymbol{ux}} F_{\boldsymbol{x}}$$

} *original Adjoint*

} *generalization to second-order*

- 😀 no recursive dependency! (everything's solved in *one single backward pass*)
- 😀 can be extended to computing higher-order derivatives (*e.g.*, 3rd-order tensors)

Derivation Roadmap

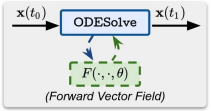
Derivation

1st-order adjoint method

Higher-order adjoint process (Theorem 1)

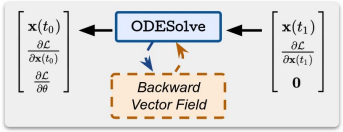
Forward ODE

$$\mathbf{x}(t_0) = \textbf{ODESolve}\left(t_0, t_1, \mathbf{x}(t_0), F\right)$$



Backward Adjoint ODE (O(1) memory)

$$[\mathbf{x}(t_0), \frac{\partial \mathcal{L}}{\partial \mathbf{x}(t_0)}, \frac{\partial \mathcal{L}}{\partial \theta}]^\mathsf{T} = \textbf{ODESolve}\left(t_1, t_0, [\mathbf{x}(t_1), \frac{\partial \mathcal{L}}{\partial \mathbf{x}(t_1)}, \mathbf{0}]^\mathsf{T}, G\right)$$



Derivation Roadmap

Derivation

1st-order adjoint method

↓

Higher-order adjoint process (Theorem 1)

Forward ODE

$$\mathbf{x}(t_0) = \mathbf{ODESolve}(t_0, t_1, \mathbf{x}(t_0), F)$$

Backward Adjoint ODE (O(1) memory)

$$[\mathbf{x}(t_0), \tfrac{\partial \mathcal{L}}{\partial \mathbf{x}(t_0)}, \tfrac{\partial \mathcal{L}}{\partial \theta}]^\mathsf{T} = \mathbf{ODESolve}(t_1, t_0, [\mathbf{x}(t_1), \tfrac{\partial \mathcal{L}}{\partial \mathbf{x}(t_1)}, \mathbf{0}]^\mathsf{T}, G)$$

☹ *coupled matrix ODEs are too expensive...*

↓

Higher-Order Backward Adjoint ODE (O(1) memory)

$$[\mathbf{x}(t_0), Q_{\boldsymbol{x}}(t_0), Q_{\boldsymbol{u}}(t_0), Q_{\boldsymbol{xx}}(t_0), Q_{\boldsymbol{xu}}(t_0), Q_{\boldsymbol{ux}}(t_0), Q_{\boldsymbol{uu}}(t_0)]^\mathsf{T}$$
$$= \mathbf{ODESolve}(t_1, t_0, [\underbrace{\mathbf{x}(t_1), \Phi_{\boldsymbol{x}}, \mathbf{0}}_{\text{1st-order}}, \underbrace{\Phi_{\boldsymbol{xx}}, \mathbf{0}, \mathbf{0}, \mathbf{0}}_{\text{2nd-order}}]^\mathsf{T}, \tilde{G})$$

from Theorem 1

Efficient Higher-Order Computation: Step 1

Derivation

1st-order adjoint method

↓

Higher-order adjoint process (Theorem 1)

↓

Low-rank representation (Proposition 2)

- Disentangle the *coupled matrix ODEs* in Theorem 1 into a set of *independent vector ODEs*.

Proposition 2 (Low-rank representation)

Suppose $\ell := 0$ *and let* $Q_{\boldsymbol{xx}}(t_1) = \sum_{i=1}^{R} \boldsymbol{y}_i \otimes \boldsymbol{y}_i$ *be a symmetric matrix of rank* R*. Then, we have the following decompositions:*

$$Q_{\boldsymbol{xx}}(t) = \sum_{i=1}^{R} \mathbf{q}_i(t) \otimes \mathbf{q}_i(t), \quad Q_{\boldsymbol{xu}}(t) = \sum_{i=1}^{R} \mathbf{q}_i(t) \otimes \mathbf{p}_i(t), \quad Q_{\boldsymbol{uu}}(t) = \sum_{i=1}^{R} \mathbf{p}_i(t) \otimes \mathbf{p}_i(t)$$

where the vectors $(\mathbf{q}_i(t), \mathbf{p}_i(t))$ *obey the following backward ODEs:*

$$-\frac{\mathrm{d}\mathbf{q}_i(t)}{\mathrm{d}t} = F_{\boldsymbol{x}}^{\mathsf{T}} \mathbf{q}_i(t), \quad -\frac{\mathrm{d}\mathbf{p}_i(t)}{\mathrm{d}t} = F_{\boldsymbol{u}}^{\mathsf{T}} \mathbf{q}_i(t)$$

with the terminal conditions given by $(\mathbf{q}_i(t_1), \mathbf{p}_i(t_1)) := (\boldsymbol{y}_i, \mathbf{0})$ *.*

Efficient Higher-Order Computation: Step 1

Derivation

1st-order adjoint method

↓

Higher-order adjoint process (Theorem 1)

↓

Low-rank representation (Proposition 2)

Higher-Order Backward Adjoint ODE (matrix form)

$$[\mathbf{x}(t_0), Q_{\boldsymbol{x}}(t_0), Q_{\boldsymbol{u}}(t_0), Q_{\boldsymbol{xx}}(t_0), Q_{\boldsymbol{xu}}(t_0), Q_{\boldsymbol{ux}}(t_0), Q_{\boldsymbol{uu}}(t_0)]^\mathsf{T}$$
$$= \textbf{ODESolve}\big(\, t_1, t_0, [\mathbf{x}(t_1), \Phi_{\boldsymbol{x}}, \mathbf{0}, \Phi_{\boldsymbol{xx}}, \mathbf{0}, \mathbf{0}, \mathbf{0}]^\mathsf{T}, \tilde{G}\big)$$

Higher-Order Backward Adjoint ODE (low rank)

$$[\mathbf{x}(t_0), Q_{\boldsymbol{x}}(t_0), Q_{\boldsymbol{u}}(t_0), \{\mathbf{q}_i(t_0)\}_{i=1}^R, \{\mathbf{p}_i(t_0)\}_{i=1}^R]^\mathsf{T}$$
$$= \textbf{ODESolve}\big(\, t_1, t_0, [\mathbf{x}(t_1), \Phi_{\boldsymbol{x}}, \mathbf{0}, \{\boldsymbol{y}\}_{i=1}^R, \mathbf{0}]^\mathsf{T}, \tilde{\tilde{G}}\big)$$

from Proposition 2

⟹ recover precondition matrix: $\mathcal{L}_{\theta\theta} \equiv Q_{\boldsymbol{uu}}(t_0) = \sum_{i=1}^{R} \mathbf{p}_i(t_0) \otimes \mathbf{p}_i(t_0)$

⟹ compute preconditioned update: $\mathcal{L}_\theta \leftarrow \mathcal{L}_{\theta\theta}^{-1} \mathcal{L}_\theta$

😀 *efficient backward pass!*
☹️ *preconditioning is still quite expensive...*

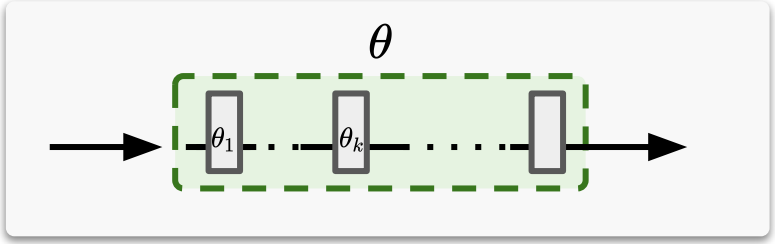
Efficient Higher-Order Computation: Step 2

Derivation

- *1st-order adjoint method*
- *Higher-order adjoint process (Theorem 1)*
- *Low-rank representation (Proposition 2)*
- *Kronecker-factored preconditioned update*

$$\mathcal{L}_{\theta\theta} \approx \begin{bmatrix} \ddots & & \\ & \mathcal{L}_{\theta_k\theta_k} & \\ & & \ddots \end{bmatrix}, \text{ where } \mathcal{L}_{\theta_k\theta_k} \approx \bar{\boldsymbol{A}}_k \otimes \bar{\boldsymbol{B}}_k$$

$$\mathcal{L}_{\theta\theta}^{-1}\mathcal{L}_{\theta} \approx \mathcal{L}_{\theta_k\theta_k}^{-1}\mathcal{L}_{\theta_k} = \mathrm{vec}(\bar{\boldsymbol{B}}_k^{-1}\mathcal{L}_{\theta_k}\bar{\boldsymbol{A}}_k^{-\mathsf{T}})$$



Efficient Higher-Order Computation: Step 2

Derivation

1st-order adjoint method

↓

Higher-order adjoint process (Theorem 1)

↓

Low-rank representation (Proposition 2)

↓

Kronecker-factored preconditioned update

(i) expand the precondition matrix with the ODE in Proposition 2

$$\mathcal{L}_{\theta\theta} = \sum_{i=1}^{R} \mathbf{p}_i(t_0) \otimes \mathbf{p}_i(t_0) = \sum_{i=1}^{R} \left(\int_{t_1}^{t_0} F_\theta^\mathsf{T} \mathbf{q}_i \, \mathrm{d}t \right) \otimes \left(\int_{t_1}^{t_0} F_\theta^\mathsf{T} \mathbf{q}_i \, \mathrm{d}t \right),$$

Efficient Higher-Order Computation: Step 2

Derivation

1st-order adjoint method

↓

Higher-order adjoint process (Theorem 1)

↓

Low-rank representation (Proposition 2)

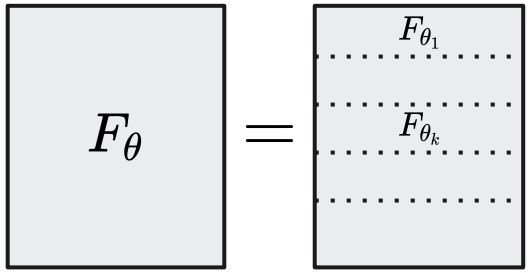
↓

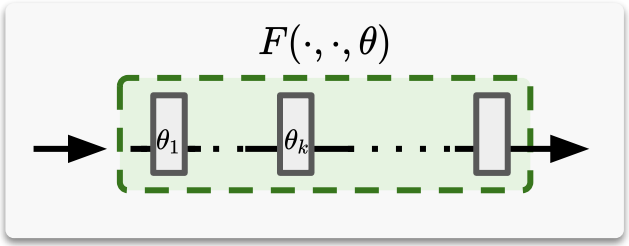
Kronecker-factored preconditioned update

(ii) *layer-wise* precondition matrix.

$$\mathcal{L}_{\theta\theta} = \sum_{i=1}^{R} \mathbf{p}_i(t_0) \otimes \mathbf{p}_i(t_0) = \sum_{i=1}^{R} \left(\int_{t_1}^{t_0} F_\theta^\mathsf{T} \mathbf{q}_i \, \mathrm{d}t \right) \otimes \left(\int_{t_1}^{t_0} F_\theta^\mathsf{T} \mathbf{q}_i \, \mathrm{d}t \right),$$

$$\mathcal{L}_{\theta_k\theta_k} = \sum_{i=1}^{R} \left(\int_{t_1}^{t_0} F_{\theta_k}^\mathsf{T} \mathbf{q}_i \, \mathrm{d}t \right) \otimes \left(\int_{t_1}^{t_0} F_{\theta_k}^\mathsf{T} \mathbf{q}_i \, \mathrm{d}t \right)$$





Efficient Higher-Order Computation: Step 2

Derivation

1st-order adjoint method

↓

Higher-order adjoint process (Theorem 1)

↓

Low-rank representation (Proposition 2)

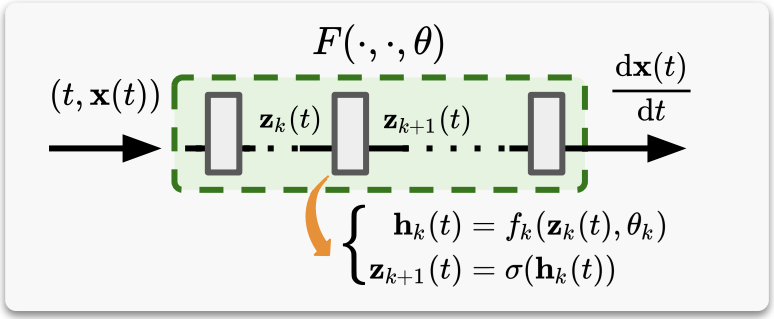
↓

Kronecker-factored preconditioned update

(iii) exploit the Kronecker factorization of JVP.

$$\mathcal{L}_{\theta\theta} = \sum_{i=1}^{R} \mathbf{p}_i(t_0) \otimes \mathbf{p}_i(t_0) = \sum_{i=1}^{R} \left(\int_{t_1}^{t_0} F_\theta^\mathsf{T} \mathbf{q}_i \, \mathrm{d}t \right) \otimes \left(\int_{t_1}^{t_0} F_\theta^\mathsf{T} \mathbf{q}_i \, \mathrm{d}t \right),$$

$$\begin{aligned} \mathcal{L}_{\theta_k\theta_k} &= \sum_{i=1}^{R} \left(\int_{t_1}^{t_0} F_{\theta_k}^\mathsf{T} \mathbf{q}_i \, \mathrm{d}t \right) \otimes \left(\int_{t_1}^{t_0} F_{\theta_k}^\mathsf{T} \mathbf{q}_i \, \mathrm{d}t \right) \\ &= \sum_{i=1}^{R} \left(\int_{t_1}^{t_0} \left(\mathbf{z}_k \otimes \left(\frac{\partial F}{\partial \mathbf{h}_k}^\mathsf{T} \mathbf{q}_i \right) \right) \mathrm{d}t \right) \otimes \left(\int_{t_1}^{t_0} \left(\mathbf{z}_k \otimes \left(\frac{\partial F}{\partial \mathbf{h}_k}^\mathsf{T} \mathbf{q}_i \right) \right) \mathrm{d}t \right) \end{aligned}$$



Efficient Higher-Order Computation: Step 2

Derivation

- ***1st-order adjoint method***
- ***Higher-order adjoint process (Theorem 1)***
- ***Low-rank representation (Proposition 2)***
- ***Kronecker-factored preconditioned update***

(iv) adopt formula of Kronecker product and few approximations.

$$\mathcal{L}_{\theta\theta} = \sum_{i=1}^{R} \mathbf{p}_i(t_0) \otimes \mathbf{p}_i(t_0) = \sum_{i=1}^{R} \left(\int_{t_1}^{t_0} F_\theta^\mathsf{T} \mathbf{q}_i \mathrm{d}t \right) \otimes \left(\int_{t_1}^{t_0} F_\theta^\mathsf{T} \mathbf{q}_i \mathrm{d}t \right),$$

$$\begin{aligned}
\mathcal{L}_{\theta_k\theta_k} &= \sum_{i=1}^{R} \left(\int_{t_1}^{t_0} F_{\theta_k}^\mathsf{T} \mathbf{q}_i \mathrm{d}t \right) \otimes \left(\int_{t_1}^{t_0} F_{\theta_k}^\mathsf{T} \mathbf{q}_i \mathrm{d}t \right) \\
&= \sum_{i=1}^{R} \left(\int_{t_1}^{t_0} \left(\mathbf{z}_k \otimes \left(\frac{\partial F}{\partial \mathbf{h}_k}^\mathsf{T} \mathbf{q}_i \right) \right) \mathrm{d}t \right) \otimes \left(\int_{t_1}^{t_0} \left(\mathbf{z}_k \otimes \left(\frac{\partial F}{\partial \mathbf{h}_k}^\mathsf{T} \mathbf{q}_i \right) \right) \mathrm{d}t \right) \\
&\approx \underbrace{\int_{t_1}^{t_0} (\mathbf{z}_k \otimes \mathbf{z}_k) \, \mathrm{d}t}_{\boldsymbol{A}_k(t)} \otimes \underbrace{\int_{t_1}^{t_0} \sum_{i=1}^{R} \left(\left(\frac{\partial F}{\partial \mathbf{h}_k}^\mathsf{T} \mathbf{q}_i \right) \otimes \left(\frac{\partial F}{\partial \mathbf{h}_k}^\mathsf{T} \mathbf{q}_i \right) \right) \mathrm{d}t}_{\boldsymbol{B}_k(t)}
\end{aligned}$$

$\Longrightarrow$ layer-wise precondition matrix: $\mathcal{L}_{\theta_k\theta_k} \approx \bar{\boldsymbol{A}}_k \otimes \bar{\boldsymbol{B}}_k$, where $\begin{cases} \bar{\boldsymbol{A}}_k = \sum_j \boldsymbol{A}_k(t_j) \cdot \Delta t \\ \bar{\boldsymbol{B}}_k = \sum_j \boldsymbol{B}_k(t_j) \cdot \Delta t \end{cases}$

Efficient Higher-Order Computation: Step 2

Derivation

1st-order adjoint method

↓

Higher-order adjoint process (Theorem 1)

↓

Low-rank representation (Proposition 2)

↓

Kronecker-factored preconditioned update

Higher-Order Adjoint ODE (low rank)

$$[\mathbf{x}(t_0), Q_{\boldsymbol{x}}(t_0), Q_{\boldsymbol{u}}(t_0), \{\mathbf{q}_i(t_0)\}_{i=1}^R, \{\mathbf{p}_i(t_0)\}_{i=1}^R]^\mathsf{T} = \textbf{ODESolve}\big(t_1, t_0, [\mathbf{x}(t_1), \Phi_{\boldsymbol{x}}, \mathbf{0}, \{\boldsymbol{y}\}_{i=1}^R, \mathbf{0}]^\mathsf{T}, \widetilde{\widetilde{G}}\big)$$

Higher-Order Adjoint ODE (low rank + Kronecker)

$$[\mathbf{x}(t_0), Q_{\boldsymbol{x}}(t_0), Q_{\boldsymbol{u}}(t_0), \{\mathbf{q}_i(t_0)\}_{i=1}^R]^\mathsf{T} = \textbf{ODESolve}\big(t_1, t_0, [\mathbf{x}(t_1), \Phi_{\boldsymbol{x}}, \mathbf{0}, \{\boldsymbol{y}\}_{i=1}^R]^\mathsf{T}, \widehat{G}\big)$$

Computation Comparison to 1st-Order Baseline

Derivation

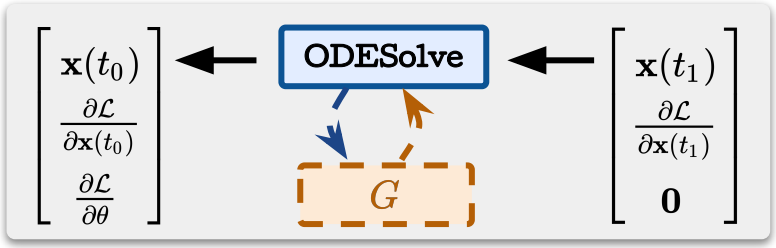
1st-order adjoint method

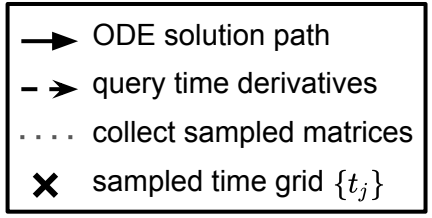
Higher-order adjoint process (Theorem 1)

Low-rank representation (Proposition 2)

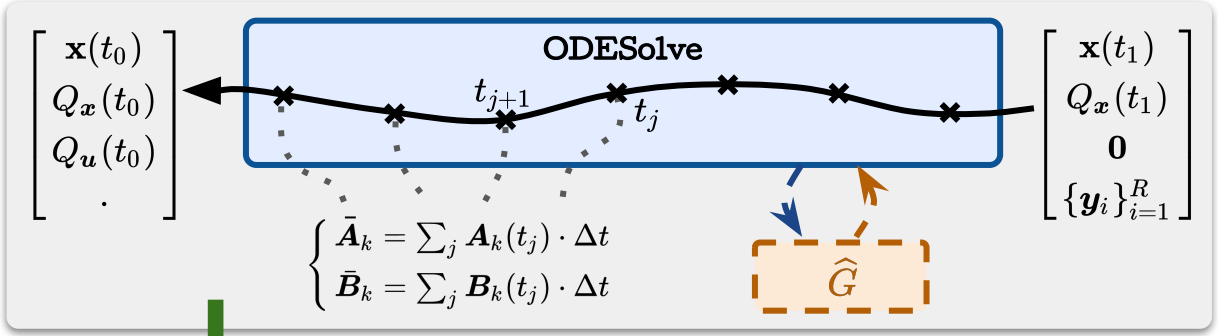
Second-Order Neural ODE Optimizer (SNOpt)

$$[\mathbf{x}(t_0), \tfrac{\partial \mathcal{L}}{\partial \mathbf{x}(t_0)}, \tfrac{\partial \mathcal{L}}{\partial \theta}]^\top = \textbf{ODESolve}\big(t_1, t_0, [\mathbf{x}(t_1), \tfrac{\partial \mathcal{L}}{\partial \mathbf{x}(t_1)}, \mathbf{0}]^\top, G\big)$$





$$[\mathbf{x}(t_0), Q_{\boldsymbol{x}}(t_0), Q_{\boldsymbol{u}}(t_0), \{\mathbf{q}_i(t_0)\}_{i=1}^R]^\top = \textbf{ODESolve}\big(t_1, t_0, [\mathbf{x}(t_1), \Phi_{\boldsymbol{x}}, \mathbf{0}, \{\boldsymbol{y}\}_{i=1}^R]^\top, \widehat{G}\big)$$



$$\mathcal{L}_{\theta_k\theta_k}^{-1} \mathcal{L}_{\theta_k} = \mathrm{vec}(\bar{\boldsymbol{B}}_k^{-1} Q_{\mathbf{u}_k}(t_0) \bar{\boldsymbol{A}}_k^{-\top})$$

Computation Comparison at Each Stage

Derivation

1st-order adjoint method

↓

Higher-order adjoint process (Theorem 1)

↓

Low-rank representation (Proposition 2)

↓

Second-Order Neural ODE Optimizer (SNOpt)

- Memory complexity ($\mathbf{x}_t \in \mathbb{R}^m, \mathbf{u}_t, \theta \in \mathbb{R}^n$, rank R)

	1st-order	*Theorem 1*	*Proposition 2*	*SNOpt*
Adjoint backward	$\mathcal{O}(1)$ $\mathcal{O}(n+m)$	$\mathcal{O}(1)$ $\mathcal{O}((n+m)^2)$	$\mathcal{O}(1)$ $\mathcal{O}(Rn+Rm)$	$\mathcal{O}(1)$ $\mathcal{O}(2n+Rm)$
Param. update	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(2n)$

  - 10-40% additional *constant* memory compared to 1st-order Adjoint. (less than 1GB on all experiments)

Results: Computational Efficiency

- Per-iteration runtime (seconds) w.r.t. Adam

	Image Classification			Time-series Prediction			Continuous NF		
	MNIST	SVHN	CIFAR10	SpoAD	ArtWR	CharT	Circle	Gas	Minib.
$\frac{\text{Ours}}{\text{Adam}}$	1.00	0.87	1.16	0.99	1.01	1.01	2.75	1.93	1.60

😀 run nearly as fast as Adam!

😲 may run *faster* (*e.g.*, SVHN)!

😐 1.5~3x on CNF datasets

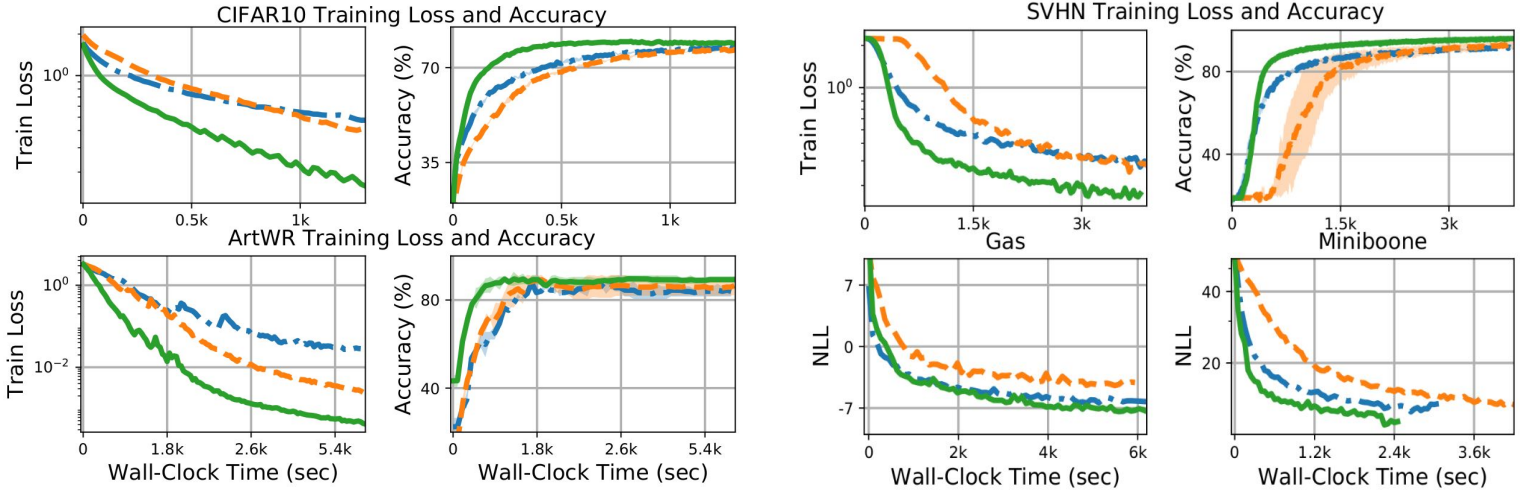
😀 still faster convergence (next slide)

- Preconditioned updates may lead to implicit regularization.

	# of function evaluation (NFE)		*Regularization* (*Finlay et al., 2020*)
	forward pass	*backward pass*	($\int \|\|\nabla_{\mathbf{x}} F\|\|^2 + \int \|\|F\|\|^2$)
Adam	*32.0*	*42.1*	*323.9 (100%)*
Ours	***26.0***	***32.6***	***199.1 (61.5%)***

Results: Computational Efficiency

- Superior convergence in wall-clock time (**Adam**, **SGD**, **SNOpt (*ours*)**)

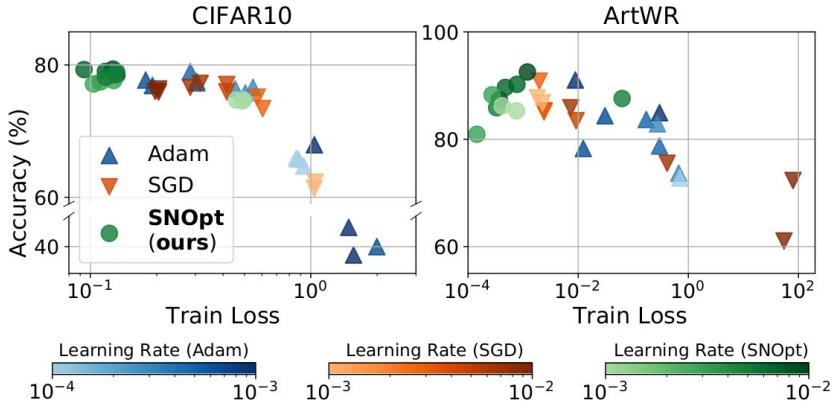


Results: Improvement & Robustness

- Improves test-time performance (accuracy for Image/Time-series, NLL for CNF)

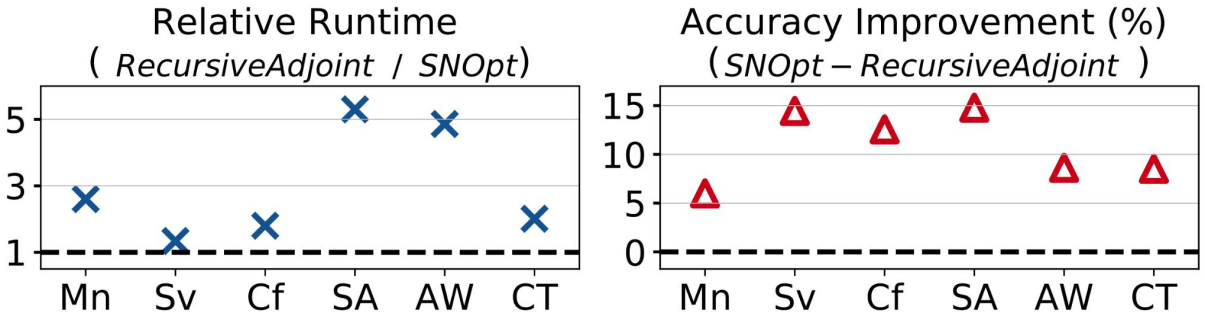
	MNIST	SVHN	CIFAR10	SpoAD	ArtWR	CharT	Circle	Gas	Miniboone
Adam	98.83	91.92	77.41	94.64	84.14	93.29	0.90	-6.42	13.10
SGD	98.68	93.34	76.42	**97.70**	85.82	95.93	0.94	-4.58	13.75
SNOpt	**98.99**	**95.77**	**79.11**	97.41	**90.23**	**96.63**	**0.86**	**-7.55**	**12.50**

- Robustifies hyper-parameter sensitivity.



Results: Comparison to Recursive Baseline

- SNOpt is around 2~5x faster than recursive adjoint baseline.
- Improve test-time accuracy by 5~15%.



Application to Architecture Optimization

Consider an extension of $Q(t, \mathbf{x}_t, \mathbf{u}_t)$ that includes the terminal horizon t_1.

$$Q(t, \mathbf{x}_t, \mathbf{u}_t) := \Phi(\mathbf{x}_{t_1}) + \int_t^{t_1} \ell(\tau, \mathbf{x}_\tau, \mathbf{u}_\tau) \mathrm{d}\tau$$

$\widetilde{\Phi} := \Phi +$ penalty on T

$$\widetilde{Q}(t, \mathbf{x}_t, \mathbf{u}_t, \mathrm{T}) := \widetilde{\Phi}(\mathrm{T}, \mathbf{x}_\mathrm{T}) + \int_t^{\mathrm{T}} \ell(\tau, \mathbf{x}_\tau, \mathbf{u}_\tau) \mathrm{d}\tau$$

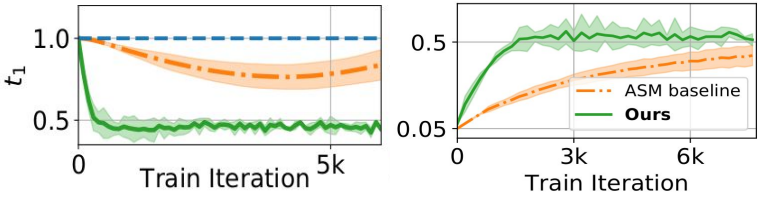
iterative ***feedback*** update rule

$$\mathrm{T} \leftarrow T - \delta\mathrm{T}(\delta\theta) \text{ , where}$$

$$\delta\mathrm{T}(\delta\theta) = [\widetilde{Q}_{TT}(t_0)]^{-1} \left(\widetilde{Q}_T(t_0) + \widetilde{Q}_{T\boldsymbol{u}}(t_0)\delta\theta\right)$$

self-adaptive policy accounting for parameter update

Terminal Horizon (t_1) Optimization on CIFAR10



😀 fast convergence to desired t_1 without divergence (as in baseline)

😀 reduce runtime by 20% without hindering test-time accuracy

Conclusion

A new second-order optimizer for training Neural ODEs that

- grounded on optimal control theory
- generalizes first-order adjoint method while retaining the same constant ***O(1)*** memory (in depth)
- achieves strong empirical results (*e.g.*, convergence & test-time performance)
- opens up new applications and questions (*e.g.*, architecture optimization, implicit regularization)

Paper Poster Code

* Code is available at https://github.com/ghliu/snopt

Reference

Weinan et al., 2018, “A mean-field optimal control formulation of deep learning.”
Liu & Theodorou, 2019, “Deep learning theory review: An optimal control and dynamical systems perspective.”
Liu et al., 2021a, “DDPNOpt: Differential dynamic programming neural optimizer.”
Liu et al., 2021b, “Dynamic game-theoretic neural optimizer.”
Pontryagin et al., 1962, "The mathematical theory of optimal processes."
Finlay et al., 2020, "How to train your neural ode: the world of jacobian and kinetic regularization."